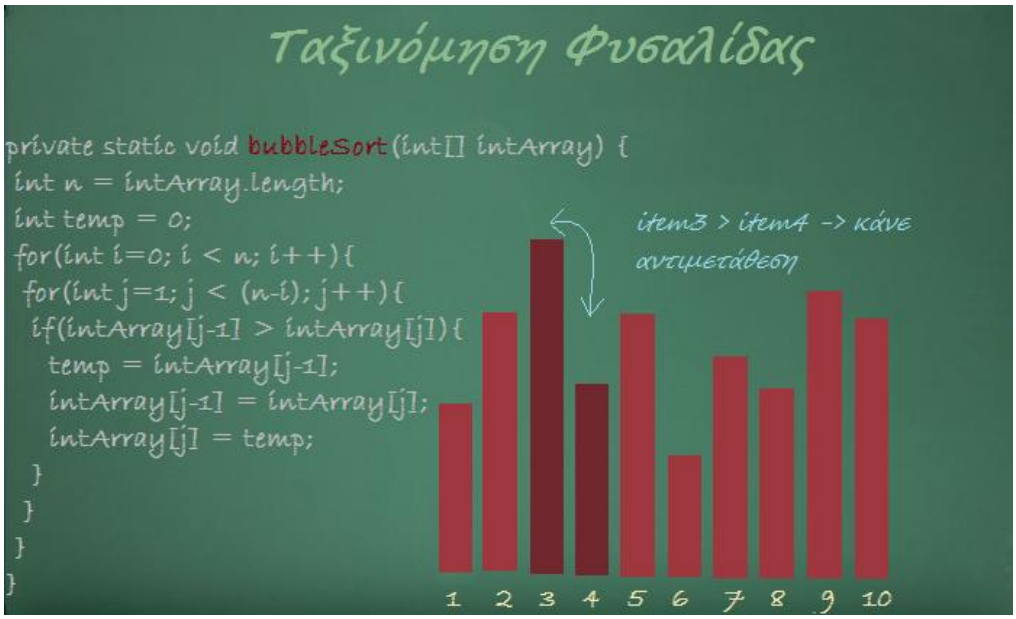
Αλεξάνδρειο Τεχνολογικό Εκπαιδευτικό Ίδρυμα Θεσσαλονίκης
Τμήμα Πληροφορικής

ΟΠΤΙΚΟΠΟΙΗΣΗ ΑΛΓΟΡΙΘΜΩΝ

(Algorithms Visualization)



ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Κριτσιμά Ανθούλα

Σούλιος Βασίλειος

Επιβλέπων Καθηγητής

Δεληγιάννης Ιγνάτιος

Θεσσαλονίκη 2012/2013

ΠΡΟΛΟΓΟΣ

Η εργασία αυτή εκπονήθηκε στα πλαίσια της πτυχιακής εργασίας με θέμα την «Οπτικοποίηση Αλγορίθμων» (Algorithms Visualization), με στόχο τη χρήση του ως παιδαγωγικό εργαλείο για την εκμάθηση των αλγορίθμων : Bubble Sort, Insertion Sort και Heap Sort. Η εφαρμογή αυτή είναι ένα εργαλείο απεικόνισης που μπορεί να χρησιμοποιηθεί για να συμπληρώσει την διδασκαλία των συγκεκριμένων αλγορίθμων, βοηθώντας έτσι τους σπουδαστές να κατανοήσουν καλύτερα την λειτουργίας τους.

Το λογισμικό γράφτηκε στην αντικειμενοστραφή γλώσσα προγραμματισμού Java και αναπτύχθηκε στην πλατφόρμα NetBeans IDE 7.3 Beta. Για να δείτε την εφαρμογή μπορείτε να επισκεφτείτε τον ιστότοπο http://aetos.it.teithe.gr/~ankri (η ιστοσελίδα υλοποιήθηκε στο Adobe DreamWeaver).

ΠΕΡΙΛΗΨΗ

Οι φοιτητές συνήθως έχουν δυσκολία στη κατανόηση των αφηρημένων εννοιών και των διαδικασιών όπως η διαδικαστική κωδικοποίηση των αλγορίθμων και των δομών δεδομένων. Ένας τρόπος να βελτιωθεί η κατανόηση των φοιτητών είναι η παροχή οπτικοποιήσεων οι οποίες καθιστούν τις αφηρημένες έννοιες πιο συγκεκριμένες. Η οπτικοποίηση λογισμικού χρησιμοποιεί τα γραφικά υπολογιστών για να μεταβιβάσει τη δομή και τη συμπεριφορά σύνθετων αλγορίθμων.

Στο πρώτο κεφάλαιο κάνουμε μια εισαγωγή στην οπτικοποίηση αλγορίθμων, παραθέτουμε τους διάφορους ορισμούς που χρησιμοποιούνται και κάνουμε μια ιστορική ανάδρομή της οπτικοποίησης αλγορίθμων.

Στο δεύτερο κεφάλαιο ασχολούμαστε με την οπτικοποίηση και την χρησιμότητά της στην εκπαιδευτική διαδικασία.

Στο τρίτο κεφάλαιο αναφέρονται οι τρεις αλγόριθμοι : bubble sort, insertion sort και heap sort. Περιγράφουμε αναλυτικά τον τρόπο λειτουργίας τους, την απόδοσή τους δίνοντας ένα παράδειγμα για την καθεμία ταξινόμηση.

Τέλος, στο τέταρτο κεφάλαιο αναλύουμε τον κώδικα της εφαρμογής «Οπτικοποίησης Αλγορίθμων» (Algorithms Visualization).

Πίνακας περιεχομένων

ΚΕΦΑΛΑΙΟ 1: Εισαγωγή στην Οπτικοποίηση Αλγορίθμων (Algorithm Visualization)

1.1 Οπτικοποίηση λογισμικού

Η Οπτικοποίηση (Visualization) είναι μία έννοια συχνά παρεξηγημένη, ακόμη και από τους έμπειρους χρήστες της αγγλικής γλώσσας. Επειδή περιέχει την λέξη ρίζα visual(οπτικό) που στα λατινικά σημαίνει όραση, πολλοί άνθρωποι πιστεύουν ότι η οπτικοποίηση αναφέρεται σε ότι αφορά το φτιάξιμο εικόνων. Στο Oxford English Dictionary (Simpson, 1989) υπάρχουν εφτά ορισμοί της λέξης visual. Οι έξι πιο κοινοί ορισμοί αναφέρονται σε εικόνες που μπορούν να δούνε οι άνθρωποι με τα μάτια τους. Ο έβδομος ορισμός προτείνει το σχηματισμό μιας διανοητικής εικόνας η οποία δεν συνδέεται απαραιτήτως με το οπτικό πεδίο κάποιου. Από αυτόν τον τελευταίο ορισμό η έννοια visualization(οπτικοποίηση) αποκτά το νόημα της.

1.1.1 Ορισμοί

Οπτικοποίηση Λογισμικού (Software Visualization, SV) είναι η οπτικοποίηση υπολογιστικών προγραμμάτων και αλγορίθμων. Η SV μπορεί να πάρει πολλές μορφές, διότι υπάρχει μια τεράστια ποικιλία από αισθητήριες εισόδους που μπορούν να σχηματίσουν μία διανοητική εικόνα ενός αντικειμένου. Ακόμη και η απλή πρόθεση ελέγχου των δομών στα προγράμματα, τα οποία οι περισσότεροι προγραμματιστές θεωρούν δεδομένα, είναι μία μορφή οπτικοποίησης (αν και είναι ένα αδύναμο είδος των σύγχρονων δεδομένων). Τα διαγράμματα ροής ήταν μία πρώιμη μορφή οπτικοποίησης, και η ιστορία της SV χρονολογείτε από τις μέρες του von Neumann στη δεκαετία του 1940.

Οπτικοποίηση Λογισμικού (SV) είναι η χρήση γραφικών αναπαραστάσεων υπολογιστών και σχεδικοκινήσεων τα οποία χρησιμοποιούνται για να εμφανίσουν και να παρουσιάσουν προγράμματα υπολογιστών, επεξεργασίες και αλγορίθμους. Συστήματα οπτικοποίησης λογισμικού μπορούν να χρησιμοποιηθούν στη διδασκαλία για να βοηθήσουν τους μαθητές να καταλάβουν πώς δουλεύουν οι αλγόριθμοι, καθώς και να χρησιμοποιηθούν στην ανάπτυξη προγραμμάτων ως ένα είδος βοήθειας για τους προγραμματιστές ώστε να καταλαβαίνουν τον κώδικα τους καλύτερα.

Στην προηγούμενη δεκαετία, υψηλής ποιότητας διεπιφάνειες (interfaces) έχουν καθιερωθεί σε ένα αυξανόμενο αριθμό περιοχών όπως π.χ. τα ηλεκτρονικά παιχνίδια και οι

εγκυκλοπαίδειες υπό μορφή CD-ROM. Όμως η συντριπτική πλειοψηφία των προγραμματιστών εισάγουν τον δικό τους κώδικα χρησιμοποιώντας ένα απλό φόντο μέσα σε ένα απλό παράθυρο και βλέπουν την εκτέλεση του κώδικα μέσω της δια χειρός εισαγωγής γραπτών δηλώσεων.

Η οπτικοποίηση λογισμικού (SV) έρχεται να αποκαταστήσει αυτήν την ανισορροπία χρησιμοποιώντας τυπογραφικές και γραφικές τεχνικές καθώς και τεχνικές οπτικοποίησης για να δείξουν τον κώδικα του προγράμματος, τα δεδομένα και τη ροή ελέγχου. Για να γίνει ο προγραμματισμός μια εμπειρία πολυμέσων (multimedia), η SV αφήνει τους προγραμματιστές και ερευνητές της επιστήμης των υπολογιστών ελεύθερους να εξερευνήσουν περισσότερο ενδιαφέροντα θέματα και να αντιμετωπίσουν προκλητικότερα προβλήματα.

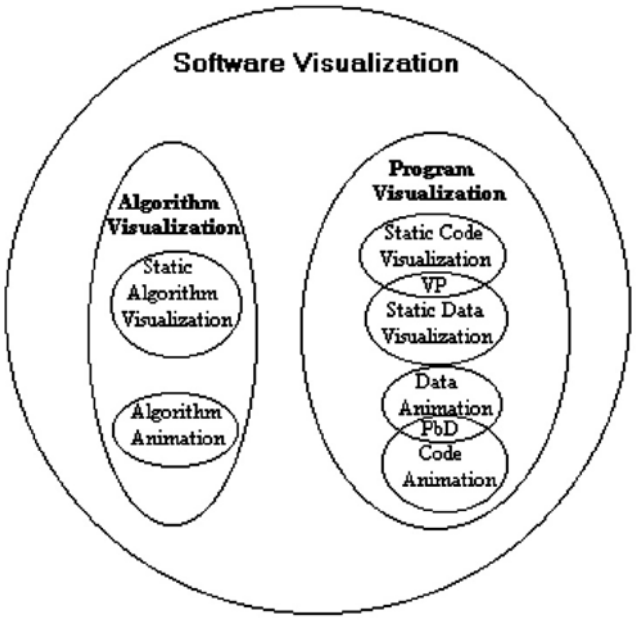
Η οπτικοποίηση λογισμικού περιλαμβάνει την ανάπτυξη και την εκτίμηση των μεθόδων για γραφικά αναπαριστάμενες διαφορετικές οπτικές λογισμικού, περιλαμβάνοντας την δομή τους, την αφηρημένη και συγκεκριμένη τους εκτέλεση και εξέλιξη τους.

Στην βιβλιογραφία για την SV συχνά συναντώνται οι φράσεις οπτικός προγραμματισμός, οπτικοποίηση προγράμματος και οπτικοποίηση αλγορίθμου να χρησιμοποιούνται ανταλλάξιμα και με ένα παραπλανητικό τρόπο. Επίσης συναντώνται οι φράσεις προγραμματισμός με επίδειξη, ή ο παλιότερος όρος, προγραμματισμός με παράδειγμα. Όλα αυτά είναι είδη οπτικοποίησης, αλλά οπτικοποιούν ένα συγκεκριμένο είδος πληροφορίας.

Ο όρος Οπτικοποίηση Προγράμματος (Program Visualization, PV) χρησιμοποιήθηκε στα τέλη της δεκαετίας του 80 (ειδικά στις ταξινομίες του Mayers(1986, 1988, 1990) για να αναφερθεί σε αυτό που αποκαλούμε Mayer SV. Είναι προτιμότερο να διαφοροποιηθεί η PV από άλλα θέματα μέσα στην SV διότι υποδηλώνει μία σύνδεση με το πρόγραμμα (κατώτερο επίπεδο) σε αντίθεση με το αλγόριθμο (υψηλότερο επίπεδο). Πράγματι, οι ταξινομίες του Mayers αναφέρονται σε δύο ορθογώνιες διαστάσεις κώδικα "εναντίον" δεδομένων και στατικών "εναντίον" δυναμικών τα οποία ακόμη σχηματίζουν μια βάση καταμερισμού(διαχωρισμού) της αρχής. Συστήματα που οπτικοποιούν κώδικα προγράμματος ή δεδομένα(ή συνδυασμό και των δύο) σκοπεύουν να απασχολήσουν μέχρι ένα σημείο μία στατική σε μία δυναμική(animated)διάσταση. Η οπτικοποίηση στατικού κώδικα ίσως μπορεί να συμπεριλάβει κάποιο είδος χάρτη προγράμματος όπως το SEE Program Visualizer (Baecker and Marcus on Printing andPublishing C Programs or

Baecker 1990a), ενώ ένα παράδειγμα οπτικοποίησης στατικών δεδομένων μπορεί να εμφανιστεί ως ένα διάγραμμα "κουτιών και τόξων" (boxes and arrows)μιας δομής διασυνδεδεμένης λίστας δεδομένων που εμφανίζει τα περιεχόμενα. Ένα παράδειγμα μιας δυναμικής οπτικοποίησης δεδομένων θα μπορούσε να δείξει το ίδιο διάγραμμα με τα τόξα και τα περιεχόμενα να αλλάζουν δυναμικά καθώς το πρόγραμμα τρέχει, ενώ μια απλή οπτικοποίηση κινούμενου(δυναμικού) κώδικα θα μπορούσε να υπογραμμίζει τις γραμμές του κώδικα καθώς αυτές εκτελούνται.

Για υψηλότερου επιπέδου περιγραφές λογισμικού (π.χ. αλγορίθμων) υπάρχει επίσης μία διάσταση από το στατικό στο δυναμικό. 1ιαγράμματα ροής είναι ένα απλό παράδειγμα στατικών οπτικοποιήσεων αλγορίθμων, καθώς οι περισσότερο ενδιαφέρουσες (τουλάχιστον από μία εκπαιδευτική προσέγγιση) οπτικοποιήσεις είναι εκείνες που χρησιμοποιούν οπτικοποίηση για να παρουσιάσουν πώς λειτουργεί ο αλγόριθμος. Αυτή η δυναμική οπτικοποίηση αλγορίθμου ονομάζεται συχνά Algorithm Animation (Σχεδιοκίνηση Αλγορίθμου), και μερικά από τα καλύτερα παραδείγματα είναι το φιλμ "Sorting Out Sorting" (SOS) του Baecker (1981), το BALSA του Brown(1988a), το Zeus (Brown, 1991) και το σύστημα TANGO του Stasko(1990b). Η παρακάτω εικόνα δείχνει την σχέση μεταξύ των διαφόρων τύπων της SV και τα σχετικά πεδία.



Εικόνα 1.1 Ένα διάγραμμα Venn παρουσιάζει τους όρους στην βιβλιογραφία της Οπτικοποίησης λογισμικού. Τα μεγέθη δεν είναι σχετικά και οι μόνες τομές που παρουσιάζονται είναι εκείνες για τον οπτικό προγραμματισμό (VP) και τον προγραμματισμό με επίδειξη (PbD).(προσαρμοσμένο από τους Price, Baecker & Small, 1998)

Ο οπτικός προγραμματισμός (Visual Programming VP). Η κύρια διαφορά μεταξύ του VP και της SV είναι ο επιδιωκόμενος στόχος: ο VP έχει σκοπό να κάνει πιο εύκολο τον προσδιορισμό των προγραμμάτων χρησιμοποιώντας μια γραφική (η "οπτική") σημειογραφία, καθώς η SV έχει σκοπό να κάνει εύκολη την κατανόηση των προγραμμάτων και των αλγορίθμων χρησιμοποιώντας ποικίλες τεχνικές. Φυσικά μια οπτική περιγραφή μπορεί από μόνη της να δίνει πληροφορία σχετικά με ένα πρόγραμμα εφόσον είναι ένα είδος στατικής οπτικοποίησης κώδικα/δεδομένων, εκ τούτου πολλά VP συστήματα παρέχουν ένα είδος SV.

Μια περιοχή που σχετίζεται με τον VP που επίσης παρουσιάζει ομοιότητα με την SV είναι ο Προγραμματισμός μέσω Επίδειξης (Programming by Demonstration PbD), που μερικές φορές σε παλιότερες βιβλιογραφίες ονομάζεται προγραμματισμός μέσω παραδειγμάτων (Programming by Example). Η ιδέα είναι ότι οι χρήστες δεν είναι απαραίτητο να έχουν ανεπτυγμένη προγραμματιστική δεξιότητα για να δομήσουν ένα πρόγραμμα , αλλά να μπορούν να επιδείξουν ένα παράδειγμα και να το σύστημα να συμπεράνει ένα πρόγραμμα. Γι' αυτό το λόγο , αν οι χρήστες γνωρίζουν πώς να εκτελέσουν μια εργασία στον υπολογιστή, τότε αυτό θα πρέπει να είναι επαρκές για να δημιουργηθεί ένα πρόγραμμα για να εκτελέσει την εργασία. Για παράδειγμα μπορεί να δείξουν πως χρησιμοποιούνται μερικά δεδομένα δείγματος. Αυτό συνήθως επιτυγχάνεται χρησιμοποιώντας μια γραφική διεπιφάνεια, η οποία , όπως το VP, κάνει το αποτέλεσμα ένα είδος από SV.

Οι Stasko και Wehrli εισήγαγαν τον όρο Υπολογιστική Οπτικοποίηση (Computation Visualization) (Stasko 1993c) για συμπεριλάβουν την οπτικοποίηση στις περιπτώσεις της απόδοσης του υλισμικού, που συχνά αποκαλείται Οπτικοποίηση Απόδοσης (Performance Visualization). Αυτό το είδος οπτικοποίησης είναι σημαντικό σε περιπτώσεις όπως το αντιστάθμισμα φόρτου (load balancing) ή στη βελτίωση της απόδοσης σε αρχιτεκτονικές πολυεπεξεργαστών.

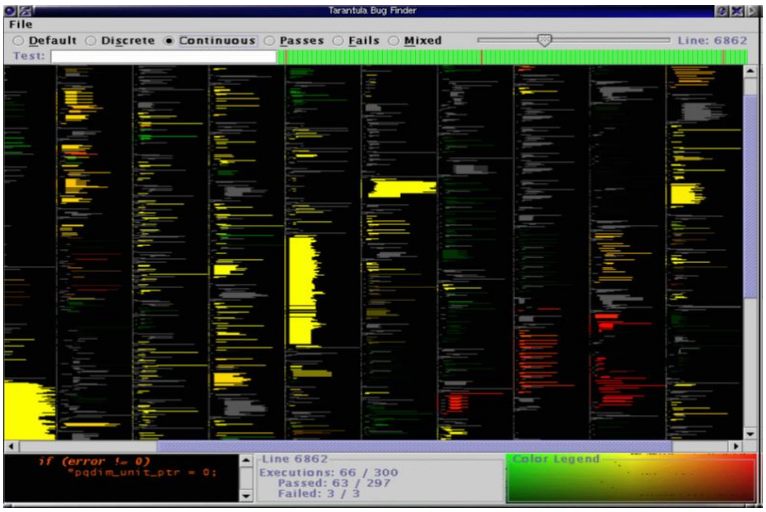
Στην εξέταση των συστημάτων της SV συναντούμε ένα αριθμό ατόμων που ενεργούν με διαφορετικούς ρόλους, μερικοί από τους οποίους αναλαμβάνουν πολλαπλούς ρόλους. Υπάρχει συνήθως ο προγραμματιστής που έγραψε το αρχικό πρόγραμμα ή τον αλγόριθμο που οπτικοποιείται. Οι προγραμματιστές ίσως να μη γνωρίζουν ότι τα προγράμματα τους πρόκειται να οπτικοποιηθούν όταν τα έγραψαν. Ένα πολύ σημαντικό πρόσωπο είναι ο υπεύθυνος ανάπτυξης λογισμικού της SV που έγραψε το λογισμικό που επιτρέπει στα προγράμματα ή στους αλγορίθμους να οπτικοποιηθούν. Ένας άλλος σημαντικός ρόλος

είναι ο “visualizer” ή “animator” ο οποίος παίρνει το πρόγραμμα ή τον αλγόριθμο και το σύστημα SV, και ορίζει πώς η οπτικοποίηση μπορεί να συνδεθεί ή να εφαρμοστεί στο πρόγραμμα. Εντέλει, το άτομο από το οποίο γράφτηκε η οπτικοποίηση είναι ο χρήστης ή θεατής, ο οποίος μπορεί να δει την οπτικοποίηση στατικά ή να διαδράσει με αυτή και να πλοηγηθεί σε αυτή.

Η τεχνολογία της οπτικοποίησης μπορεί να χρησιμοποιηθεί για να αναπαραστήσει γραφικά ποικίλες έννοιες στην επιστήμη των υπολογιστών. Η οπτικοποίηση λογισμικού εμφανίστηκε στα τέλη της δεκαετίας του 1980 με σκοπό την δημιουργία και διαδραστική εξερεύνηση γραφικών αναπαραστάσεων των εννοιών της επιστήμης των υπολογιστών. Αυτό βγαίνει και ως συμπέρασμα από τον Stephan Diehl, ο οποίος διαχωρίζει την οπτικοποίηση λογισμικού σε τρείς πτυχές, την δομή, την συμπεριφορά και την εξέλιξη. [27]

1.1.2 Παράδειγμα τεχνολογίας οπτικοποίησης λογισμικού (SV)

Ταραντούλα(Tarantula) [40]:
Στο GVU και στο Κολέγιο Προγραμματισμού (GVU Center and the College of Computing), υπάρχει ένα μάθημα αποφοίτων που έχει ως αντικείμενο την Software Visualization. Η “Ταραντούλα”, παρακάτω, είναι ένα από τα ερευνητικά project σχετικά με την SV: Η Ταραντούλα είναι ένα σύστημα οπτικοποίησης το οποίο παραθέτει τα αποτελέσματα τρεχόντων ακολουθιών από τεστ έναντι συστημάτων λογισμικού. Με την επίδειξη των κομματιών του κώδικα που εκτελούνται από επιτυχημένα και αποτυχημένα τεστ, το σύστημα βοηθά τους χρήστες να αναγνωρίσουν τα λάθη στα προγράμματά τους.



Εικόνα 1.2 Παράδειγμα στιγμιότυπου οθόνης.

1.2 Αλγόριθμοι και Οπτικοποίηση

Εισαγωγή

1.2.1 Ορισμός της οπτικοποίησης αλγορίθμων και οι λειτουργίες της

Τι είναι "*οπτικοποίηση αλγορίθμου*";

Οπτικοποίηση αλγορίθμου (Algorithm Visualization "AV") είναι η διαδικασία αφαίρεσης των δεδομένων, των λειτουργιών και της σημασιολογίας ενός προγράμματος και η δημιουργία δυναμικών γραφικών αναπαραστάσεων όψεων αυτών των αφηρημένων εννοιών (Stasko, 1990).

Η οπτικοποίηση αλγορίθμου, AV, είναι μια σημαντική υπο-περιοχή της ευρύτερης περιοχής οπτικοποίησης λογισμικού. Όπως οι Price, Baecker και Small (1998) εξηγούν, "οπτικοποίηση" σημαίνει τη "δύναμη ή διαδικασία παρουσίασης μιας διανοητικής εικόνας ή της όψης από κάτι όχι πραγματικά παρόν σε θέα". Αυτοί οι συντάκτες καθορίζουν την οπτικοποίηση λογισμικού ως "χρήση των τεχνών της τυπογραφίας, του γραφικού σχεδίου, της σχεδιοκίνησης (animation), και της κινηματογραφίας με την σύγχρονη τεχνολογία της αλληλεπίδρασης ανθρώπου-υπολογιστή και των γραφικών υπολογιστών για να διευκολύνουν και την ανθρώπινη κατανόηση και την αποτελεσματική χρήση του λογισμικού υπολογιστών". Η οπτικοποίηση λογισμικού, επομένως, αναφέρεται στα βασισμένα σε υπολογιστή περιβάλλοντα, όπου οι κατάλληλες οπτικοποιήσεις χρησιμοποιούνται προκειμένου να μεταβιβαστεί στο χρήστη μια βαθύτερη κατανόηση των μετασχηματισμών των δομών δεδομένων και των διαδικασιών λογισμικού.

Η οπτικοποίηση αλγορίθμου περιλαμβάνει τις οπτικοποιήσεις που εξετάζουν τους αλγορίθμους. Στον τομέα της πληροφορικής, τα συστήματα AV κάνουν διαθέσιμες στους σπουδαστές πολλαπλές, δυναμικές και διαλογικές οπτικοποιήσεις των αλλαγών στις οποίες υποβάλλονται τα σύνολα δεδομένων όταν κάποιος αλγόριθμος εφαρμόζεται σε αυτά. Η οπτικοποίηση αλγορίθμου περιλαμβάνει και τη χρήση των στατικών οπτικών αναπαραστάσεων (π.χ. διαγράμματα ροής) και δυναμικών (animated). Στην τελευταία περίπτωση ο όρος " οπτικοποίηση αλγορίθμου" χρησιμοποιείται συνήθως, αναφερόμενος συγκεκριμένα στη χρήση των δυναμικών οπτικών αναπαραστάσεων για την οπτικοποίηση των υψηλού επιπέδου αφαιρέσεων που περιγράφουν το λογισμικό, δηλ. οι αλγόριθμοι. Μία οπτικοποίηση αλγορίθμου δημιουργεί αρχικά μια αφηρημένη γραφική αντιπροσώπευση του συνόλου των δεδομένων, χαρτογραφώντας τις τρέχουσες τιμές των

μεταβλητών που χρησιμοποιούνται στον αλγόριθμο επάνω στα σχετικά γραφικά στοιχεία (παραδείγματος χάριν σημεία, ραβδιά, κύκλοι ή ελλειψοειδή). Έπειτα, αυτά τα στοιχεία γίνονται δυναμικά, αντιπροσωπεύοντας τις διαδικασίες μεταξύ των επιτυχημένων καταστάσεων στην εκτέλεση του αλγορίθμου. Η οπτικοποίηση ενός αλγόριθμου έχει ως στόχο να προωθήσει την καλύτερη κατανόηση της δομής, των περιπλοκών, των ανεπαρκειών και των πλεονεκτημάτων του αλγορίθμου από τους σπουδαστές, επιτρέποντας ακόμη και περαιτέρω βελτιστοποίηση.

Η οπτικοποίηση αλγορίθμου σχεδιάστηκε για γα βοηθήσει τους χρήστες να κατανοήσουν έναν αλγόριθμο, να εκτιμήσουν ένα υπάρχον πρόγραμμα και να διορθώσουν ένα πρόγραμμα.

Ένα πρόγραμμα υπολογιστή περιέχει δομές δεδομένων του αλγορίθμου που εφαρμόζει. Κατά τη διάρκεια της εκτέλεσης του προγράμματος , οι εκτιμήτριες της δομής δεδομένων αλλάζουν σύμφωνα με τον εφαρμοζόμενο αλγόριθμο. Η οπτικοποίηση αλγορίθμων χρησιμοποιεί μία γραφική αναπαράσταση των εσωτερικών δομών δεδομένων του εφαρμοζόμενου αλγορίθμου στο πρόγραμμα, και δείχνει τις αλλαγές των δομών δεδομένων για κάθε κατάσταση(βήμα), για να δείξει την εκτέλεση και την δυναμική συμπεριφορά του προγράμματος αυτού. Με μια τέτοια παρουσίαση, οι χρήστες μπορούν να παρακολουθήσουν την εκτέλεση του προγράμματος βήμα προς βήμα και έχουν την δυνατότητα να διερευνήσουν τις λεπτομέρειες του αλγορίθμου και να αποκτήσουν καλύτερη κατανόηση έπ’ αυτού.

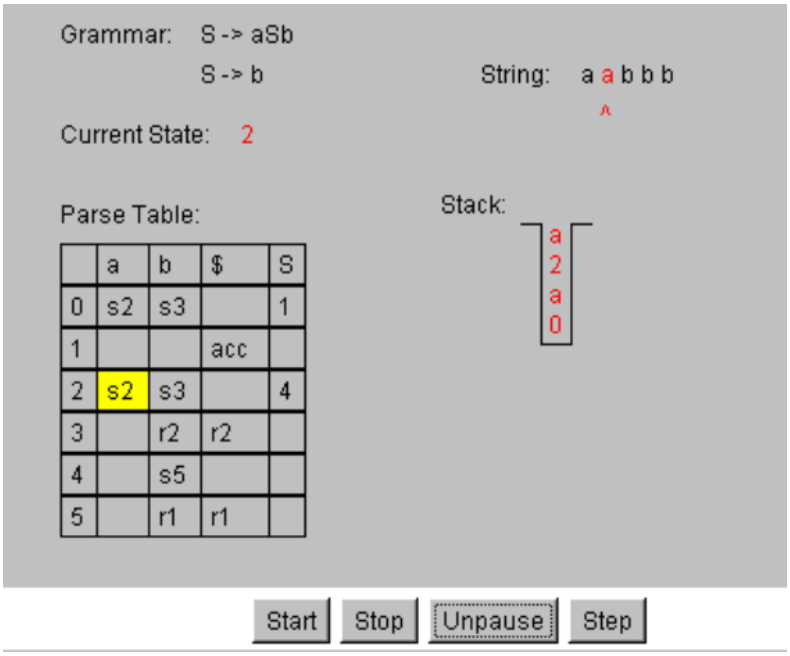
Η οπτικοποίηση αλγορίθμου μπορεί επίσης να χρησιμοποιηθεί για να εκτιμήσει υπάρχοντα προγράμματα παρέχοντας δυναμική όψη σε περιεχόμενα συστήματος για να βοηθήσει στην οπτικοποίηση απόδοσης συστήματος.

Εκτός από την παροχή βοήθειας στους χρήστες για να κατανοήσουν έναν αλγόριθμο, η οπτικοποίηση αλγορίθμων μπορεί να χρησιμοποιηθεί για να κάνει τη διόρθωση ενός προγράμματος ευκολότερη. Για να χρησιμοποιήσουν την οπτικοποίηση αλγορίθμων κατά την διόρθωση ενός προγράμματος, οι χρήστες σχολιάζουν συγκεκριμένες καταστάσεις στα προγράμματα τους για να εξάγουν διαταγές οπτικοποίησης, οι οποίες είναι μεταγενέστερα τοποθετημένες στο σύστημα οπτικοποίησης του αλγορίθμου για να ξεκινήσει την παραγωγή της οπτικοποίησης. Οι χρήστες μπορούν να δουν πως το πρόγραμμα τους εκτελείται, τις αξίες των δεδομένων σε κάθε βήμα και πως κάθε βήμα επιδρά στα

μεταγενέστερα βήματα στο πρόγραμμα. Αυτό βοηθά τους χρήστες να ανιχνεύσουν τυχόν λάθη, που μπορεί να εμφανιστούν στα προγράμματά τους.

1.2.2 Παράδειγμα οπτικοποίησης αλγορίθμων

Παρακάτω παρατίθεται ένα παράδειγμα οπτικοποίησης αλγορίθμου, και συγκεκριμένα ενός αλγορίθμου ανάλυσης LR. Ένα πρόγραμμα εφαρμόζει έναν αλγόριθμο ανάλυσης LR, για να αναλύσει μια εισαγόμενη συμβολοσειρά(string). Το πρόγραμμα περιέχει τρεις δομές δεδομένων, ένα πίνακα ανάλυσης μία στοίβα και μία εισαγόμενη συμβολοσειρά. Όταν το πρόγραμμα εκτελείται , αποκτά οδηγίες ψάχνοντας να βρει εισόδους στον πίνακα ανάλυσης και εκτελεί ενέργειες στην στοίβα για να αναλύσει τη συμβολοσειρά. Η οδηγία, το εισαγόμενο σύμβολο string και τα στοιχεία στην στοίβα σε κάθε φάση αλλάζουν κάθε στιγμή καθώς ο αλγόριθμος εκτελείται. Θα είναι δύσκολο για ένα σπουδαστή να συλλάβει τις κρυμμένες αλλαγές αν του /της παρουσιάζονται μόνο στατικό κείμενο και εικόνα. Η οπτικοποίηση αλγορίθμων , παρόλα αυτά, μπορεί να δείξει όλη την πληροφορία και τις πρόσφατες αλλαγές σε μία γραφική όψη ταυτοχρόνως. Ένα παράδειγμα στιγμιότυπου από μία οπτικοποίηση ανάλυσης LR φτιαγμένο από ένα σύστημα οπτικοποίησης αλγορίθμων, φαίνεται στο παρακάτω διάγραμμα.



	a	b	$	S
0	s2	s3		1
1			acc	
2	s2	s3		4
3		r2	r2	
4		s5		
5		r1	r1	

Εικόνα 1.3 Ένα στιγμιότυπο οπτικοποίησης ανάλυσης LR

Το διάγραμμα δείχνει την προοδευτική εκτέλεση του προγράμματος που εφαρμόζει αλγόριθμο ανάλυσης LR για να αναλύσει την εισαγόμενη συμβολοσειρά. Στο διάγραμμα,

οι δομές δεδομένων παρουσιάζονται σαν γραφικά αντικείμενα στην οπτικοποίηση. Όταν η οπτικοποίηση εκτελείται, κάθε βήμα της ανάλυσης της εισαγόμενης συμβολοσειράς συμπεριλαμβανομένου και του πρόσφατου συμβόλου ανάλυσης συμβολοσειράς, πρόσφατη οδηγία στον πίνακα ανάλυσης, ωθώντας(σπρώχνοντας) στοιχεία μέσα ή απωθώντας στοιχεία έξω από την στοίβα φαίνονται εν δράση ταυτόχρονα.

1.3 Ιστορία της οπτικοποίησης αλγορίθμων

Η ιστορία της ανάπτυξής της οπτικοποίησης αλγορίθμων έχει ξεκινήσει εδώ και δύο δεκαετίες. Αλλά η πρώτη εξέλιξη της οπτικοποίησης αλγορίθμων ανιχνεύεται στο 1966, όταν μία γλώσσα οπτικοποίησης συνδεδεμένης λίστας, η L[6], δημιουργήθηκε από τον Ken Knowlton στα εργαστήρια Bell Telephone. Η κύρια ανάπτυξη της οπτικοποίησης αλγορίθμων ξεκινά στις αρχές της δεκαετίας του 80.

Το 1981 ένα βίντεο, με την ονομασία “Sorting Out Sorting”, δημιουργήθηκε από τον Ronald Baecker στο Πανεπιστήμιο του Τορόντο. Αυτό αναγνωρίστηκε ως η αφετηρία της οπτικοποίησης αλγορίθμων. Από τότε, οι εκπαιδευτές έχουν υιοθετήσει την οπτικοποίηση αλγορίθμων ως μέσο για να διδάξουν αλγορίθμους. Μεταξύ του 1980 και των αρχών του 1990, αναπτύχθηκαν δύο δημιουργικά συστήματα, τα οποία έχουν σημαντική επίδραση σε όσα άλλα συστήματα ακολούθησαν. Αυτά τα συστήματα ήταν το BALSA-I (Brown ALgorithm Simulator and Animator, Brown 1984) και το TANGO (Transition-based Animation GeneratiOn, Stasko 1990).

Το BALSA-I ήταν το πρώτο ευρέως γνωστό σύστημα οπτικοποίησης αλγορίθμων. Αναπτύχθηκε από τους Marc Brown και Robert Sedgewick στο Πανεπιστήμιο Brown. Το BALSA-I είναι ένα διαδραστικό σύστημα οπτικοποίησης αλγορίθμων το οποίο υποστηρίζει πολλαπλές ταυτόχρονες όψεις από τις δομές δεδομένων ενός αλγορίθμου και μπορεί να επιδείξει ταυτόχρονα την εκτέλεση πολλαπλών αλγορίθμων. Η εξέλιξή του κινητοποίησε και άλλους ερευνητές να λάβουν μέρος στην ανάπτυξη και άλλων συστημάτων οπτικοποίησης αλγορίθμων. Επίσης προέβαλλε και έναν αριθμό από ζητήματα τα οποία μπορούν να βελτιώσουν καλύτερα το σύστημα για το οποίο ενδιαφέρονται και άλλοι ερευνητές που βρίσκονται στον πεδίο της οπτικοποίησης αλγορίθμων.

Ένα άλλο σύστημα, το TANGO, αναπτύχθηκε από τον John Stasko ο οποίος ήταν στο Πανεπιστήμιο Brown όταν το σύστημα αναπτύχθηκε. Η εμφάνιση του TANGO μας

εισάγει στο παράδειγμα "μετάβαση μονοπατιού"(path-transition) για την σχεδίαση μιας οπτικοποίησης και συστήνει ένα νέο εννοιολογικό πλαίσιο το οποίο υιοθετείται από πολλά μεταγενέστερα συστήματα ως η θεμελιώδης αρχιτεκτονική των συστημάτων οπτικοποίησης αλγορίθμων.

Από τότε που αναπτύχθηκαν το BALSA και το TANGO, αναπτύχθηκαν συστήματα απόγονοι αυτών των δύο αξιοσημείωτων συστημάτων. Το BALSA-I έχει ένα σύστημα απόγονο, το BALSA-II (Brown, 1988). Το BALSA-II είναι ένα σύστημα οπτικοποίησης αλγορίθμων το οποίο χρησιμοποιεί εικόνες με πολλαπλές όψεις και παρέχει δυνατότητα προγραμματισμού με γλώσσα script. Το TANGO, από την άλλη πλευρά, έχει έναν αριθμό από συστήματα απογόνους.

Το XTANGO (Stasko, 1992) είναι ένας απευθείας απόγονος του συστήματος TANGO. Είναι μία έκδοση παραθύρου X του TANGO και χρησιμοποιεί ένα παράδειγμα μετάβασης μονοπατιού για να επιτύχει ομαλή οπτικοποίηση. Το XTANGO έχει ένα άλλο σύστημα απόγονο, το POLKA και το αντίστροφό του, το Samba (πληροφορίες σχετικά με το POLKA και το Samba μπορεί να βρεθεί στην ιστοσελίδα http://www.cc.gatech.edu/gvu/softviz/parviz/polka.html). Το POLKA είναι σχεδιασμένο για να χτίζει ταυτόχρονη οπτικοποίηση για παράλληλα προγράμματα. Είναι ένα δύο διαστάσεων προσανατολισμένο στο αντικείμενο σύστημα οπτικοποίησης αλγορίθμων και έχει εξελιχθεί σε ένα σύστημα τριών διαστάσεων, το POLKA 3-D. Το POLKA 3-D παρέχει όψεις τριών διαστάσεων και σχήματα τριών διαστάσεων όπως κώνοι, σφαίρες, κύβοι και άλλα. Οι χρήστες δεν είναι απαραίτητο να έχουν προηγούμενη γνώση γραφικών υπολογιστών τριών διαστάσεων για να χρησιμοποιήσουν το POLKA 3-D. Το Samba είναι ένας μεταφραστής (διερμηνέας) διαδραστικής οπτικοποίησης ο οποίος διαβάζει εντολές ASCII και εκτελεί τις αντίστοιχες ενέργειες της οπτικοποίησης. Υπάρχει επίσης και μία java έκδοση της Samba η οποία ονομάζεται Jsamba (βλέπε http://www.cc.gatech.due/gvu/softviz/parviz/samba.html).

Άλλα δημοφιλή συστήματα οπτικοποίησης είναι τα Zeus, Leonardo, CATAI, Mocha. Το Zeus (Brown, 1991) αναπτύχθηκε στο Πανεπιστήμιο Brown και μαζί με τα BALSA και BALSA-II, θεωρήθηκε ένα από τα πρώτα σημαντικά διαδραστικά συστήματα οπτικοποίησης λογισμικού(SV). Υποστηρίζει πολλαπλές συγχρονισμένες όψεις και επιτρέπει στους χρήστες να μετατρέπουν αυτές τις όψεις και να αλλάζουν την αναπαράσταση δεδομένων ενός αντικειμένου.

Το Zeus εφαρμόζεται σε ένα πολύ-επεξεργαστικό περιβάλλον, έτσι ώστε να μπορεί να σχεδιοκινεί παράλληλα προγράμματα. Ο Leonardo (βλέπε http://www.dis.uniromal.it/~demetres/Leonoardo/Overview/Overview.html) είναι ένα ολοκληρωμένο περιβάλλον για την ανάπτυξη και την οπτικοποίηση προγραμμάτων σε γλώσσα C. Ολοκληρώνει την ανάπτυξη και την οπτικοποίηση προγραμμάτων σε γλώσσα C μέσα στο ίδιο περιβάλλον μαζί με εκτεταμένο έλεγχο χρήστη στην οπτικοποίηση. Το CATAI (βλέπε http://wonderland.dia.unisa.it/catai/) είναι ένα σύστημα σχεδιοκινήσεων το οποίο σχεδιοκινεί προγράμματα σε C++. Βασίζεται σε διαδεδομένες τεχνολογίες αντικειμένου και επιτρέπει σε αρκετούς χρήστες να μοιράζονται την ίδια οπτικοποίηση μέσα σε μία εικονική τάξη αφηρημένης έννοιας. Οι οπτικοποιήσεις CATAI παρουσιάζονται από νοήμονες Java clients. Οι επικοινωνίες συγχρονίζονται ανάμεσα στους clients της οπτικοποίησης και στον δυναμικό(κινούμενο) αλγόριθμο επιβεβαιώνονται από τον εξυπηρέτη οπτικοποίησης Java ο οποίος χρησιμοποιεί τεχνολογία CORBA. Το Mocha (βλέπε http://www.cs.brown.edu/people/jib/Mocha.html) είναι ένα διαδεδομένο μοντέλο με μία αρχιτεκτονική πελάτη-εξυπηρέτη η οποία προαιρετικά χωρίζει τα περιεχόμενα του λογισμικού ενός τυπικού συστήματος οπτικοποίησης αλγορίθμου. Ξεπερνά το πρόβλημα που κληρονομείται από το παράθυρο X και το μοντέλο Java. Στο μοντέλο Mocha, μόνο ο κώδικας της διεπιφάνειας εξάγεται στη μηχανή χρήστη, καθώς ο αλγόριθμος εκτελείται σε ένα εξυπηρέτη ο οποίος τρέχει στη μηχανή του παροχέα.

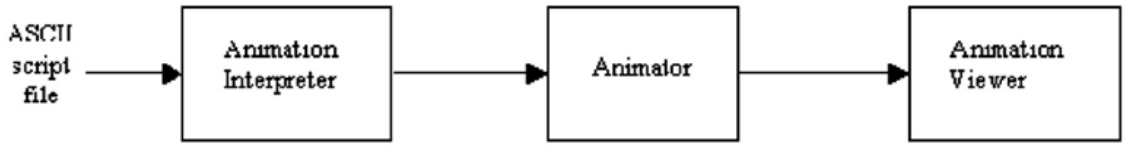
Τα πρώτα συστήματα οπτικοποίησης αλγορίθμων ήταν εξαρτώμενα από πλατφόρμα. Με την άφιξη της νέας τεχνολογίας, την δημοτικότητα του διαδικτύου(World Wide Web) και την εξέλιξη της γλώσσας προγραμματισμού java, οι σχεδιαστές προσανατολίστηκαν στο να αναπτύσσουν on-line συστήματα οπτικοποίησης αλγορίθμων, τα οποία έχουν το πλεονέκτημα της ανεξαρτησίας από πλατφόρμα και της ανοιχτής προσβασιμότητας πάνω σε παλιότερα συστήματα. Μερικοί σχεδιαστές επίσης ενσωμάτωσαν την χρήση πολυμέσων στα συστήματα τους. Η χρήση συστημάτων οπτικοποίησης αλγορίθμων δεν περιορίζεται πλέον σε παραδοσιακές τάξεις διδασκαλίας ή σε εργαστηριακή διδασκαλία αλλά έχει τώρα επεκταθεί σε εκπαίδευση από απόσταση.

Ένας μεγάλος αριθμός από συστήματα οπτικοποίησης αλγορίθμων έχει αναπτυχθεί στις τελευταίες δύο δεκαετίες. Τα περισσότερα από αυτά τα συστήματα που αναφέρθηκαν παραπάνω ανήκουν στα πιο δημοφιλή και εκλεπτυσμένα συστήματα που χρησιμοποιούνται. Έχουν αναπτυχθεί και χρησιμοποιηθεί από τους σχεδιαστές τους για

εκπαιδευτικούς σκοπούς ή για εμπειρική έρευνα. Μερικά από αυτά τα συστήματα έχουν μια πολύπλοκη αρχιτεκτονική και απαιτούν συγκεκριμένη τεχνολογία να τα υποστηρίζει για να "τρέξουν". Δεν χρησιμοποιήσαμε κανένα από αυτά τα συστήματα για να χτίσουμε τα δικά μας διαδραστικά συστήματα ανάλυσης οπτικοποίησης. Αντιθέτως έχουμε εκτιμήσει άλλα υπάρχοντα γενικού σκοπού συστήματα οπτικοποίησης αλγορίθμων τα οποία είναι μικρότερα σε μέγεθος και απλούστερης αρχιτεκτονικής. Στην επόμενη ενότητα , ερευνούμε τη χρήση της οπτικοποίησης αλγορίθμων στην εκπαίδευση και αναλύουμε την αποτελεσματικότητα τους.

1.4 Χαρακτηριστικά συστήματα και τεχνολογίες ανάπτυξης οπτικοποίησης.

Υπάρχει ένας αριθμός από συστήματα οπτικοποίησης αλγορίθμων γραμμένα σε java με αρχιτεκτονική όπως φαίνεται στην εικόνα.



Εικόνα 1.4 Αρχιτεκτονική συστήματος οπτικοποίησης αλγορίθμου

1.4.1 Παραδείγματα χαρακτηριστικών συστημάτων

Παρακάτω θα παραθέσουμε χαρακτηριστικά συστήματα οπτικοποίησης και θα εκτιμήσουμε κάθε ένα από αυτά παρουσιάζοντας τα πλεονεκτήματα και τα μειονεκτήματα τους.

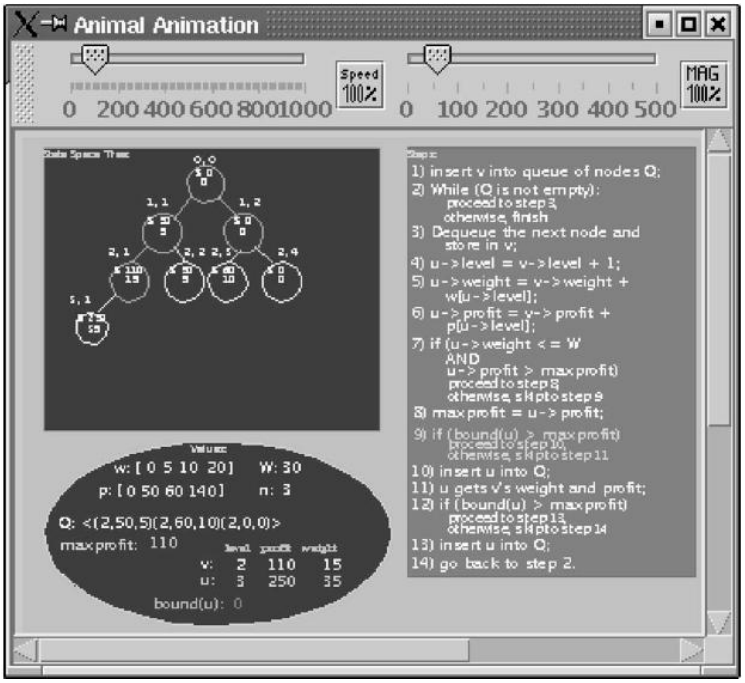
Γενικού σκοπού εργαλεία οπτικοποίησης αλγορίθμων

ANIMAL – ANIMALSCRIPT – ANIMAL-FARM

Το σύστημα δυναμικής παρουσίασης αλγορίθμων ANIMAL (***A**dvanced **N**avigation and **I**nteractive **M**odelling of **A**nimations in **L**ectures*) (Rößling, Schüler & Freisleben, 2000; Rößling & Freisleben, 2002), από το όνομα φαίνεται ότι η εφαρμογή δημιουργήθηκε ειδικά για σκοπό τις διαλέξεις και γενικά για εκπαιδευτική χρήση.

Σχεδιάστηκε για να καλύψει τους τρεις από τους τέσσερις ξεχωριστούς ρόλους που απαιτούνται σε ένα σύστημα οπτικοποίησης. Αυτοί οι ρόλοι είναι: ο αρχικός προγραμματιστής του αλγορίθμου, ο σχεδιαστής των εργαλείων της δυναμικής παρουσίασης, ο παραγωγός που δημιουργεί την δυναμική παρουσίαση και ο τελικός αποδέκτης που παρακολουθεί την παρουσίαση.

Τα κύρια χαρακτηριστικά του συστήματος ANIMAL είναι η δυναμική επεκτασιμότητα, η *διεθνοποίηση των συστατικών του γραφικού περιβάλλοντος αλληλεπίδρασης με το χρήστη* (*internation-alization of GUI components*) και του περιεχομένου της δυναμικής παρουσίασης, αντίστροφη εκτέλεση της δυναμικής παρουσίασης και ο ευέλικτος τρόπος εισαγωγής και εξαγωγής πληροφοριών. Επίσης άλλα βασικά γνωρίσματα του ANIMAL είναι η *δυναμική επαναδιαμόρφωση* (*dynamic reconfiguration*), η κλιμάκωση του μεγέθους της επίδειξης, η εξαγωγή πολλαπλών μορφών αρχείων και ο πλήρης έλεγχος τύπου βίντεο, *Εικόνα 1.4α.*



Εικόνα 1.4α Στιγμιότυπο του παραθύρου δυναμικής παρουσίασης του ANIMAL

Η δυναμική παρουσίαση αποτελείται από μια συνδεδεμένη σειρά των βημάτων της παρουσίασης, ένα βήμα μπορεί να περιλαμβάνει ενέργειες δυναμικής κίνησης χωρίς όριο, το καθένα από τα οποία μπορεί να επηρεάσει διάφορα αντικείμενα συγχρόνως. Η παρουσίαση μπορεί να χτιστεί με το μαρκάρισμα των μεμονωμένων βημάτων, το επόμενο βήμα της παρουσίασης εμφανίζεται όταν ο εκπαιδευόμενος ενεργοποιήσει κάποιο ειδικό

γραφικό στοιχείο ή αυτόματα μετά από μια προαιρετική καθυστέρηση που ορίστηκε από τον παραγωγό της οπτικοποίησης.

Το ANIMAL στηρίζεται στο σύστημα *πλαισίου εργασίας* (*framework*) για την ανάπτυξη εφαρμογών οπτικοποίησης αλγορίθμων ANIMAL-FARM (***A**dvanced **N**avigation and **I**nteractive **M**od-elling of **A**nimations in **L**ectures – **F**ramework for **A**nimation **R**esources **M**anagement)*

Πλεονεκτήματα

Το Animal προσφέρει δυο τρόπους για να γίνει η οπτικοποίηση:

α) με τη χρήση μιας γλώσσας σεναρίων που ονομάζεται AnimalScript. Η γλώσσα αυτή υποστηρίζει και δομές δεδομένων, όπως οι πίνακες και οι λίστες.

Το AnimalScript (Rößling & Freisleben, 2001A; Rößling & Freisleben, 2001B), παρέχει τις βασικές γραφικές δυνατότητες και τις διαδικασίες, όπως *κίνηση* (*move*), *περιστροφή* (*rotate*), *εμφάνιση* (*show*), *απόκρυψη* (*hide*) και την αλλαγή των χρωμάτων των αντικειμένων. Η γλώσσα σεναρίων μπορεί να επεκταθεί, ώστε να περιλαμβάνει νέες λειτουργίες.

β) με τη χρήση της βιβλιοθήκης που παρέχει το εργαλείο. Πρόκειται για μια βιβλιοθήκη σε Java (Java API). Αφού υλοποιήσει κάποιος τον αλγόριθμο (σε Java), στη συνέχεια προσθέτει στον κώδικα του αλγορίθμου τις κατάλληλες κλήσεις συναρτήσεων της βιβλιοθήκης, που έχουν ως αποτέλεσμα την παραγωγή κώδικα στη γλώσσα AnimalScript, ο οποίος στη συνέχεια θα εκτελεστεί από το Animal. Πρόσφατα παρουσιάστηκε και ένα πρόσθετο (Animalipse plugin) της γλώσσας AnimalScript για το περιβάλλον Eclipse (Rossling Schroeder, 2009).

Επειδή οι οπτικοποιήσεις που παράγονται είναι γραμμένες σε Java μπορούν να εκτελεστούν σε οποιοδήποτε υπολογιστή και λειτουργικό σύστημα, δηλαδή είναι ανεξάρτητες πλατφόρμας.

Μειονεκτήματα

Το σημαντικό όμως μειονέκτημα είναι ότι δεν απευθύνεται τόσο σε μαθητές/φοιτητές όσο σε διδάσκοντες, αφού για να μπορεί να τη χρησιμοποιήσει κάποιος θα πρέπει ήδη να γνωρίζει προγραμματισμό σε Java, μαζί με τη γλώσσα AnimalScript. Υπάρχει φυσικά ο γραφικός συντάκτης, ο οποίος όμως δεν έχει την πληρότητα και τη δύναμη έκφρασης που έχει η γλώσσα σεναρίων οπτικοποίησης AnimalScript.

TRAKLA

Το Trakla ξεκίνησε στις αρχές της δεκαετίας του 90. Σκοπός του ήταν να εξετάσει την δυνατότητα της αυτόματης αξιολόγησης διαφόρων ασκήσεων των μαθητών στο μάθημα Δομές Δεδομένων και Αλγόριθμοι. Αρχικά οι ασκήσεις λύνονταν στο χέρι με χρήση χαρτιού και μολυβιού χωρίς να δίνουν την δυνατότητα γραφικής απεικόνισης και οπτικοποίησης των εκάστοτε αλγορίθμων. Έτσι στήθηκε μια πλατφόρμα μέσω της τεχνολογίας του email όπου οι απαντήσεις των ασκήσεων δίνονταν αυτόματα από το σύστημα. Η όλη ιδέα έχρηζε περεταίρω βελτίωσης, έτσι αρχικά δημιουργήθηκε ένα ξεχωριστό περιβάλλον διαχείρισης της εφαρμογής με την ονομασία Tred. Το γραφικό περιβάλλον πρόσθεσε την δυνατότητα της απεικόνισης και του χειρισμού διαφόρων δομών από τον εκάστοτε χρήστη – μαθητή, είχε την ικανότητα να αποκρύπτει λεπτομέρειες και να επιστρέφει στην αρχική μορφή της άσκησης μετά τη διαδικασία προσομοίωσης του κάθε αλγόριθμου. Επίσης ήταν δυνατή η οπτικοποίηση του ακριβούς τρόπου εκτέλεσης κάθε αλγορίθμου.

Η αρχική ιδέα του Trakla μετέπειτα ενσωματώθηκε σε ένα πιο περίπλοκο και διαδραστικό εκπαιδευτικό περιβάλλον. Το WWW-TRAKLA, ένα ολοκληρωμένο εκπαιδευτικό περιβάλλον το οποίο περιλάμβανε εργαλεία email, newsgroups, forms και των προϋπαρχόντων δυνατοτήτων του Trakla. Το σύστημα αποδείχθηκε ένα βασικό και επιτυχημένο εργαλείο στα χέρια των εκπαιδευτικών για πάνω από μία δεκαετία με εντυπωσιακά αποτελέσματα.

Κάποια από τα χαρακτηριστικά του Trakla ήταν:

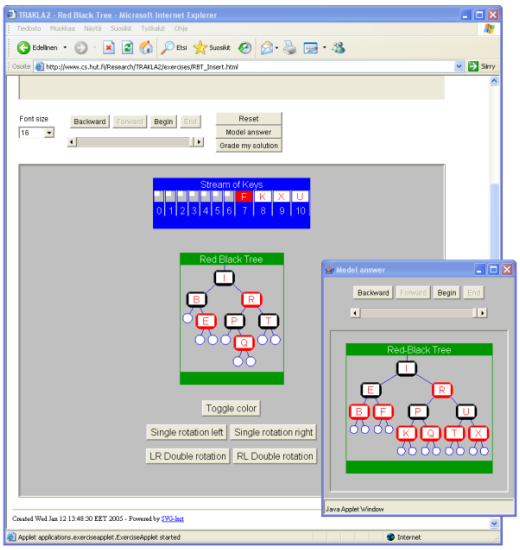
- Το σύστημα δημιουργούσε αυτόματα διαφορετικές ασκήσεις για κάθε μαθητή.
- Οι περισσότερες ασκήσεις συνοδεύονταν από γραφική απεικόνιση.
- Επέτρεπε στον μαθητή να πειραματιστεί με την γραφική απεικόνιση ώστε να βρεί την λύση της άσκησής του.

- Έδινε δυνατότητα online παράδοσης των ασκήσεων από τους μαθητές.
- Έστελνε στον μαθητή αυτόματα μία εκτίμηση για την σωστή λύση της κάθε άσκησης.
- Έδινε την δυνατότητα αναθεώρησης λανθασμένων απαντήσεων.
- Δημιουργούσε μια λύση πρότυπο για κάθε συγκεκριμένη άσκηση.
- Παρουσίαζε στατιστικά στοιχεία από τις επιδόσεις των μαθητών μεμονωμένα αλλά και σαν σύνολο.
- Ενθάρρυνε την δημιουργία συζητήσεων μεταξύ των μαθητών.

Μειονεκτήματα

Παρά το γεγονός ότι πολλά νέα χαρακτηριστικά έχουν εισαχθεί κατά την τελευταία δεκαετία στο σύστημα του Trakla, πολλές νέες καινοτομίες μένουν ακόμη ανεκμετάλλευτες και πολλά προβλήματα περιμένουν να λυθούν. Αρχικά η διαδικασία εισαγωγής και δημιουργίας νέων ασκήσεων χαρακτηρίζεται χρονοβόρα και απαιτεί ιδιαίτερες και απαιτητικές γνώσεις, διότι το γεγονός της δυνατότητας που έχει κάθε άσκηση να παράγει αυτόματα μία μοναδική πρότυπη λύση, αποτρέπει έναν μέσο εκπαιδευτικό να ολοκληρώσει αυτή την διαδικασία. Επίσης δημιουργούνται απορίες στο θέμα της αυτόματης υποβολής των ασκήσεων και κάτω από ποιες προϋποθέσεις μια άσκηση βαθμολογείται σωστή εφόσον έχει αυτόματα παράγει δεδομένα το ίδιο το σύστημα. Όπως και κατά πόσο επιτυγχάνεται η εκπαιδευτική διαδικασία αποτελεσματικά όταν οι πρότυπες λύσεις των ασκήσεων δεν παρουσιάζονται σε γραφική μορφή αλλά περιγραφικά.

Δυστυχώς ο σχεδιασμός και η αρχιτεκτονική του συστήματος δεν πληρούσε τις απαιτήσεις για την εισαγωγή νέων δυνατοτήτων. Έτσι βλέποντας το κενό που υπήρχε από παρόμοιες πλατφόρμες δημιουργήθηκε ένα νέο σύστημα που συνδύαζε τα πλεονεκτήματα του Trakla και του Tred σε ένα ενιαίο ολοκληρωμένο περιβάλλον το **Trakla2.**



Εικόνα 1.4β Παράδειγμα εφαρμογής του Trakla2

Γίνεται λοιπόν αντιληπτό ότι το Trakla προσφέρει πάρα πολύ χρήσιμες υπηρεσίες στην προσπάθεια κατανόησης και εκμάθησης των αλγορίθμων χρησιμοποιώντας έναν πρωτότυπο τρόπο οπτικοποίησης και μοντελοποίησής τους.

Jsamba

Το JSamba (http://www.cc.gatech.edu/gvu/softviz/algoanim/samba.html) είναι μία Java έκδοση του POLKA's front-end Samba. Είναι ένα σύστημα οπτικοποίησης αλγορίθμων γενικού σκοπού που τρέχει σαν ένα applet [1]. Το Jsamba είναι δομημένο από "πρωτόγονα" αντικείμενα (κείμενο, γραμμές, τρίγωνα, κύκλους και πολύγωνα). Στο σύστημα περιλαμβάνονται διαφορετικά εσωτερικά μεγέθη κειμένου. Μπορεί να σχεδιάσει δυναμικά αντικείμενα με μεγάλη ταχύτητα. Οι κινήσεις των δυναμικών αντικειμένων είναι λείες και γρήγορες. Δεν υπάρχουν εφέ που αναβοσβήνουν όταν η οπτικοποίηση 'τρέχει'. Τα δυναμικά αντικείμενα εμφανίζονται σε ένα παράθυρο applet, το οποίο είναι ξέχωρο από το control panel της οπτικοποίησης. Επιτρέπει στον χρήστη να δίνει άλλο μέγεθος στα δυναμικά αντικείμενα , σχεδιάζοντας γωνίες στο παράθυρο.

Υπάρχει ένας αριθμός από ευκολίες ελέγχου χρήσης. Οι προεπιλεγμένοι έλεγχοι περιλαμβάνουν ξεκίνημα, σταμάτημα, παύση και ταχύτητα της οπτικοποίησης που "τρέχει". Άλλοι περισσότερο συγκεκριμένοι έλεγχοι περιλαμβάνουν rewind και fast forward έλεγχο πάνω στην τρέχουσα οπτικοποίηση.

[1] Applet = εφαρμογή σε java που "τρέχει" στο διαδίκτυο.

Πλεονεκτήματα

Η JSamba είναι ένα γενικού σκοπού σύστημα οπτικοποίησης αλγορίθμου και γι' αυτό μπορεί να χρησιμοποιηθεί για προγράμματα γραμμένα σε κάθε γλώσσα. Χρησιμοποιεί εσωτερικές βιβλιοθήκες java για να δημιουργήσει οπτικοποίηση σε πολύ καλή ποιότητα. Η ταχύτητα σχεδίασης αντικειμένων είναι αρκετά μεγάλη και η κίνηση των δυναμικών αντικειμένων είναι επίσης γρήγορη. Αποκομίζει όλα τα πλεονεκτήματα από την java όπως μεταφερσιμότητα, ανεξαρτησία από πλατφόρμα, εύκολη πρόσβαση από έναν web browser και δεν απαιτούνται plug-ins. Τρέχει σαν ένα applet που ελαχιστοποιεί την ανάγκη για εγκατάσταση τοπικού λογισμικού.

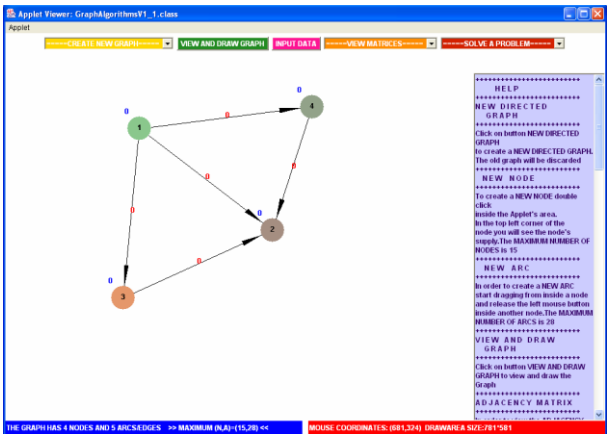
Μειονεκτήματα

Το κύριο μειονέκτημα της JSamba είναι ότι δεν έχει καθόλου εσωτερικά αντικείμενα δομών δεδομένων. Αυτό έχει ως αποτέλεσμα το σύστημα να είναι λιτό και απλό, αλλά από την άλλη μεριά, περιορίζει τις πιθανές περιοχές εφαρμογής. Αυτό το μειονέκτημα απαιτεί από τους χρήστες να γράφουν περισσότερο πολύπλοκες εντολές αρχείου script για να μοντελοποιήσουν αντικείμενα δομών δεδομένων.

Ειδικού σκοπού εργαλεία οπτικοποίησης αλγορίθμων

JAVENGA

Το JAVENGA (JAva-based Visualization Environment for Network and Graph Algorithms) είναι ένα **ε**κπαιδευτικό λογισμικό για τη διδασκαλία αλγορίθμων γραφημάτων. Το πιο σημαντικό χαρακτηριστικό του λογισμικού – το οποίο είναι γραμμένο στη γλώσσα προγραμματισμού Java – είναι ότι μπορεί να χρησιμοποιηθεί τόσο τοπικά όσο και από απόσταση με τη βοήθεια ενός φυλλομετρητή του διαδικτύου. Συνεπώς είναι κατάλληλο για την ανοιχτή και εξ' αποστάσεως εκπαίδευση. Η πρώτη του έκδοση δημοσιεύτηκε το 2009



Εικόνα 1.4γ Παράδειγμα εφαρμογής του JAVENGA

Το εργαλείο περιλαμβάνει έναν επεξεργαστή γράφων, όπου μπορεί να σχεδιαστεί οποιαδήποτε μορφή γράφου και επιτρέπει τους χρήστες να δώσουν αρχικές τιμές σε έναν αλγόριθμο ώστε μετέπειτα να δουν την οπτικοποιημένο εκτέλεση του σύμφωνα με τις δικές τους επιλογές βήμα βήμα ή σαν ολοκληρωμένη διαδικασία.

JELLRAP

Το JELLRAP (Java Enhanced LL/LR Animated Parser) είναι διαδραστικές ασκήσεις από LL(1), LL(2) και LR τεχνικές ανάλυσης. Παρέχει ασκήσεις για το χτίσιμο Πρώτου, Ακόλουθου και πίνακα ανάλυσης και για τις τρεις τεχνικές ανάλυσης. Και οι τρεις τεχνικές ανάλυσης καταλήγουν με μία δυναμική ανάλυση ενός τρόπου βήματος μιας εισαγόμενης από τον χρήστη συμβολοσειράς(string). Η οπτικοποίηση εμφανίζει ένα βήμα κάθε φορά την διαδικασία ανάλυσης του εισαγόμενου string και την παραγωγή ενός ανεστραμμένου ή μη δέντρου ανάλυσης.

Οι χρήστες μπορούν να εισάγουν οποιοδήποτε τύπο γραμματικής για να κάνουν την άσκηση. Κάθε τομέας άσκησης τρέχει σε ένα ξεχωριστό παράθυρο με ένα εισαγόμενο πλαίσιο κειμένου για να εισάγουν απαντήσεις οι χρήστες. Οι χρήστες μπορούν να εισάγουν τις απαντήσεις και να ζητούν το πακέτο για να ελέγχουν την ακρίβεια τους ή απλά να απαιτούν από το σύστημα να τους παρέχει τις σωστές απαντήσεις της άσκησης. Οι λάθος απαντήσεις είναι υπογραμμισμένες για να δείχνουν τα λάθη και οι χρήστες είναι ελεύθεροι να διορθώνουν τις απαντήσεις τους ή να ζητούν τις σωστές απαντήσεις από το σύστημα.

Το JELLRAP είναι γραμμένο σε Java και μπορεί να τρέξει σαν ένα applet ή σαν εφαρμογή.

Πλεονεκτήματα

Όπως κάθε προηγούμενο αξιολογημένο γενικού σκοπού εργαλείο οπτικοποίησης αλγορίθμου το JELLRAP είναι γραμμένο σε Java , και για αυτό έχει όλα τα πλεονεκτήματα που κληρονομούνται από την java. Είναι μία καλά σχεδιασμένη on-line άσκηση ανάλυσης που καλύπτει τα περισσότερα από τα θέματα ανάλυσης που μπορεί να εφαρμοστεί σε αυτή την εργασία. Οι χρήστες μπορούν να εισάγουν οποιαδήποτε γραμματική στην εργασία. Καλύπτει τις περισσότερες ασκήσεις που χρειάζεται να παράγει αυτή η εργασία και το εργαλείο μπορεί να υιοθετηθεί εύκολα για τους σκοπού αυτής της εργασίας.

Η διανομή του είναι ελεύθερη.

Μειονεκτήματα

Η δυναμική διαδικασία ανάλυσης είναι μόνο σε μορφή βήματος. Οι χρήστες πρέπει να κλικάρουν στο αντίστοιχο κουμπί της εφαρμογής για κάθε βήμα της ανάλυσης για να δουν το κάθε στάδιο της διαδικασίας. Υιοθετώντας αυτό το εργαλείο μπορεί να καταναλωθεί έξτρα χρόνος για την κατανόηση του κώδικα. Οι αλλαγές μπορεί να είναι προβληματικές γιατί μπορεί να απαιτήσουν εκτεταμένες αλλαγές στον υπάρχον κώδικα. Αυτό μπορεί να οδηγήσει σε συμβιβασμό με την τρέχουσα εφαρμογή.

1.5 Βασικές τεχνικές για τις επιδείξεις οπτικοποίησης αλγορίθμου

Οι Brown και Hershberger (1998) υπογραμμίζουν ότι οι σχεδιαστές οπτικοποίησης λογισμικού αντιμετωπίζουν σημαντικά προβλήματα προκειμένου να επιδείξουν αποδοτικά όλη την διαθέσιμη πληροφορία στην οθόνη. Υπάρχει ένας μεγάλος αριθμός πληροφοριών που επιδεικνύονται σε οθόνες μικρού σχετικά μεγέθους και χαμηλότερης ανάλυσης από το χαρτί. Αυτοί οι συντάκτες παρουσιάζουν έναν κατάλογο τεχνικών που οι σχεδιαστές AV χρησιμοποιούν για να αντιμετωπίσουν τέτοια προβλήματα συμπεριλαμβανομένης της χρήσης πολλαπλών όψεων, ενδείξεων καταστάσεων, επιλογών δεδομένων εισόδου, και τεχνικών χρώματος για την κωδικοποίηση των καταστάσεων των δομών δεδομένων και δίνοντας έμφαση στην δραστηριότητα.

Πολλαπλές όψεις: τα συστήματα AV πρέπει να κωδικοποιούν πολλές πληροφορίες (ειδικά όταν εξετάζουν σύνθετους αλγορίθμους ή πολλούς αλγορίθμους ταυτόχρονα) και είναι σχεδόν αδύνατο να παρουσιαστούν όλες οι διαθέσιμες (και επιθυμητές) πληροφορίες σε μια ενιαία όψη. Οι πολλαπλές όψεις έχουν υιοθετηθεί από τους σχεδιαστές για να διανείμουν αποτελεσματικά τις πληροφορίες και να παρουσιάζουν στο χρήστη συμπληρωματικές αναπαραστάσεις του αλγορίθμου. Κάθε όψη επιδεικνύει μόνο μερικές πτυχές του αλγορίθμου που βοηθά το χρήστη να καταλάβει την παρουσίαση.

Ενδείξεις κατάστασης: Οι animators μπορούν να παρουσιάσουν αλλαγές στην κατάσταση ενός συνόλου στοιχείων με την αλλαγή των στοιχείων των γραφικών αναπαραστάσεών τους στην οθόνη (π.χ. θέση, μορφή ή χρώμα). Σύμφωνα με Lowe (2003) οι αλλαγές στις δυναμικές επιδείξεις μπορούν να είναι αλλαγές "μορφής" (μετασχηματισμοί) που αναφέρονται στις αλλαγές των ιδιοτήτων των αντικειμένων (π.χ. μέγεθος, μορφή και χρώμα), αλλαγές "θέσης" (μεταφράσεις) με τα αντικείμενα να κινούνται από τη μια θέση προς την άλλη, και αλλαγές "συνυπολογισμού" (μεταβάσεις) όταν εμφανίζονται τα αντικείμενα ή εξαφανίζονται από την εικόνα.

Επιλογή δεδομένων εισόδου: Είναι επίσης σημαντικό να επιτραπεί στο χρήστη να επιλέξει το είδος και το ποσό των στοιχείων για να εξετάσει τη συμπεριφορά του αλγορίθμου. Προτείνεται να χρησιμοποιηθεί ένα μικρό ποσό στοιχείων όταν εισάγεται για πρώτη φορά ένας αλγόριθμος και μεγαλύτερα ποσά σε μεταγενέστερα στάδια της εργασίας όταν ο στόχος της διδασκαλίας είναι να αναπτυχθεί επιμελημένη και διαισθητική κατανόηση της συμπεριφοράς του αλγορίθμου (Brown & Hershberger, 1998). Η δυνατότητα για την επιλογή ειδικών μορφών δεδομένων για παιδαγωγικούς λόγους είναι επίσης σημαντική στα συστήματα AV. Οι χρήστες πρέπει να είναι σε θέση να επιλέξουν τα προκαθορισμένα σύνολα δεδομένων που είτε ωθούν τον αλγόριθμο σε ακραία συμπεριφορά είτε παρουσιάζουν ειδικές περιπτώσεις συμπεριφοράς προκειμένου να προωθηθεί η κατανόηση του σπουδαστή.

Τεχνικές χρώματος: Οι τεχνικές χρώματος χρησιμοποιούνται επίσης ευρέως στις απεικονίσεις AV (AV displays) για να μεταβιβάσουν αποτελεσματικά σημαντικές πληροφορίες κατάστασης. Το χρώμα μπορεί να χρησιμοποιηθεί για να κωδικοποιήσει την κατάσταση των δομών δεδομένων ή για να δώσει έμφαση στη δραστηριότητα. Παραδείγματος χάριν μια οπτικοποίηση μπορεί προσωρινά να χρωματίσει μια μικρή περιοχή με ένα αντιπαραβαλλόμενο χρώμα για να στρέψει την προσοχή στη χρωματισμένη

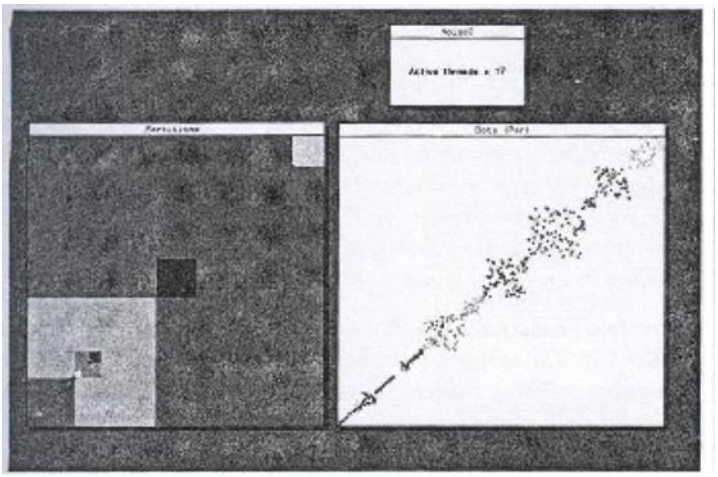
περιοχή και να δώσει έμφαση στα ακριβή στοιχεία στα οποία έχει επιπτώσεις η συγκεκριμένη λειτουργία του αλγορίθμου.

Ομαλή οπτικοποίηση: Τα συστήματα AV υιοθετούν συνήθως την "ομαλή" οπτικοποίηση για την παρουσίαση της μετάβασης των δεδομένων από τη μια θέση στην άλλη. Οι μεταβάσεις κατάστασης του συνόλου δεδομένων μπορούν να είναι διακριτές αλλά "η οπτικοποίηση μπορεί να είναι χρήσιμη για να βοηθήσει στην ομαλοποίηση της μετάβασης μεταξύ διακριτών καταστάσεων ενός σύνθετου αλγορίθμου ή μιας διαδικασίας" (Stasko, 1998, σελ. 104). Ένα χαρακτηριστικό παράδειγμα ομαλής οπτικοποίησης είναι όταν δύο ραβδιά αλλάζουν ομαλά θέση απεικονίζοντας την ιδιαίτερη αλλαγή της θέσης δύο τιμών στο σύνολο δεδομένων.

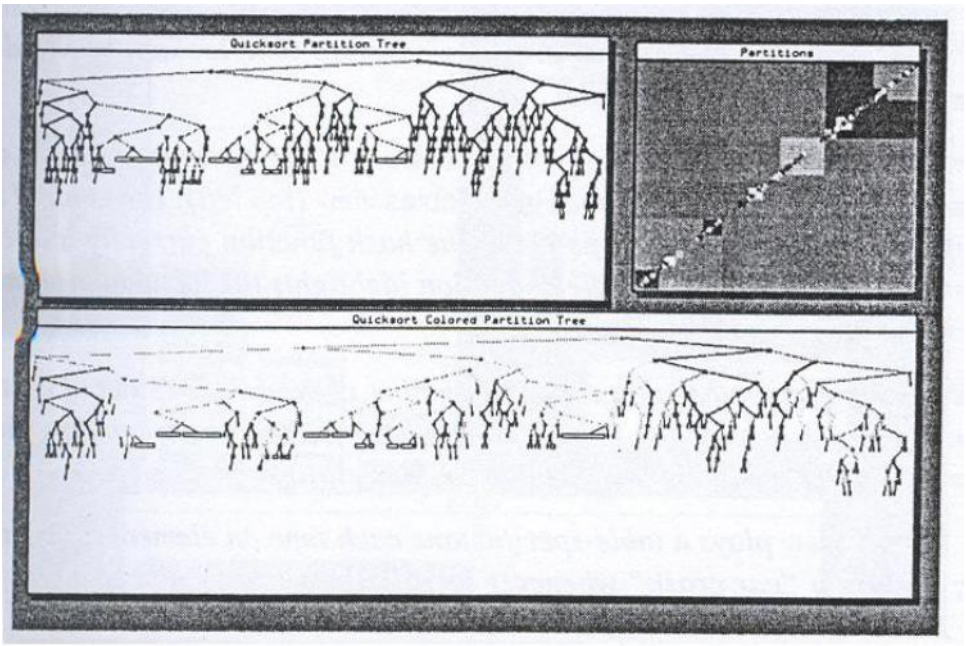
1.5.1 Παραδείγματα θεμελιωδών τεχνικών οπτικοποίησης αλγορίθμων

Παρακάτω παρατίθενται παραδείγματα κάποιων από τις παραπάνω θεμελιώδεις τεχνικές οπτικοποίησης αλγορίθμων.

Quicksort

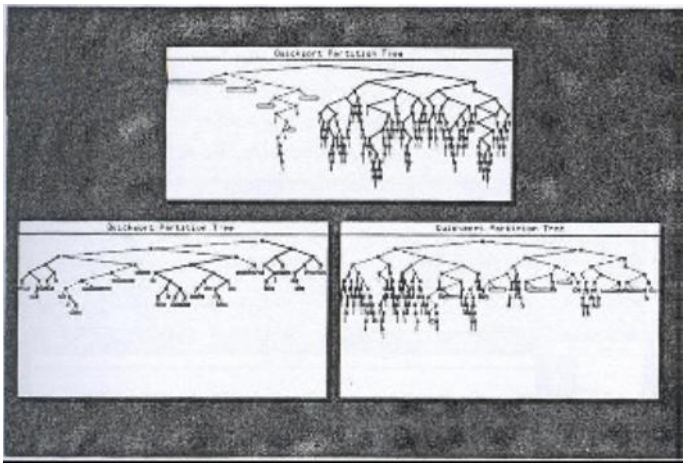


Η εικόνα 1.5α δείχνει τρεις όψεις μίας απλοποιημένης εκδοχής παράλληλης εφαρμογής του αλγορίθμου quicksort. Ο αλγόριθμος αρχικά χωρίζει τα στοιχεία που πρέπει να ταξινομηθούν σε δύο υποενότητες. Μετά το διαχωριστικό νήμα ταξινομεί κατ' επανάληψη το ένα υπό-αρχείο ενώ ένα νέο νήμα χρησιμοποιείται για να ταξινομήσει το άλλο.



Εικόνα 1.5β

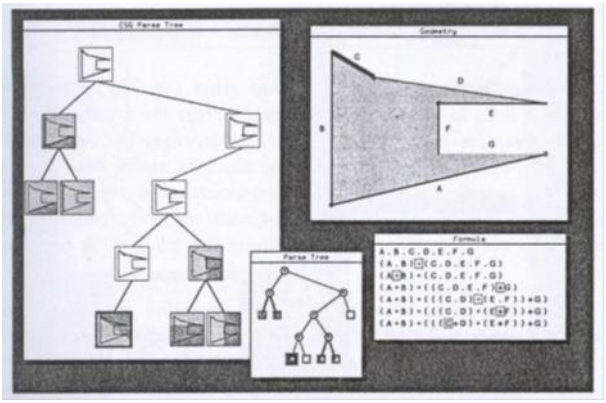
Η εικόνα 1.5β παρουσιάζει δύο δυαδικά δέντρα της διαδικασίας διαχωρισμού. Και στα δύο δέντρα κάθε κόμβος αναπαριστά μία θέση στον πίνακα που ταξινομείται και κάθε υπό-δέντρο αναπαριστά ένα υπό-αρχείο των στοιχείων που ταξινομούνται εγγενώς. Το υπό-αρχείο χωρίζεται σε δύο υπό-αρχεία που χωρίζονται από τη ρίζα του υπό-δέντρου.



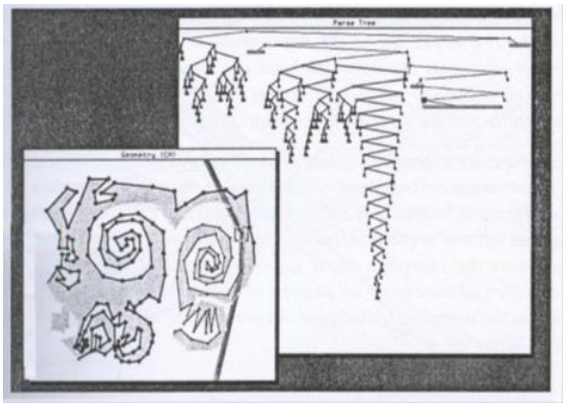
Εικόνα 1.5γ

Η εικόνα 1.5γ συγκρίνει τρεις διαφορετικές εφαρμογές του quicksort. Το πάνω δέντρο δείχνει έναν διαδοχικό αλγόριθμο. Το δέντρο στα δεξιά δείχνει την αφελή παράλληλη εφαρμογή και το αριστερό δείχνει έναν πιο περίπλοκο παράλληλο αλγόριθμο.

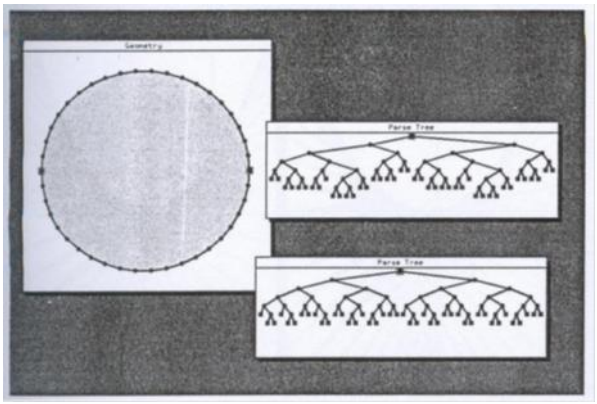
Boolean φόρμουλες για απλά πολύγωνα



Εικόνα 1.6α



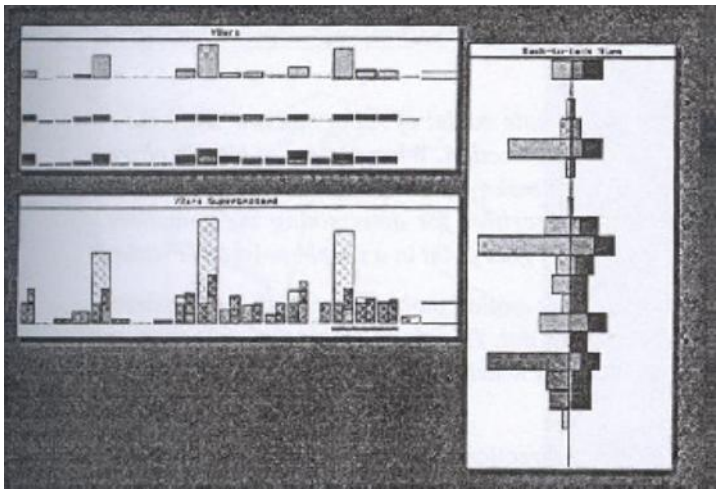
Εικόνα 1.6β



Εικόνα 1.6γ

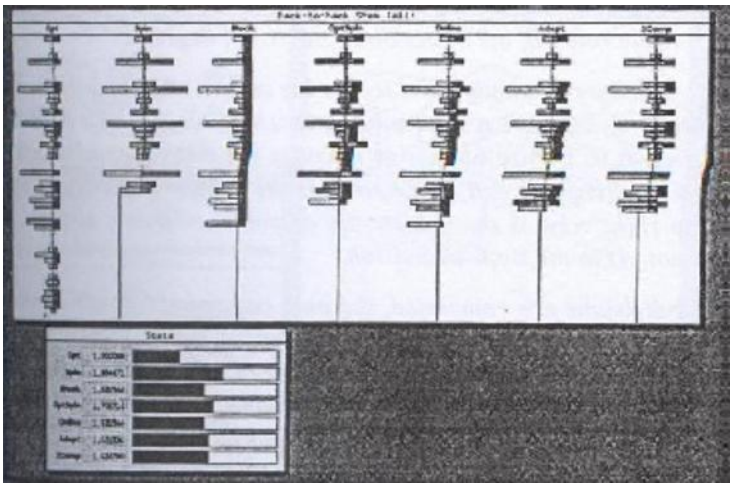
Οι εικόνες 1.6α – 1.6γ παρουσιάζουν έναν αλγόριθμο που βρίσκει την φόρμουλα ενός δοθέντος απλού πολυγώνου(Dobkin, 1988]. Ο αλγόριθμος αρχικά χωρίζει το πολύγωνο στις αριστερότερες και δεξιότερες κορυφές του για να παράγει δύο πολυγωνικές αλυσίδες, μια ανώτερη αλυσίδα και μια χαμηλότερη αλυσίδα. Μετά βρίσκει φόρμουλες για τις δύο αλυσίδες και τις κόβει. Για να βρεθεί η Boolean φόρμουλα για μία πολυγωνική αλυσίδα, ο αλγόριθμος χωρίζει την αλυσίδα σε μία κορυφή στην κυρτή περιφέρεια της αλυσίδας και συνδυάζει τους τύπους που βρίσκει κατ' επανάληψη για τις δύο υπό-αλυσίδες. Αν οι δύο αλυσίδες ενώνονται σε μία κορυφή σχηματίζοντας μία οξεία γωνία, οι φόρμουλες που αντιστοιχούν στις δύο αλυσίδες κόβονται. Αλλιώς οι φόρμουλες συγκλίνουν.

To spin or to block?



Εικόνα 1.7α

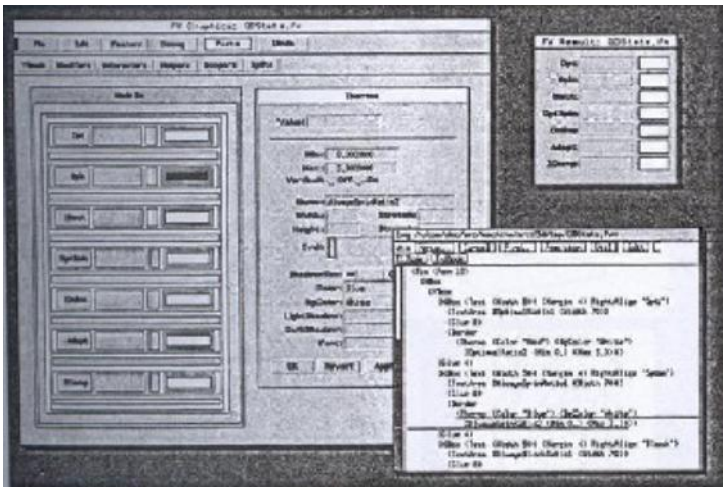
Η εικόνα 1.7α συγκρίνει ένα on-line αλγόριθμο που εξετάζει τους προηγούμενους τρεις χρόνους αναμονής ενάντια στο βέλτιστο αλγόριθμο. Τα τρία παράθυρα παρουσιάζουν διαφορετικές όψεις των ίδιων δεδομένων. Στην όψη Vbars, η πάνω σειρά παρουσιάζει τους χρόνους αναμονής, η μεσαία σειρά αναπαριστά τον βέλτιστο αλγόριθμο και η κάτω σειρά αναπαριστά τον on-line αλγόριθμο που παρουσιάζεται.



Εικόνα 1.7β

Η εικόνα 1.7β χρησιμοποιεί μια ιδιαίτερη τεχνική (back-to-back stems) για να συγκρίνει έξι διαφορετικούς on-line αλγορίθμους με τον βέλτιστο off-line αλγόριθμο. Η απόδοση της κάθε μεθόδου παρουσιάζεται γραφικά από την πρόοδο προς τα κάτω στη στήλη του. Η

όψη Stats στο κάτω μέρος της οθόνης δείχνει τους ρυθμούς απόδοσης με νούμερα και με γραφική αναπαράσταση.



Εικόνα 1.7γ

Η εικόνα 1.7γ δείχνει τον γραφικό συντάκτη στα αριστερά. Ένας μετρητής θερμομέτρων έχει επιλεχτεί, ο οποίος τοποθετεί επάνω μια μορφή ιδιότητας(στο δεξιό μέρος του συντάκτη) και υπογραμμίζει την κειμενική περιγραφή του μετρητή στον συντάκτη του κειμένου(κάτω δεξιά).

1.6 Ενδιαφέροντα γεγονότα

Οι περισσότερες επιδείξεις οπτικοποίησης προγράμματος μπορούν να δημιουργηθούν αυτόματα, ενώ οι περισσότερες οπτικοποιήσεις αλγορίθμου δεν μπορούν. Εδώ παρουσιάζονται τα λεγόμενα "ενδιαφέροντα γεγονότα", μια προσέγγιση που τα περισσότερα συστήματα οπτικοποίησης αλγορίθμου έχουν υιοθετήσει για το σχολιασμό των αλγορίθμων με πρόσθετες πληροφορίες που διαβιβάζονται στις απεικονίσεις για τη δημιουργία των επιδείξεων. Τα ενδιαφέροντα γεγονότα είναι σημαντικά και στους χρήστες που αλληλεπιδρούν με ένα σύστημα οπτικοποίησης αλγορίθμου και στους προγραμματιστές που αναπτύσσουν σχεδιοκινήσεις (ως τρόπος να απλοποιηθεί η διαδικασία εφαρμογής μιας οπτικοποίησης).

Η μέθοδος που υιοθετήθηκε από την BALSA και ακολουθήθηκε από τα περισσότερα συστήματα οπτικοποίησης αλγορίθμου γενικού σκοπού (π.χ., TANGO (Stasko, 1990b), ANIM (Bentley, 1991a), Zeus (Brown, 91), και η CAT (Brown, 1996)) είναι να σχολιαστούν οι αλγόριθμοι με τα ενδιαφέροντα γεγονότα παρά τον καταναγκασμό ενός

αλγορίθμου για γίνει ριζικά διαδικαστικός έτσι ώστε να ενσωματωθεί κάθε σημαντική λειτουργία. Αυτά τα γεγονότα διαβιβάζονται στη συνέχεια από το σύστημα οπτικοποίησης αλγορίθμου σε όλες τις απεικονίσεις. Κάθε απεικόνιση αποκρίνεται στο ενδιαφέρον γεγονός με το σχεδιασμό των κατάλληλων εικόνων. Στην περίπτωση προσανατολισμένων προς το αντικείμενο συστημάτων οπτικοποίησης αλγορίθμου όπως το Zeus, κάθε απεικόνιση είναι μια υποκατηγορία ενός παραθύρου με πρόσθετες μεθόδους για να χειριστεί κάθε ενδιαφέρον γεγονός.

Η προσέγγιση BALSA ελαχιστοποιεί τις αλλαγές στον αλγόριθμο, δεδομένου ότι ο αλγόριθμος αυξάνεται και δεν μετασχηματίζεται. Φυσικά εάν κάποιος είναι πρόθυμος να τροποποιήσει τον αλγόριθμο σε διαδικασία, τα γεγονότα μπορούν να προκύψουν ‘‘αυτόματα’’ από μάλλον έναν απλό προ-επεξεργαστή που παρεμβάλλει έναν σχολιασμό ως την πρώτη δήλωση κάθε διαδικασίας. Οι παράμετροι του γεγονότος θα είναι το όνομα της διαδικασίας που ακολουθείται από τα κατηγορήματα της διαδικασίας.

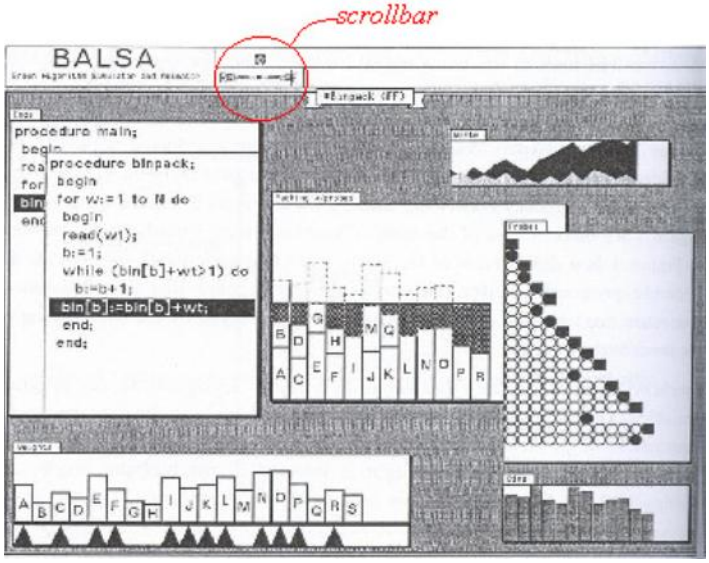
Μερικά συστήματα οπτικοποίησης αλγορίθμου, όπως το TANGO(Stasko, 1990b), παρέχουν έναν συντάκτη έτσι ώστε τα γεγονότα να μπορούν να προστεθούν με έναν δομημένο τρόπο χωρίς πραγματικά να αλλάξουν τον πηγαίο κώδικα. Ο συντάκτης τροποποιεί τον πηγαίο κώδικα για να περιλάβει τους σχολιασμούς προτού να οργανωθεί, χωρίς να παρουσιάζεται ποτέ ο τροποποιημένος πηγαίος κώδικας στον προγραμματιστή.

Τα ενδιαφέροντα γεγονότα βοηθούν επίσης να λυθούν προβλήματα όπως, η σε πραγματικό χρόνο απόδοση (τα στοιχεία που προσεγγίζονται ή τροποποιούνται μπορούν να είναι δαπανηρά για να προσδιοριστούν, με συνέπεια απαράδεκτη απόδοση) και την πληροφοριακή φύση των επιδείξεων. Κατ' αρχάς, τα στοιχεία που μπορούν να είναι δαπανηρά για να προσδιοριστούν αυτόματα συχνά μπορούν να προσδιορίζονται εύκολα από τον αλγόριθμο. Τα δεδομένα μπορούν να συνδεθούν με ένα γεγονός και ως εκ τούτου να βελτιώσουν την απόδοση σε πραγματικό χρόνο. Δεύτερον, ‘‘οι υπαινιγμοί’’ σχετικά με τα χαρακτηριστικά των αλγορίθμων (που είναι γνωστοί από τον αλγόριθμο αλλά όχι από τις επιδείξεις) μπορούν να συνδεθούν με τα γεγονότα.

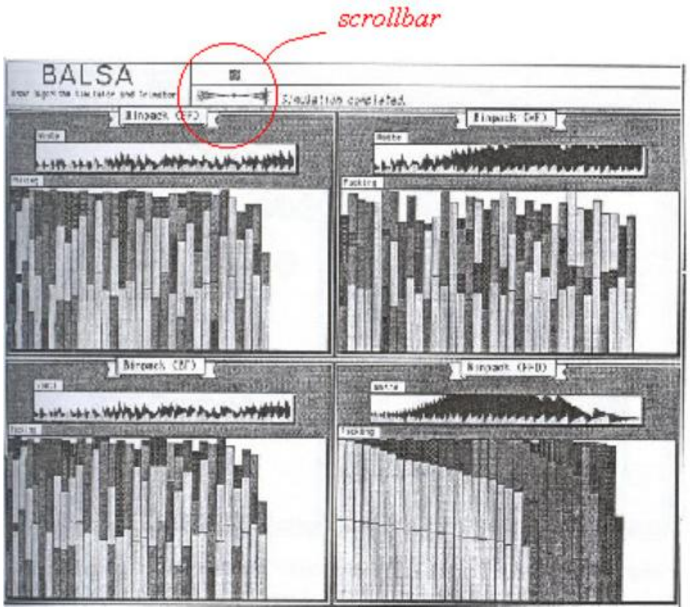
Υπάρχουν πολυάριθμες πρόσθετες ευχάριστες ιδιότητες αυτού του προτύπου για την οπτικοποίηση ενός αλγόριθμου. Μια όψη μπορεί να διορθωθεί και να εξασκηθεί ανεξάρτητα οποιουδήποτε αλγορίθμου με τη τροφοδότηση του από μία σειρά γεγονότων που παράγονται όταν ο χρήστης εισάγει τα δεδομένα (ή ακόμα και τυχαία). Ομοίως ένας αλγόριθμος μπορεί να διορθωθεί ανεξάρτητα οποιωνδήποτε όψεων. Επειδή οι όψεις

εφαρμόζονται εξωτερικά στους αλγορίθμους, είναι εύκολο να επαναχρησιμοποιηθούν οι όψεις μεταξύ των σχετικών αλγορίθμων. Ένας αλγόριθμος που σχεδιοκινείται δεν χρειάζεται να εφαρμοστεί σε οποιαδήποτε ιδιαίτερη γλώσσα εφ' όσον είναι προσφωνήσιμος από το σύστημα οπτικοποίησης του αλγορίθμου, ούτε και υπάρχουν οποιοιδήποτε περιορισμοί στις δομές δεδομένων που χρησιμοποιούνται στον αλγόριθμο.

Εάν η σειρά των γεγονότων που παράγονται από έναν αλγόριθμο σώζεται καθώς ο αλγόριθμος "τρέχει", τότε οι αλγόριθμοι μπορούν να τρέξουν "προς τα πίσω" τροφοδοτώντας τις όψεις των γεγονότων με αντίστροφη σειρά, με μια πρόσθετη "σημαία" που δείχνει ότι ο αλγόριθμος είναι σε αντίστροφη μορφή. Το BALSA είχε αυτό το χαρακτηριστικό γνώρισμα (βλ. εικόνες 1.8 και 1.19: το scrollbar ακριβώς δεξιά του λογότυπου BALSA επέτρεψε στο χρήστη για να θέσει το βήμα και την κατεύθυνση της οπτικοποίησης). Ένας άλλος λόγος να διατηρηθούν τα γεγονότα είναι να επιτραπεί στο χρήστη να ανοίξει μια νέα όψη ενώ ένας αλγόριθμος τρέχει. Το σύστημα φέρνει τη νέα όψη προωθώντας στην όψη όλα τα γεγονότα που έχουν συμβεί ήδη.



Εικόνα 1.8 αλγόριθμος first-fit binpacking



Εικόνα 1.9 τέσσερις binpacking αλγόριθμοι οι οποίοι λειτουργούν στο ίδιο σύνολο βαρών.

Τελικά σε μία διαδραστική "ρύθμιση", ένας χρήστης που ερευνά έναν αλγόριθμο μπορεί να διευκρινίσει τα γεγονότα για τον καθορισμό των σημείων παύσης, για τον καθορισμό της τμηματοποίησης ενός ενιαίου βήματος, για το χαρακτηρισμό πόσου χρόνου χρειάζεται για να εκτελεστεί κάθε γεγονός, και για το συγχρονισμό των πολλαπλών αλγορίθμων.

ΚΕΦΑΛΑΙΟ 2: Σύγχρονες προοπτικές στην παιδαγωγική αξία της οπτικοποίησης αλγορίθμου

Γενικά

Η εμφάνιση της τεχνολογίας υπολογιστών προσφέρει την ευκαιρία για πολύμορφες, δυναμικές και διαλογικές αναπαραστάσεις γνώσης που αναμένονται να ενισχύσουν σημαντικά την μάθηση. Αυτό το κεφάλαιο παρουσιάζει μια επισκόπηση της παιδαγωγικής αποτελεσματικότητας των συστημάτων οπτικοποίησης αλγορίθμου, τα οποία χρησιμοποιούν δυναμικές οπτικές αναπαραστάσεις για την υποστήριξη της διδασκαλίας στον τομέα των αλγορίθμων υπολογιστών. Ο συντάκτης δίνει έμφαση στις σημαντικότερες εννοιολογικές και μεθοδολογικές προόδους στον τομέα, αναλύοντας τις ιδιότητες των αναπαραστάσεων που επιδεικνύονται συνήθως από τέτοια συστήματα και παρουσιάζοντας σημαντικά ερευνητικά αποτελέσματα σχετικά με την παιδαγωγική αποδοτικότητά τους.

Οι διαθέσιμες μελέτες δείχνουν ότι δεν είναι η ποιότητα της γραφικής επίδειξης (“τι βλέπουν οι σπουδαστές”), αλλά η εμπλοκή των σπουδαστών στις ενεργές καταστάσεις εκμάθησης με τα συστήματα οπτικοποίησης αλγορίθμου (“τι κάνουν οι σπουδαστές”), η οποία έχει επιπτώσεις ουσιαστικά στην έκβαση της εκμάθησης. Επιπλέον, φαίνεται ότι ένα σημαντικό επίπεδο εκμάθησης επιτυγχάνεται όταν τα συστήματα οπτικοποίησης αλγορίθμου ενσωματώνονται στις εκπαιδευτικές τοποθετήσεις που ακολουθούν το παράδειγμα των εποικοδομιστών. Σε αυτήν την περίπτωση οι σπουδαστές καθοδηγούνται όχι απλά να δουν τις οπτικοποιήσεις των εμπειρογνωμόνων και να αλληλεπιδράσουν με αυτές αλλά και να κατασκευάσουν τις δικές τους και να τις παρουσιάσουν στους συνομήλικους, αρχίζοντας κατά συνέπεια καρποφόρες συνομιλίες οικοδόμησης της γνώσης. Κατ' αυτό τον τρόπο τα συστήματα οπτικοποίησης αλγορίθμου γίνονται καλύτερα αντιληπτά ως ενισχυτικά “εργαλεία” κατασκευής από ότι πιο απλά ως “μεταφορείς γνώσης”. Προς ενίσχυση αυτού του ρόλου του λογισμικού φαίνεται ότι τα συστήματα κατασκευής “χαμηλής-τεχνολογίας και πιστότητας” AV μπορούν να είναι επαρκή για την υποστήριξη της εμπλοκής των σπουδαστών στις ουσιαστικές δραστηριότητες εκμάθησης.

Λέξεις κλειδιά. Οπτικοποίηση αλγορίθμου, γραφική αναπαράσταση, πολλαπλές αναπαραστάσεις, εκμάθηση πολυμέσων.

2.1 Εισαγωγή

Η εκμάθηση, σε μια μεγάλη έκταση, προκύπτει από την αλληλεπίδραση μεταξύ των εξωτερικών και εσωτερικών αναπαραστάσεων της γνώσης. Μια "εξωτερική" αναπαράσταση είναι μια συμβολική δομή που αντιπροσωπεύει κάτι άλλο. Αυτό το "κάτι άλλο" μπορεί να είναι ένα μέρος του αντιληπτού κόσμου (π.χ. μια εικόνα ενός τοπίου είναι μια αντιπροσώπευση του τοπίου) ή των προτύπων που χρησιμοποιούμε για να περιγράψουμε και να καταλάβουμε τον κόσμο (ένας τύπος, μια περιγραφή, μια γραφική παράσταση είναι όλες αναπαραστάσεις των προτύπων που χρησιμοποιούμε για να θεωρητικολογήσουμε για τον πραγματικό κόσμο).

Οι εξωτερικές αναπαραστάσεις υιοθετούν κάποιο κατάλληλο αντιπροσωπευτικό κώδικα (μπορεί να είναι περιγραφικός κώδικας όπως η γλώσσα ή απεικονιστικός κώδικας όπως οι εικόνες και η γραφική παράσταση) και μια μορφή (ακουστική, οπτική, αφής) για να καταστήσουν ρητές τις ποσοτικές ή/και ποιοτικές αλληλεξαρτήσεις μεταξύ των κατηγορικών στοιχείων του κόσμου.

Παραδείγματος χάριν, ο τύπος "F=m.a" αντιπροσωπεύει συμβολικά τη σχέση μεταξύ των μεγεθών της μάζας m, της επιτάχυνσης a και της δύναμης F που εφαρμόζονται στους πραγματικούς οργανισμούς. Τα παραδείγματα των εξωτερικών αναπαραστάσεων περιλαμβάνουν τους ορισμούς που δηλώνονται στη σαφή ή τεχνική γλώσσα, σε πίνακες στοιχείων, σε μαθηματικές εκφράσεις, σε γραφική παράσταση, σε σχεδιοκινήσεις, ακουστικά συνθήματα. Οι εξωτερικές παρατηρήσεις μπορούν να θεωρηθούν είτε ως "οχήματα" για τη μεταβίβαση στον σπουδαστή των κατάλληλων εκπαιδευτικών μηνυμάτων είτε ως αντικείμενο για τη διευκόλυνση της αυτό-έκφρασης, της αντανάκλασης και της συνεργασίας των σπουδαστών.

Καθένας τρόπος, είναι στο κέντρο της εκπαιδευτικής διαδικασίας και οι εκβάσεις της μάθησης εξαρτώνται έντονα από το κατάλληλο σχέδιο και τη χρησιμοποίησή του. Οι εσωτερικές αναπαραστάσεις, αφ' ετέρου, είναι αφηρημένα γνωστικά κατασκευάσματα (παραδείγματος χάριν ένα σχήμα, ένα προτασιακό δίκτυο ή ένα διανοητικό πρότυπο) που οι σπουδαστές αναπτύσσουν ως προϊόν της διαδικασίας εκμάθησης και στις οποίες βασίζουν την απόδοσή τους στην επίλυση προβλημάτων. Οι σπουδαστές υποτίθεται ότι πρέπει να αναπτύξουν τέτοιες εσωτερικές δομές όταν αντιλαμβάνονται και επεξεργάζονται γνωστικά ερεθίσματα πληροφοριών από τις εξωτερικές αναπαραστάσεις. Η ποιότητα

αυτής της αλληλεπίδρασης έχει επιπτώσεις έντονα στην ποιότητα της γνώσης και της περαιτέρω απόδοσης των σπουδαστών στις καταστάσεις επίλυσης προβλήματος.

Για την κατασκευή των αναπαραστάσεων στο έντυπο υλικό, οι εκπαιδευτικοί χρησιμοποιούν παραδοσιακά δύο σημαντικούς αντιπροσωπευτικούς κώδικες: κείμενο και στατικές απεικονίσεις (εικόνες ή γραφικά). Εντούτοις, η τεχνολογία υπολογιστών προσφέρει την ευκαιρία για ανάπτυξη πολύμορφων, δυναμικών και διαλογικών εξωτερικών αναπαραστάσεων που αναμένονται να ενισχύσουν σημαντικά την μάθηση. Στη σφαίρα της ενισχυμένης τεχνολογίας μάθησης, οι εκπαιδευτικοί μπορούν να χρησιμοποιήσουν πολλαπλούς αντιπροσωπευτικούς κώδικες και μορφές (τέτοια συστήματα καλούνται χαρακτηριστικά "συστήματα πολυμέσων") προκειμένου να παραχθούν πολλαπλές αναπαραστάσεις για την εκμάθηση. Μια μεγάλη πρόκληση στον τομέα ήταν πάντα, φυσικά, πώς να συνδυάσει τις πολλαπλές αναπαραστάσεις με έναν τρόπο όπου η εκμάθηση συμβαίνει με τον αποτελεσματικότερο και τον αποδοτικότερο τρόπο.

Οι δυναμικές αναπαραστάσεις παράγονται όταν αλλάζουν με κάποιο τρόπο τα εξωτερικά ερεθίσματα στο χρόνο. Η δυναμική γραφική παράσταση, η αφήγηση και το βίντεο είναι όλοι τύποι δυναμικών αναπαραστάσεων. Οι δυναμικές γραφικές αναπαραστάσεις (σχεδιοκινήσεις) χρησιμοποιούν το είδος του οπτικού κώδικα που αλλάζει στην οθόνη (με έναν συνεχή ή διακριτό τρόπο) τις ιδιότητες των γραφικών στοιχείων (όπως η θέση, το χρώμα, το μέγεθός τους). Είναι κοινή λογική ότι η οπτικοποίηση μπορεί να χρησιμοποιηθεί κατάλληλα στην εκπαίδευση για την αντιπροσώπευση των στατικών αλλαγών κάποιου φυσικού, τεχνητού ή φανταστικού συστήματος που εξελίσσεται στο χρόνο. Η κεντρική ιδέα για τη χρησιμοποίηση της οπτικοποίησης είναι η προσδοκία ότι το χαρακτηριστικό γνώρισμα του συστήματος που αλλάζει στο χρόνο θα μπορούσε να αντιπροσωπευθεί με συνέπεια από κάποια αντίστοιχη αλλαγή της δυναμικής γραφικής παράστασης και αυτή η λειτουργία χαρτογράφησης θα επέτρεπε στους σπουδαστής να αναπτύξουν αποτελεσματικά ένα κατάλληλο δυναμικό διανοητικό πρότυπο του συστήματος.

Η κατανόηση, επομένως, του πώς η οπτικοποίηση μπορεί να χρησιμοποιηθεί καλύτερα για λόγους εκμάθησης είναι σημαντική στα πλαίσια της ενίσχυσης της τεχνολογίας στην εκμάθηση ακριβώς επειδή η οπτικοποίηση προσφέρει στους εκπαιδευτικούς την πλήρως

διαφορετική δυνατότητα της παρουσίασης δυναμικής πληροφορίας στους σπουδαστής. Αυτό δεν μπορεί να επιτευχθεί στο τυπωμένο μέσο.

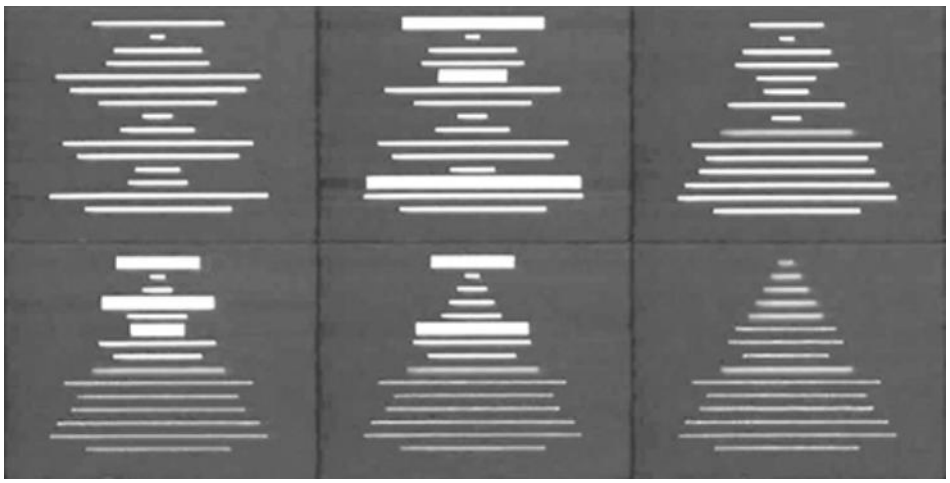
Μια σημαντική εφαρμογή των δυναμικών οπτικών αναπαραστάσεων γίνεται στη διδασκαλία εννοιών και μεθόδων στην περιοχή των αλγορίθμων υπολογιστών. Η εκμάθηση των αλγορίθμων θεωρείται δύσκολος στόχος (Stasko & Lawrence, 1998) όχι μόνο επειδή οι αλγοριθμικές διαδικασίες είναι γενικά σύνθετες, αλλά και επειδή είναι αρκετά αφηρημένες. Ένας αλγόριθμος είναι, κατ' αρχάς , μια σειρά καλά καθορισμένων βημάτων που εφαρμόζονται σε δομές δεδομένων και, υπό αυτήν τη μορφή, υπάρχει μόνο ως ένα "πλάσμα" στον αφηρημένο κόσμο των χρονικών μετασχηματισμών δεδομένων. Η χρήση των κατάλληλων οπτικών δυναμικών αναπαραστάσεων για την απεικόνιση της συμπεριφοράς του αλγορίθμου είναι, ενδεχομένως, ένα βήμα προς το να καταστήσει το αφηρημένο πιο συγκεκριμένο και το σύνθετο πιο κατανοητό.

Η εστίαση σε αυτό το κεφάλαιο είναι στο τρέχον επίπεδο γνώσης μας σχετικά με την παιδαγωγική αποδοτικότητα του λογισμικού οπτικοποίησης αλγορίθμου. Για να προσφέρουμε μία όσο το δυνατόν πληρέστερη επισκόπηση του θέματος έχουμε αναθεωρήσει πολλές από τις διαθέσιμες σήμερα εμπειρικές ερευνητικές μελέτες και μερικές από τις πιο διαφωτιστικές μετά-μελέτες. Ακολούθως πρόκειται να παρουσιάσουμε (α) τα βασικά χαρακτηριστικά γνωρίσματα των συστημάτων οπτικοποίησης αλγορίθμου με το σχολιασμό των ιδιοτήτων των δυναμικών αναπαραστάσεων που τα συστήματα αυτά συνήθως υιοθετούν, και (β) μια συνοπτική επισκόπηση των διαθέσιμων σήμερα ερευνητικών στοιχείων, συζητώντας επίσης τις αναδυόμενες επιπτώσεις για την εκπαιδευτική αποτελεσματικότητα αυτών των συστημάτων.

2.2 Οπτικοποίηση Αλγορίθμου

Η οπτικοποίηση λογισμικού περιλαμβάνει δύο βασικές υπο-περιοχές: οπτικοποίηση αλγορίθμου και προγράμματος. Η οπτικοποίηση προγράμματος (Program Visualization) αναφέρεται στις οπτικοποιήσεις του κώδικα και των στοιχείων του προγράμματος ενώ η οπτικοποίηση αλγορίθμου (Algorithm Visualization) περιλαμβάνει τις οπτικοποιήσεις που εξετάζουν τους αλγορίθμους (εικόνα 1.1). Στον τομέα της πληροφορικής, τα συστήματα AV κάνουν διαθέσιμες στους σπουδαστές πολλαπλές, δυναμικές και διαλογικές οπτικοποιήσεις των αλλαγών στις οποίες υποβάλλονται τα σύνολα δεδομένων όταν κάποιος αλγόριθμος εφαρμόζεται σε αυτά.

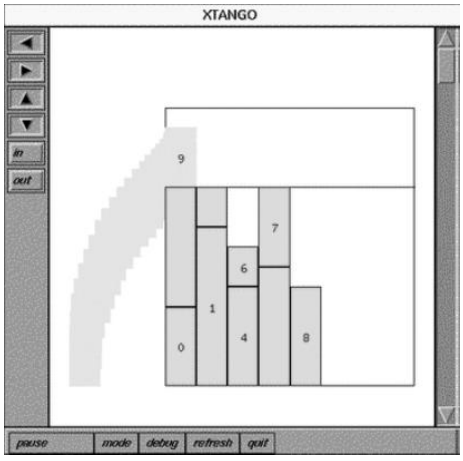
Σαν κοινός πρόγονος όλων των περιβαλλόντων AV αναφέρεται συνήθως μια διάρκειας τριάντα λεπτών ταινία χρώματος που παράχθηκε από τον Baecker (βλ. Baecker, 1981, 1998), που τιτλοφορείται "ταξινομώντας την ταξινόμηση". Η ταινία παρουσιάζει τη λειτουργία εννέα διαφορετικών εσωτερικών αλγορίθμων ταξινόμησης χρησιμοποιώντας την οπτικοποίηση του συνόλου δεδομένων που συνδέεται με επεξηγηματικό αφήγημα. Προσφέρει τη δυνατότητα να βιωθεί οπτικά η δυναμική του αλγορίθμου με τρόπους που είναι δύσκολο να περιγραφθούν απλά με τη χρησιμοποίηση κειμενικών αναπαραστάσεων. Ο Baecker (1998) υπογραμμίζει ότι "μπορούμε να δούμε τα προγράμματα κατά την διαδικασία, "τρέξιμο", και επομένως βλέπουμε τους αλγορίθμους με νέους και απροσδόκητους τρόπους... Αυτές οι όψεις παράγουν νέους όρους που είναι δύσκολο να εκφραστούν με λέξεις"(βλ. εικόνα 2.1).



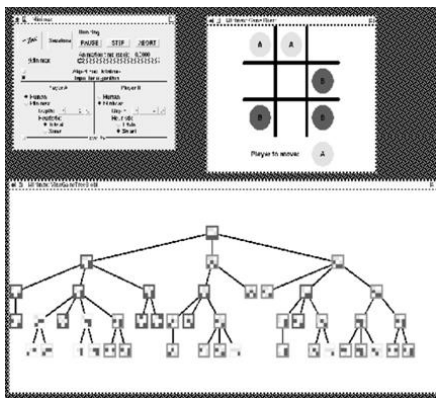
Εικόνα 2.1 Στιγμιότυπο οθόνης από το "ταξινομώντας την ταξινόμηση" (Baecker, 1981, 1998). Τα δεδομένα αντιπροσωπεύονται ως οριζόντιες ράβδους που αλλάζουν σταδιακά τη σχετική θέση τους καθώς ο αλγόριθμος ταξινόμησης εφαρμόζεται σε αυτές.

Αυτή η πρόωρη μη-διαλογική οπτικοποίηση ήταν μια χρήσιμη εμπειρία εκμάθησης; Ο Baecker (αυτόθι, σελ. 378) καταλήγει στο συμπέρασμα ότι οι "σημαντικές ιδέες στη συμπεριφορά του αλγορίθμου μπορούν να αποκτηθούν βλέποντας μόνο τα δεδομένα, εάν οι απεικονίσεις και ο συγχρονισμός σχεδιάζονται προσεκτικά, και συνοδεύονται από την κατάλληλη αφήγηση". Η λέξη κλειδί εδώ είναι "βλέποντας μόνο". 1εδομένου ότι θα συζητήσουμε αργότερα σε αυτό το κεφάλαιο, διάφορες μελέτες δείχνουν ότι βλέποντας απλά τις οπτικοποιήσεις του αλγορίθμου δεν φαίνεται να ωφελεί τους σπουδαστές ουσιαστικά. Εντούτοις, υπάρχει επίσης ισχυρή υποστήριξη (π.χ. Clark & Mayer, 2003) της άποψης ότι η ποιότητα της εκμάθησης βελτιώνεται σημαντικά με το να παρουσιάζονται

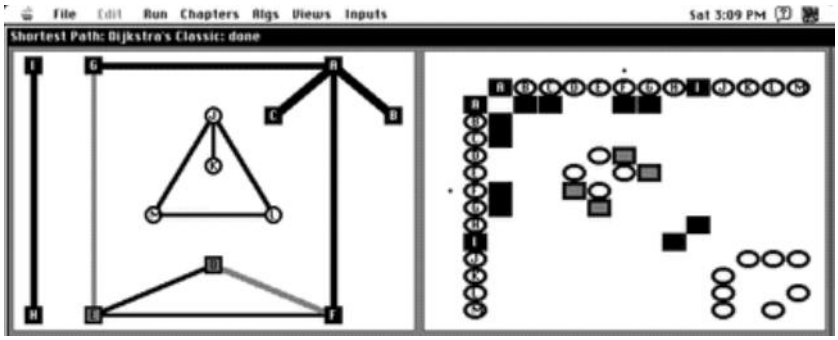
κατάλληλα στους σπουδαστές κειμενικές και οπτικές πληροφορίες. Σχολιάζουμε αυτήν την προφανή αντίφαση στο τελευταίο τμήμα του κεφαλαίου. Διάφορα βασισμένα σε υπολογιστή συστήματα SV έχουν αναπτυχθεί από τότε (π.χ. η BALSA (Brown, 1988a), Zeus (Brown, 1991), Tango (Stasko, 1990), βλ. εικόνα 2.2) και πολλά από αυτά έχουν υποβληθεί σε σημαντική εμπειρική αξιολόγηση, προσφέροντας κατά συνέπεια σημαντικές ιδέες στην παιδαγωγική αποδοτικότητά τους. Δύο βασικές εκτιμήσεις σχεδίασης στην ανάπτυξη αυτών των συστημάτων (και οποιουδήποτε άλλου συστήματος AV) είναι πάντα (α) πώς να προσαρμόσουν ελαστικά το επίπεδο αλληλεπίδρασης χρήστη-συστήματος για να ικανοποιηθούν οι απαιτήσεις των διάφορων εκπαιδευτικών δραστηριοτήτων, και (β) πώς να χτίσουν τις κατάλληλες οπτικοποιήσεις για να ενθαρρύνουν την ανάπτυξη αποδοτικών εσωτερικών αναπαραστάσεων του σπουδαστή.



Εικόνα 2.2 Οπτικοποίηση XTANGO first-fit
Binpacking (http://www.cc.gatech.edu/gvu/softviz/algoanim/xtango.html)



Εικόνα 2.3 Οπτικοποίηση του αλγορίθμου Zeus minmax
(http://www.research.compaq.com/SRC/zeus/home.html)



Εικόνα 2.4 Οπτικοποίηση αλγορίθμου Dijkstra's shortest path στο BALSA (http://www.eg.bucknell.edu/~zaccone/MacBALSA/MacBALSA.html)

Εικόνες 2.2-2.4 Στιγμιότυπα από συστήματα AV. Ο αναγνώστης μπορεί να δει τις διάφορες μορφές γραφικών τα οποία χρησιμοποιούνται για τη διαμόρφωση των διαδικασιών του αλγορίθμου.

Το πρώτο σημαίνει ότι τα συστήματα AV πρέπει να επιτρέπουν στους σπουδαστές να πειραματιστούν με την προσαρμογή των διάφορων λειτουργιών των συστημάτων ανάλογα με την κατάσταση. Εξετάστε, παραδείγματος χάριν, την περίπτωση συλλογής αρχικών δεδομένων. Το σύστημα πρέπει να προσφέρει τουλάχιστον τρεις ευδιάκριτες δυνατότητες: τα τυχαία στοιχεία, τα προκαθορισμένα ειδικά στοιχεία της περίπτωσης (π.χ. στοιχεία ακραίας συμπεριφοράς αλγορίθμου) και δεδομένα καθορισμένα από τον χρήστη. Το τελευταίο αναφέρεται στο πρόβλημα του πώς να αντιπροσωπεύσει τις δομές δεδομένων και τις διαδικασίες αλγορίθμου, για να επιτρέψει να προκύψουν αποδοτικοί όροι εκμάθησης. Παραδείγματος χάριν, υπάρχει οποιαδήποτε διαφορά εάν τα στοιχεία αντιπροσωπεύονται από κατακόρυφες αντί των οριζόντιων ράβδων (όπως στην εικόνα 2.1); Ή, πώς θα έπρεπε οι διάφορες όψεις ενός σύνθετου αλγορίθμου να διασυνδεθούν για να αποφευχθούν προβλήματα χρησιμοποίησης και να επιτραπεί στους σπουδαστές η μετάφραση μεταξύ τους;

Ποιοι είναι οι λόγοι για την ανάπτυξη συστημάτων οπτικοποίησης αλγορίθμου;

Κατ' αρχάς, επειδή υπάρχουν σημαντικοί εκφραστικοί περιορισμοί στις αναπαραστάσεις που χρησιμοποιούνται παραδοσιακά για την παρουσίαση των διαδικασιών του αλγορίθμου. Οι σπουδαστές αντιμετωπίζουν γενικά σοβαρές δυσκολίες κατά την προσπάθεια να κατανοήσουν τους συνήθως πολυσύνθετους μετασχηματισμούς των συνόλων δεδομένων που υποστηρίζονται μόνο από τις επίσημες κειμενικές αναπαραστάσεις (κώδικας προγράμματος) και άλλες στατικές οπτικοποιήσεις

(φωτογραφικές διαφάνειες, εικόνες και φιγούρες σχηματισμένες με το χέρι) (Naps et al., 2003). Οι στατικές απεικονίσεις δεν αντιπροσωπεύουν τα δυναμικά χαρακτηριστικά των αλγορίθμων ενώ η χρήση των δυναμικών αναπαραστάσεων αναμένεται να υποστηρίξει τους σπουδαστές στην βαθύτερη κατανόηση και της εννοιολογικής και της διαδικαστικής γνώσης της περιοχής.

Δεύτερον, επειδή οι άνθρωποι παρουσιάζουν μία ποικιλία από διαφορετικά μαθησιακά και γνωστικά στυλ, που τους κάνουν να προτιμούν ένα συγκεκριμένο αντιπροσωπευτικό κώδικα από έναν άλλο (π.χ. Wu & Martin,, 1997). Οι σπουδαστές φαίνονται βεβαίως να απολαμβάνουν τις σχεδιοκινήσεις και αυτό είναι ήδη ένα σημαντικό κίνητρο για τη συμμετοχή στις σημαντικές δραστηριότητες εκμάθησης που δεν πρέπει να υποτιμηθεί (Stasko & Lawrence, 1998).

Τρίτον, επειδή οι εκπαιδευτικοί διαισθητικά θεωρούν ότι η χρησιμοποίηση των συστημάτων AV παράγει καλύτερες εκβάσεις εκμάθησης (Naps et al., 2003). Αν και η πεποίθηση με την πραγματικότητα δεν συμπίπτουν απαραιτήτως, ένας εκπαιδευτικός που τίθεται θετικά προς τη χρήση των συστημάτων AV είναι αυτός που πιθανότερα θα εισαγάγει ένα τέτοιο σύστημα στην εκπαίδευση και θα προσπαθήσει να επιτύχει ό,τι καλύτερό από αυτό. Εντούτοις, πρέπει να παρατηρηθεί ότι οι δαπάνες χειρισμού των συστημάτων AV (για την ανάπτυξη, την υποστήριξη και τη διατήρηση τους) είναι μάλλον υψηλές. "Η φωτεινή υπόσχεση αυτών των τεχνικών εξασθενίζεται από το κόστος του σχεδίου, της κατασκευής, της ολοκλήρωσης και της συντήρησής τους", (Bazik, Tamassia, Reiss & Van Dam, 1998, σελ. 383).

2.2.1 Ιδιότητες των αναπαραστάσεων στις επιδείξεις απεικόνισης αλγορίθμου

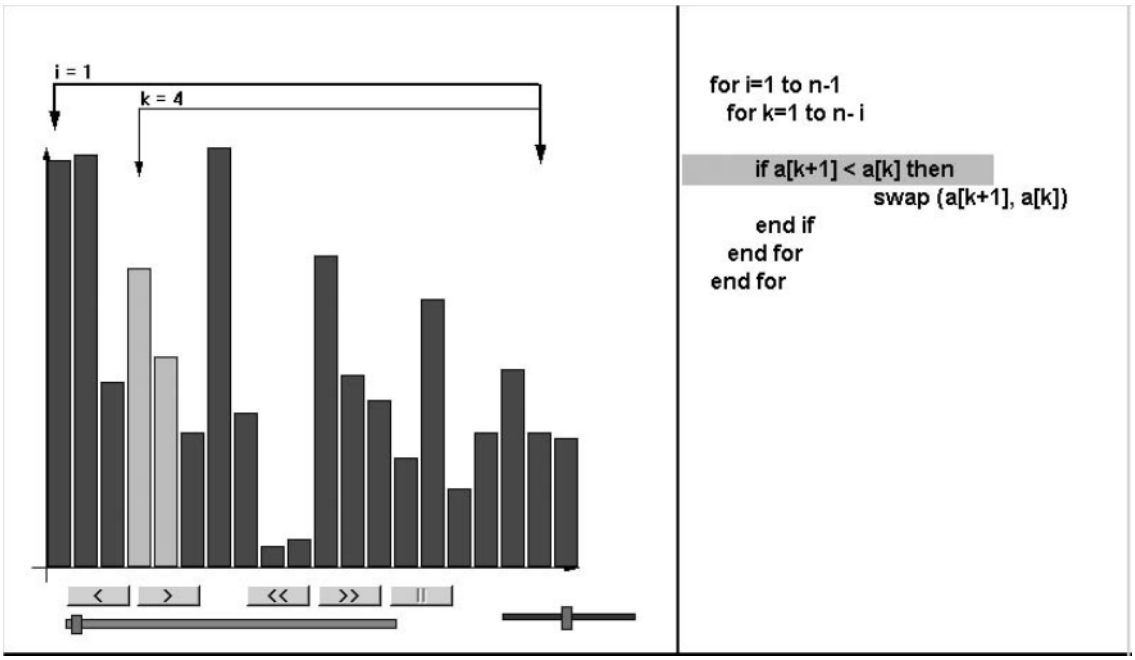
Ο σχεδιασμός των απεικονίσεων αλγορίθμου για διδασκαλία δεν είναι μια τετριμμένη προσπάθεια. Είναι σημαντικό ότι τα συστήματα AV δίνουν έμφαση μόνο στις ουσιαστικές πτυχές ενός αλγορίθμου που καταστέλλει οποιεσδήποτε ξένες λεπτομέρειες και που παρέχει σαφή και τακτοποιημένα γραφικά σχέδια (Baecker, 1998). Οι σχεδιαστές, επιπλέον, με τη χρησιμοποίηση της εμπειρίας των εκπαιδευτικών, πρέπει να αποφασίσουν πώς να αναπτύξουν και να συνδέσουν αποτελεσματικά τις οπτικοποιήσεις με τις κειμενικές αναπαραστάσεις (κώδικας προγράμματος) προκειμένου να καταστούν κατανοητά ακόμη και τα πιο περίπλοκα χαρακτηριστικά γνωρίσματα του αλγορίθμου. Για

να ολοκληρώσουν αυτούς τους στόχους, οι σχεδιαστές προσφεύγουν στις διάφορες τεχνικές επίδειξης και αναπτύσσουν τις αναπαραστάσεις με τις συγκεκριμένες ιδιότητες και χαρακτηριστικά γνωρίσματα. Αυτό το τμήμα εισάγει τον αναγνώστη σε αυτές τις αντιπροσωπευτικές τεχνικές σχολιάζοντας αναλυτικά τις ιδιότητες των αναπαραστάσεων που πρόκειται να βρεθούν χαρακτηριστικά σε ένα σύστημα AV.

Ο αντιπροσωπευόμενος και αντιπροσωπευτικός κόσμος

Ο "αντιπροσωπευόμενος κόσμος" σε ένα σύστημα AV είναι η περιοχή των συνόλων δεδομένων και των διαδικασιών του αλγορίθμου. Αυτό που αντιπροσωπεύεται πραγματικά σε ένα τέτοιο σύστημα είναι ο αφηρημένος κόσμος των λογικών διαδικασιών που αποτελούν την ουσία των αλγοριθμικών διαδικασιών. Αυτή η περιοχή αντιπροσωπεύεται παραδοσιακά από επίσημες κειμενικές αναπαραστάσεις (ψευδοκώδικας ή κώδικας που γράφεται σε κάποια γλώσσα προγραμματισμού) που περιγράφουν τα βήματα του αλγορίθμου. Οι οπτικοποιήσεις αλγορίθμου, εντούτοις, επιδιώκουν να εκμεταλλευτούν τη δύναμη των οπτικών αναπαραστάσεων με την αποκατάσταση μιας διαισθητικής αλληλογραφίας μεταξύ των δομών δεδομένων και των επιδειχθέντων γραφικών στοιχείων. Οι Petre, Blackwell και Green (1998) αποκαλούν αυτές οι οπτικοποιήσεις "χειροποίητα αντικείμενα πληροφοριών".

Στις επιδείξεις AV υπάρχουν συνήθως δύο βασικοί αντιπροσωπευτικοί κώδικες: κείμενο και γραφική παράσταση. Εκτός από την παρουσίαση του κώδικα του αλγορίθμου, το κείμενο χρησιμοποιείται επίσης για το σχολιασμό της γραφικής αναπαράστασης, την παρουσίαση των αριθμητικών στοιχείων, και την παροχή εξηγήσεων. Η γραφική αναπαράσταση αντιπροσωπεύει τα στοιχεία με τρόπους που θα επέτρεπαν στους σπουδαστές να κατανοήσουν διαισθητικά τις βασικές λειτουργίες του αλγορίθμου. Παραδείγματος χάριν, το ύψος των ράβδων (βλ. εικόνα 2.5) είναι ανάλογο προς την αξία των στοιχείων και έτσι αυτό βοηθά το θεατή να καταλάβει ευκολότερα πώς οι αλγόριθμοι ταξινόμησης κάνουν τις συγκρίσεις μεταξύ των στοιχείων και τις ταξινομούν αναλόγως.



Εικόνα 2.5 Στιγμιότυπο ενός πρώιμου πρότυπου AV . Δύο αναπαραστάσεις (ράβδοι κώδικα και δεδομένων) χρησιμοποιούνται για να παρουσιάσουν έναν απλό αλγόριθμο ταξινόμησης (bubble sort). Το ύψος των ράβδων είναι ανάλογο προς τις τιμές των δεδομένων. Τα κουμπιά κάτω από τα στοιχεία επιτρέπουν στο σπουδαστή να προωθήσουν ή γυρίσουν πίσω την οπτικοποίηση είτε βήμα προς βήμα είτε συνεχώς. Οι ολισθαίνοντες ρυθμιστές επιτρέπουν την προσαρμογή της ταχύτητας και την εύκολη μετακίνηση σε σημεία ενδιαφέροντος. Οι γραμμές και τα βέλη επάνω από τα στοιχεία και τις ελαφρότερα γκρίζες περιοχές (τονισμένες ράβδοι στοιχείων και γραμμή κώδικα στα δεξιά) υποστηρίζουν τους σπουδαστές στην μετάφραση μεταξύ δύο απεικονίσεων.

Οι πτυχές του αντιπροσωπευόμενου κόσμου που αντιπροσωπεύεται είναι εκείνες που θα βοηθούσαν τους σπουδαστές να καταλάβουν καλύτερα πώς ο απεικονισμένος αλγόριθμος έχει επιπτώσεις στο σύνολο των δεδομένων. Αυτές οι πτυχές μπορούν να ποικίλουν ανάλογα με το είδος οπτικοποίησης του αλγορίθμου. Παραδείγματος χάριν, κατά την παρουσίαση των αλγορίθμων ταξινόμησης είναι σημαντικό να αντιπροσωπευθούν οι τιμές των δεδομένων με τέτοιο τρόπο όπου οι διαδικασίες ταξινόμησης μπορούν εύκολα να απεικονιστούν. Οι πτυχές του αντιπροσωπευόμενου κόσμου που κάνουν τη μοντελοποίηση είναι οι γραφικές ιδιότητες όπως η μορφή του γραφικού στοιχείου (π.χ. κύκλος ή τετράγωνο), το μέγεθος, το χρώμα και η θέση του.

Οι εικόνες των αντικειμένων πραγματικού χρόνου μπορούν επίσης να χρησιμοποιηθούν ειδικά στις επιδείξεις όπου ο στόχος είναι να παρουσιαστεί κάποια αναλογία μεταξύ του αλγορίθμου και μιας κατάστασης πραγματικού κόσμου. Η έρευνα δείχνει ότι τέτοιες

αναλογίες καθιστούν πρωταρχική την εμπειρία της μάθησης και ενισχύουν τις εκβάσεις της μάθησης (Hansen & Narayanan, 2000). Αξίζει να σημειωθεί ότι υπάρχουν επίσης συστήματα που χρησιμοποιούν ήχο για να δημιουργήσουν τα ακουστικά σχέδια των διαδικασιών του αλγορίθμου (συστήματα "ηχοποίησης αλγορίθμου", Brown & Hershberger, 1998). Πιο πρόσφατα έχουμε παραδείγματα εκπαιδευτικής υλοποίησης αλγορίθμων με τη χρήση προγραμματιζόμενων ρομπότ για να μιμηθούν τους μετασχηματισμούς του συνόλου δεδομένων (Lopez, Myller & Sutinen, 2004).

Ισοδυναμία, αντοχές και διαστάσεις των αναπαραστάσεων στα συστήματα AV

Ισοδυναμία (Equivalence): Συγκρίνοντας τις απεικονίσεις που εμφανίζονται συνήθως στα συστήματα AV κάποιος μπορεί να τις κατηγοριοποιήσει ως ισοδύναμες ή μη ισοδύναμες πληροφοριακά. Ως παράδειγμα των παραπάνω, εξετάστε την περίπτωση της αναπαράστασης ψευδοκώδικα (ή κώδικα προγραμματισμού) και της δυναμικής αναπαράστασης του αλγορίθμου ταξινόμησης (όπως απεικονίζεται στην εικόνα 2.3). Αυτές οι δύο αναπαραστάσεις αντιπροσωπεύουν γενικά τις ίδιες σχέσεις (βήματα του αλγορίθμου) αν και με το διαφορετικό τρόπο (στατικό κειμενικό εναντίον της δυναμικής οπτικής αντιπροσώπευσης). Σε άλλες περιπτώσεις οι αναπαραστάσεις δεν είναι ισοδύναμες με οποιουσδήποτε άλλες. Παραδείγματος χάριν, μια κοινή αναπαράσταση σχετικά με τους αλγορίθμους ταξινόμησης είναι μια γραφική παράσταση δύο διαστάσεων που απεικονίζει την κατάσταση του συνόλου δεδομένων ως συλλογή των σημείων. Κάθε σημείο αντιπροσωπεύει μια αξία στο σύνολο στοιχείων, Χ όντας ο δείκτης του πίνακα και το Υ η τιμή. Κατά το "τρέξιμο" του αλγορίθμου τα σημεία κινούνται προς τις κατάλληλες θέσεις τους ανάλογα με το μετασχηματισμό του συνόλου δεδομένων, και, συνεπώς, η δυναμική γραφική παράσταση μεταβιβάζει τον τρέχοντα χρόνο και τη μέθοδο που χρησιμοποιείται από κάθε αλγόριθμο. Ένα τέτοιο είδος αντιπροσώπευσης δεν είναι ισοδύναμο με οποιαδήποτε επίσημη βασισμένη στο κείμενο αντιπροσώπευση.

Αντοχές (Affordances): Οι οπτικοποιήσεις αναμένονται: (α) να περιορίσουν την αφαιρετικότητα, καθιστώντας συγκεκριμένο και αισθητό αυτό που ειδάλλως δεν θα ήταν εύκολα αντιληπτό ή διαθέσιμο για επιθεώρηση, και (β) να δομήσουν το συλλογισμό της δραστηριότητας των σπουδαστών (με την υποστήριξη κάποιου "βασισμένου στην εικόνα" συλλογισμού και ακολουθώντας την έννοια ότι μια αποτελεσματική επίδειξη-εικόνα μπορεί να διευκολύνει τον συλλογισμό του χρήστη).

Προοπτική (Perspective), Ακρίβεια (Precision), Πολυπλοκότητα (Complexity): Από αυτό που έχει συζητηθεί μέχρι τώρα πρέπει να είναι σαφές ότι οι αναπαραστάσεις στα συστήματα AV ποικίλλουν σημαντικά στην προοπτική που υιοθετούν. Ενώ οι παραδοσιακές κειμενικές αναπαραστάσεις προσφέρουν μια πιο επίσημη αναλυτική προοπτική που υπογραμμίζει τα λογικά βήματα του αλγορίθμου, οι οπτικοποιήσεις φέρνουν έξω δυναμικότερα χαρακτηριστικά γνωρίσματα, επιτρέποντας στους σπουδαστές να δοκιμάσουν πόσο γρήγορα ένας αλγόριθμος λειτουργεί επάνω στα δεδομένα και να εστιάσουν επίσης στη σύγκριση του τρέχοντος χρόνου σε συνάρτηση με την αποδοτικότητα των διάφορων αλγορίθμων. Ένα πρόσθετο ζήτημα της προοπτικής αυτής είναι ότι ενώ οι κειμενικές αναπαραστάσεις επιτρέπουν στους χρήστες να εστιάσουν μόνο σε ένα συγκεκριμένο σημείο του κώδικα κάθε φορά, οι οπτικοποιήσεις προσπαθούν να φέρουν τις μεγάλες δομές λογισμικού στο πλαίσιο μιας ενιαίας άποψης (κάτι σαν το "ελικόπτερο πάνω από το τοπίο" (Petre, Blackwell και πράσινος, 1998) προσφέροντας στους χρήστες την ευκαιρία να οπτικοποιήσουν τα σημαντικά δομικά χαρακτηριστικά της γενικής αρχιτεκτονικής λογισμικού.

Οι αναπαραστάσεις στα συστήματα AV μπορούν επίσης να ποικίλουν στην ακρίβεια. Παραδείγματος χάριν, μια επίδειξη-εικόνα(display) μπορεί να παρουσιάσει ακριβής ποσοτική πληροφορία για τις διαδικασίες του αλγορίθμου (π.χ. μέτρα αποδοτικότητας ή τον αριθμό των προσδοκώμενων συγκρίσεων δεδομένων) ενώ μια άλλη μπορεί να παρουσιάσει ποιοτικά την πληροφορία για την οπτικοποίηση των θεμελιωδών μεθόδων επεξεργασίας δεδομένων (π.χ. η συγκεκριμένη μέθοδος ταξινόμησης που εφαρμόζεται στα στοιχεία). Η πολυπλοκότητα μπορεί επίσης να είναι ένα σοβαρό πρόβλημα στο σχεδιασμό συστημάτων AV και, όπως αναφέρεται ήδη, οι σχεδιαστές υιοθετούν πολλαπλάσιες όψεις για να παρουσιάσουν τις συμπληρωματικές πληροφορίες προκειμένου να ελαχιστοποιηθεί η πολυπλοκότητα. Η χρησιμοποίηση μιας ενιαίας αντιπροσώπευσης θα ήταν ανεπαρκής για την παρουσίαση όλων πτυχών της περιοχής και θα έπρεπε να γίνει πάρα πολύ σύνθετη εάν έπρεπε να παρουσιάσει όλες τις πληροφορίες. Συνολικά, τα συστήματα οπτικοποίησης αλγορίθμου υιοθετούν τις πολλαπλάσιες κειμενικές και οπτικές αναπαραστάσεις για να ποικίλουν κατάλληλα στην προοπτική, την ακρίβεια και την πολυπλοκότητα της περιοχής.

Περιεχόμενο (Content), Επιμονή (Persistence), Μετασχηματισμός (Transformation), Μεταβλητότητα (Versatility)

Ο Brown (1988b) συζητώντας για τις επιδείξεις οπτικοποίησης αλγορίθμου πρότεινε ότι η οπτικοποίηση (γενικά) μπορεί να περιγραφεί χρησιμοποιώντας τρεις ανεξάρτητες διαστάσεις: Περιεχόμενο, Επιμονή και Μετασχηματισμό.

Περιεχόμενο(Content): Το περιεχόμενο μιας επίδειξης μπορεί να είναι άμεσο ή συνθετικό. Οι άμεσες αναπαραστάσεις είναι εκείνες που παράγονται άμεσα με την οπτικοποίηση των αντίστοιχων δομών δεδομένων ή κώδικα του προγράμματος. Κατά συνέπεια, η επίδειξη κατασκευάζεται από τη δομή δεδομένων (και αντίστροφα) που εφαρμόζει μια απλή διαδικασία χαρτογράφησης χωρίς οποιεσδήποτε άλλες πληροφορίες να είναι απαραίτητες. Οι συνθετικές αναπαραστάσεις, αντίθετα, παρουσιάζουν τις έννοιες που δεν συμπεριλαμβάνονται αρχικά (και επομένως παραχθείς από) στη δομή των στοιχείων ή οποιασδήποτε άλλης μεταβλητής προγράμματος. Είναι συνήθως κάποια μορφή αφαίρεσης των στοιχείων ή μια σύνθεση για να παρουσιάσουν πώς οι διαδικασίες προκαλούν αλλαγές στα στοιχεία.

Επιμονή (Persistence): αυτή η διάσταση αναφέρεται στην ιδιοκτησία της αντιπροσώπευσης για είτε να παρουσιάσει τις πληροφορίες απλά για την τρέχουσα κατάσταση των στοιχείων είτε να περιλάβει κάπως μια πλήρη ιστορία αυτού που έχει συμβεί μέχρι τώρα (αυτές είναι οι προηγούμενες καταστάσεις στοιχείων). Οι Ainsworth και VanLabeke (2004) προτείνουν ότι μπορούμε να διακρίνουμε μεταξύ των χρόνο-επίμονων, χρόνο-υπονοούμενων και χρόνο-μοναδικών αναπαραστάσεων:

Μια χρόνο-επίμονη αντιπροσώπευση (time-persistent (T-P)) περιλαμβάνει τον άξονα του χρόνου και επιτρέπει στους σπουδαστής να παρατηρήσουν τον τρόπο που μια (ή περισσότερες) μεταβλητές ποικίλλει στο χρόνο με την παρουσίαση των πρόσφατων και όλων των άλλων τιμών της μεταβλητής που υπολογίζονται μέχρι τώρα.

Μια χρόνο-υπονοούμενη αντιπροσώπευση (time-implicit (T-I)) δεν περιλαμβάνει ρητά το χρόνο αλλά παρουσιάζει τον τρόπο που η σχέση μεταξύ δύο μεταβλητών εξελίσσεται στον χρόνο.

Μια χρόνο-μοναδική αντιπροσώπευση (time-singular (T-S)) επιδεικνύει τις τιμές μίας (ή περισσότερων) μεταβλητών σε μια δεδομένη ενιαία στιγμή του χρόνου (επομένως δεν περιέχει οποιαδήποτε ιστορική πληροφορία της δυναμικής διαδικασίας).

Μετασχηματισμός (Transformation): οι διαστάσεις μετασχηματισμού κυμαίνονται από τις επιδείξεις που παρουσιάζουν αλλαγές στις εικόνες ασυνεχώς σε εκείνες που παρουσιάζουν επαυξητικές και συνεχείς αλλαγές. Οι διακριτές μεταβάσεις αντικαθιστούν απλά στην οθόνη τα παλαιά χαρακτηριστικά δεδομένων με νέα, καθώς οι επαυξητικοί μετασχηματισμοί παρουσιάζουν μία ομαλή μετάβαση μεταξύ της προηγούμενης και της επόμενης κατάστασης. Η χρησιμότητα των διακριτών και των επαυξητικών επιδείξεων εξαρτάται από το ποσό των δεδομένων που απεικονίζονται: οι διακριτοί μετασχηματισμοί τείνουν να είναι οι πιο χρήσιμοι σε μεγάλα σύνολα στοιχείων ενώ οι επαυξητικές μεταβάσεις είναι αποτελεσματικότερες όταν οι χρήστες εξετάζουν έναν αλγόριθμο που τρέχει σε ένα μικρό σύνολο δεδομένων (Brown, 1988b).

Μεταβλητότητα (Versatility): ανάλογα με τη μεταβλητότητά τους οι επιδείξεις AV μπορούν να περιγραφούν ως γενικές ή να προσαρμοσμένες. Μια γενική επίδειξη είναι μια η οποία αφού αναπτυχθεί αρχικά για έναν αλγόριθμο, μπορεί έπειτα να προσαρμοστεί εύκολα για να φιλοξενήσει τις αναπαραστάσεις άλλων σχετικών αλγορίθμων. Οι προσαρμοσμένες επιδείξεις, εντούτοις, κωδικοποιούνται σκληρά στις αναπαραστάσεις επίδειξης ενός συγκεκριμένου αλγορίθμου.

Σε ένα χαρακτηριστικό σύστημα AV υπάρχουν συνήθως διαθέσιμες χρόνο-επίμονες και χρόνο-μοναδικές αναπαραστάσεις. Η χρήση των δυναμικών ραβδιών για να επιδείξει πώς λειτουργεί ένας αλγόριθμος ταξινόμησης είναι μια χρόνο-μοναδική αντιπροσώπευση εάν δεν επιτρέπει στο χρήστη να το ξανατυλίξει προς τα πίσω και να δει τις προηγούμενες καταστάσεις των στοιχείων. Εντούτοις συστήνεται ιδιαίτερα (Ainsworth & Van Labeke, 2004) ότι οι αναπαραστάσεις πρέπει να σχεδιαστούν ως χρόνο-επίμονες έτσι ώστε οι σπουδαστές να μπορούν εύκολα να κινηθούν πίσω και να διαβιβάσουν μέσω των διάφορων καταστάσεων και να ελέγξουν τα σημεία του ενδιαφέροντός τους.

Τα στοιχεία για το σχεδιασμό των αναπαραστάσεων σε χρόνο-επίμονη μορφή προσφέρονται από τους Stasko και Lawrence (1998) οι οποίοι αναφέρουν ότι οι σπουδαστές προσδιόρισαν ως αρνητικές πτυχές λογισμικού την απουσία της δυνατότητας να περιηγηθούν μέσω της οπτικοποίησης κατά ένα πλαίσιο τη φορά και η ανικανότητα να ξανατυλίξουν και να επαναλάβουν (επανέλθετε στην προηγούμενη κατάσταση). Οι Ainsworth και VanLabeke (το 2004, σελ. 9) επίσης υπογραμμίζουν ότι “θα αναμέναμε ότι οι αναπαραστάσεις T-P θα βοηθήσουν τους σπουδαστής να εκτελέσουν τους στόχους που περιλαμβάνουν και τις τρέχουσες και τις προηγούμενες τιμές, με τη μείωση των

απαιτήσεων μνήμης που κρατούν τις προηγούμενες καταστάσεις στη μνήμη και να την ενσωματώσει με τρέχουσες". Τέλος, οι αναπαραστάσεις AV μπορούν να είναι και του γενικού και προσαρμοσμένου τύπου που εξαρτάται, φυσικά, από το είδος των πληροφοριών που παρουσιάζεται.

Οι λειτουργίες των πολλαπλών αναπαραστάσεων στα συστήματα οπτικοποίησης αλγορίθμου

Οι Ainsworth και VanLabeke (2004) τονίζουν τη σημασία του να γίνουν οι πολλαπλές αναπαραστάσεις διαθέσιμες στους σπουδαστής προκειμένου να τους επιτραπεί (και στους εκπαιδευτικούς) να χρησιμοποιήσουν ελαστικά εκείνες των αναπαραστάσεων που ταιριάζουν καλύτερα στους στόχους εκμάθησής τους. Η ευελιξία της μετατροπής μεταξύ των αναπαραστάσεων είναι ιδιαίτερα σημαντική για την κατανόηση των σύνθετων φαινομένων, είναι ένας γνωστικός στόχος που απαιτεί να διεξαχθούν διάφοροι τύποι συμπερασμάτων βασισμένοι στις ποιοτικές ή ποσοτικές πληροφορίες. Η χρησιμοποίηση των πολλαπλών αναπαραστάσεων στα συστήματα οπτικοποίησης αλγορίθμου αναμένεται να υποστηρίξει και τις τρεις θεμελιώδεις λειτουργίες (Ainsworth, 1999): (α)να συμπληρώνει ο ένας τον άλλον, (β) να περιορίζονται οι άγνωστες αναπαραστάσεις και (γ) να ενισχύεται η βαθύτερη κατανόηση της περιοχής από τους σπουδαστές.

Συμπλήρωμα (Complement): Οι πολλαπλές αναπαραστάσεις μπορούν να συμπληρώσουν η μια την άλλη όταν διαφέρουν είτε στις πληροφορίες που περιέχουν ή στις γνωστικές διαδικασίες που καθεμία υποστηρίζει. Στα συστήματα AV οι πολλαπλές αναπαραστάσεις επιδεικνύουν διαφορετικές όψεις της περιοχής και αυτό μπορεί να είναι συμφέρον για τους σπουδαστής σχετικά με τους στόχους που πρέπει να εκτελέσουν (π.χ. μια γραφική όψη των μεταβαλλόμενων καταστάσεων στο σύνολο των στοιχείων θα μπορούσε να υποστηρίξει αποτελεσματικότερα το είδος του συμπεράσματος που είναι απαραίτητο για έναν ορισμένο στόχο), ή τη στρατηγική που ακολουθούν (οι πολλαπλές αναπαραστάσεις μπορούν να ενθαρρύνουν την εναλλαγή των στρατηγικών εκμάθησης μεταξύ των σπουδαστών) ή τις μεμονωμένες προτιμήσεις τους. (οι σπουδαστές μπορούν να εργαστούν με την προτιμημένη αντιπροσώπευσή τους συμμετέχοντας έτσι στις δραστηριότητες εκμάθησης πιο ανυπόμονα). Οι αναπαραστάσεις γενικά αναμένονται να έχουν διαφορετική υπολογιστική αποτελεσματικότητα ανάλογα με την καταλληλότητά τους να υποστηρίξουν την επαγωγική διαδικασία του σπουδαστή για έναν συγκεκριμένο στόχο. Η ίδια η ουσία της προσπάθειας για την ανάπτυξη συστημάτων AV είναι η προσδοκία ότι οι δυναμικές οπτικές αναπαραστάσεις είναι πιο κατάλληλες για τη διευκόλυνση των σπουδαστών να

αναπτύξουν το είδος του συλλογισμού ο όποιος είναι κατάλληλος για τη βαθύτερη κατανόηση μιας τέτοιας σύνθετης περιοχής. Μια αντιπροσώπευση μπορεί να είναι καταλληλότερη για το πρόβλημα αλλά λιγότερο ομοειδής στο χρήστη, και έτσι μια δεύτερη αντιπροσώπευση να βοηθήσει το χρήστη να κατανοήσει την πρώτη. Η συνδυασμένη χρήση πολλαπλών αναπαραστάσεων μπορεί να βοηθήσει το χρήστη να επεκτείνει τις στρατηγικές αναζήτησης και συλλογισμού κατάλληλα. Εντούτοις, δεν είναι σαφές ακόμα ποιο είδος αντιπροσώπευσης μπορεί να είναι πιο συμφέρον για τα διάφορα είδη στόχων για τους οποίου προορίζονται τα συστήματα AV. Οι Petre, Blackwell και Green (1998) υπογραμμίζουν ότι "πρέπει να ξέρουμε περισσότερα για τις χρήσεις, για τους στόχους μέσα στις χρήσεις, και για τις αναπαραστάσεις για τους στόχους" (σελ. 455).

Περιορίστε (Constrain): Η προγενέστερη εξοικείωση των σπουδαστών με ορισμένες αντιπροσωπευτικές πτυχές μπορεί να τους στηρίξει στον περιορισμό της ερμηνείας των νέων αναπαραστάσεων. Παραδείγματος χάριν, οι σπουδαστές αναμένονται να είναι εξοικειωμένοι με την ιδέα ενός συνόλου δεδομένων και επομένως διαισθητικά να καταλάβουν ότι το ύψος των ραβδιών στην οθόνη αντιστοιχεί στις τιμές των αντιπροσωπευόμενων στοιχείων.

Κατασκευάστε (Construct): Οι πολλαπλάσιες αναπαραστάσεις στα συστήματα AV μπορούν να χρησιμοποιηθούν για να προωθήσουν τη βαθύτερη κατανόηση των σπουδαστών (να προωθήσουν την αφαίρεση). Αυτό συμβαίνει, παραδείγματος χάριν, όταν η συγκριτική επίδειξη των πολλαπλών αλγορίθμων στην οθόνη επιτρέπει στους σπουδαστές να βγάλουν συμπέρασμα για τη σχετική αποδοτικότητά τους, κερδίζοντας κατά συνέπεια μια βαθύτερη κατανόηση για τη γενική απόδοση του αλγορίθμου, κάτι που δεν θα ήταν δυνατό μόνο με τη μελέτη των απομονωμένων κειμενικών αναπαραστάσεων.

Διπλή κωδικοποίηση

Η διπλή κωδικοποίηση σημαίνει κυρίως ότι οι σχετικές λεκτικές και οπτικές πληροφορίες παρουσιάζονται στον σπουδαστή με έναν συντονισμένο τρόπο έτσι ώστε το διπλό κύκλωμα του εγκεφάλου επεξεργάζεται ταυτόχρονα τα διαφορετικά εξωτερικά ερεθίσματα. Η διπλή επεξεργασία οδηγεί στην οικοδόμηση αναφερόμενων συνδέσεων μεταξύ των προτύπων γνώσης βασισμένων στις λεκτικές και οπτικές πληροφορίες έτσι ώστε να παράγουν την καλύτερη μακροπρόθεσμη διατήρηση μνήμης.

Η εφαρμογή της διπλής υπόθεσης κωδικοποίησης στο σχέδιο AV θα σήμαινε ότι οι γλωσσικές πληροφορίες (κατά προτίμηση με αφηγηματική μορφή) θα πρέπει να συνοδεύσουν την παρουσίαση οθόνη της δυναμικής γραφικής παράστασης, που προσαρμόζεται στις σχετικές αρχές παρουσίασης (π.χ. συνάφεια, μορφή, πλεονασμός, συνοχή και εξατομίκευση). Ενσωματώνοντας τις αρχές της διπλής κωδικοποίησης στο σχέδιο συστημάτων AV, φαίνεται να είναι ένας ελπιδοφόρος ερευνητικός τομέας που αξίζει να εξερευνηθεί περαιτέρω (Hundhausen, Douglas & Stasko, 2002). Η εμπειρική αξιολόγηση τέτοιων συστημάτων έχει αναφερθεί ήδη στη βιβλιογραφία δείχνοντας τη βελτίωση στην ποιότητα της εκμάθησης (π.χ. Hansen, Schrimpsher & Narayanan, 1998).

Προβλήματα με τις πολλαπλές αναπαραστάσεις και τους τύπους υποστήριξης

Η αντιστάθμιση στη χρησιμοποίηση πολλαπλών αναπαραστάσεων είναι ότι για κάθε νέα αντιπροσώπευση που προστίθεται στο σύστημα, οι σπουδαστές πρέπει να καταλάβουν τη σύνταξή του (ερμηνεία του σχήματος και των χειριστών), να καταλάβουν τι αντιπροσωπεύεται και να είναι επίσης σε θέση να μεταφράσουν μεταξύ των αναπαραστάσεων. Για να υποστηρίξουν τους σπουδαστές που αντιμετωπίζουν την υπερφόρτωση μετάφρασης, οι σχεδιαστές συστημάτων AV μπορούν να προγραμματίσουν τα συστήματά τους να εκτελούν αυτόματα τη μετάφραση. Αυτό γενικά σημαίνει ότι όποτε ένα στοιχείο αλλάζει σε μια αντιπροσώπευση (π.χ. η αξία μιας μεταβλητής) το σύστημα ενημερώνει αυτόματα το αντίστοιχο στοιχείο σε οποιαδήποτε άλλη σχετική αντιπροσώπευση. Στην περίπτωση των συστημάτων AV μια πολύ κοινή τεχνική σχεδίου για την υποστήριξη της μετάφρασης είναι να δοθεί έμφαση ταυτόχρονα το μέρος του κώδικα που εκτελείται και των γραφικών οντοτήτων όπου αυτή η εκτέλεση έχει επιπτώσεις (βλ. εικόνα 2.3).

Ένας άλλος τρόπος υποστήριξης των σπουδαστών για να δρέψουν τα οφέλη της τεχνολογίας AV είναι να τους κάνουν να χρησιμοποιήσουν τα συστήματα οπτικοποίησης στα πλαίσια των συνεργαστικών δραστηριοτήτων, που τους ενθαρρύνουν να κατασκευάσουν και να παρουσιάσουν τις οπτικοποιήσεις τους. Σε τέτοιες εκπαιδευτικές τοποθετήσεις, τα συστήματα AV πρέπει να σχεδιαστούν κατά τέτοιο τρόπο ώστε να γίνονται εργαλεία κατασκευής των σπουδαστών, επιτρέποντας στους σπουδαστής να αναπτύξουν σε συνεργασία και να παρουσιάσουν τις προοριζόμενες οπτικοποιήσεις, βελτιώνοντας κατά συνέπεια τη μεμονωμένη επεξεργασία της εξωτερικής αντιπροσώπευσης.,

2.3 Παιδαγωγική αποτελεσματικότητα της οπτικοποίησης αλγορίθμου

2.3.1 Είναι η τεχνολογία AV χρήσιμη στην διδασκαλία;

Πρόκειται να δώσουμε μια απλή απάντηση σε αυτήν την ερώτηση ευθύς εξαρχής. Οι ερευνητές συμφωνούν ομόφωνα ότι τα συστήματα AV μπορούν να είναι σημαντικής χρησιμότητας εκμάθησης όταν χρησιμοποιούνται για να εμπλέκουν τους σπουδαστές στις ενεργές καταστάσεις εκμάθησης και όχι μόνο να τους θέσουν στη θέση ενός παθητικού θεατή που παρατηρεί απλά τις οπτικοποιήσεις των εμπειρογνωμόνων (π.χ. Hundhausen, Douglas and Stasko, 2002; Naps et al. , 2003; Stasko & Lawrence, 1998).

Για να επιτύχουν αυτήν την εμπλοκή έχουν αναφερθεί διάφορες τεχνικές, όπως το να έχουν τους σπουδαστές να εισάγουν και να μελετούν τη συμπεριφορά του αλγορίθμου στα δικά τους σύνολα δεδομένων εισόδου, προβλέποντας τις μελλοντικές καταστάσεις οπτικοποίησης, προγραμματίζοντας τον οπτικοποιημένο αλγόριθμο, απαντώντας στις ερωτήσεις σχετικά με την οπτικοποίηση και κατασκευάζοντας επίσης τις δικές τους οπτικοποιήσεις και παρουσιάζοντας τις στους συνομήλικους και τους εκπαιδευτικούς. Στο εξής τμήμα πρόκειται να παρουσιάσουμε αναλυτικά και να σχολιάσουμε τα αποτελέσματα από τις εμπειρικές μελέτες που υποστηρίζουν την ανωτέρω δηλωμένη διατριβή.

2.3.2 Πυκνότητα αναπαράστασης

Από ό,τι έχει παρουσιαστεί μέχρι τώρα, είναι σαφές ότι η απλή παρουσίαση των οπτικοποιήσεων αλγορίθμου δεν αναμένεται να προωθήσει ουσιαστικά την εκμάθηση σε σύγκριση με την εμπλοκή του σπουδαστή στην ενεργό εκμάθηση. Εντούτοις, άλλες μελέτες υποστηρίζουν την άποψη ότι η μάθηση ενισχύεται σημαντικά όταν τα οπτικά γλωσσικά (λεκτικά) στοιχεία των πληροφοριών είναι επιτυχώς κωδικοποιημένα σύμφωνα με την θεωρία διπλής κωδικοποίησης στον εγκέφαλο του σπουδαστή. Έχουν αναφερθεί διάφορα πειράματα (π.χ. Mayer, 2003) όπου η κατάλληλα οργανωμένη παρουσίαση του λεκτικού και οπτικού υλικού οδήγησε σε σημαντικά καλύτερες εκβάσεις μάθησης σε σύγκριση με την μάθηση βασισμένη μόνο στο κειμενικό υλικό μάθησης. Γιατί η απλή παρουσίαση των απεικονίσεων έχει επιπτώσεις με τόσο διαφορετικό τρόπο στην κατάσταση εκμάθησης; Οι ανωτέρω δύο γραμμές έρευνας οδηγούν σε αντιφατικά αποτελέσματα;

Πιστεύουμε ότι η απάντηση αυτών των ερωτήσεων θα καθιστούσε σαφές ότι η γνωστική επεξεργασία των εξωτερικών αναπαραστάσεων δεν μπορεί να είναι εξίσου αποδοτική σε όλες τις καταστάσεις. Για να καταλάβετε καλύτερα αυτό ρίξτε μια πιο στενή ματιά σε αυτό που στη βιβλιογραφία καλεί συνήθως ως "ενεργό μάθηση". Ο όρος "ενεργός" είναι παραπλανητικός στο σημείο που επιτρέπει σε κάποιον να υποθέσει ότι υπάρχει επίσης "παθητική" μάθηση.

Εντούτοις, όλη η εκμάθηση είναι ενεργός. Για να πραγματοποιηθεί εκμάθηση υπάρχει πάντα κάποια γνωστική δραστηριότητα που είναι ο λόγος για την ανάπτυξη των διανοητικών αναπαραστάσεων στη μακροπρόθεσμη μνήμη των σπουδαστών. Προφανώς, μπορούμε να μάθουμε ακόμη και με το να δούμε απλά μια εξωτερική οπτική αντιπροσώπευση, υπό τον όρο ότι φυσικά μπορεί να υποβληθεί αποτελεσματικά σε επεξεργασία. Εντούτοις το γνωστικό σύστημά μας εκθέτει συγκεκριμένους περιορισμούς ένας συγκεκριμένα από τους οποίους είναι η περιορισμένη ικανότητα της λειτουργικής μνήμης (βλ., παραδείγματος χάριν, Clark και Mayer, 2003). Όταν μια σύνθετη και απαιτητική εξωτερική οπτική αντιπροσώπευση παρουσιάζεται στους σπουδαστές για πρώτη φορά έπειτα το γνωστικό σύστημά τους είναι υπερφορτωμένο.

Για να περιγράψουμε καλύτερα αυτήν την κατάσταση είναι χρήσιμο να εισαγάγουμε την έννοια της "πυκνότητας αναπαράστασης". Μια αντιπροσώπευση για έναν σπουδαστή είναι πυκνή εάν, για κάποιους λόγους, ο σπουδαστής δοκιμάζει μια υψηλή γνωστική υπερφόρτωση κατά την προσπάθεια να υποβληθούν σε επεξεργασία οι πληροφορίες που μεταβιβάζονται από την αντιπροσώπευση. Αυτοί οι λόγοι μπορούν να περιλάβουν:

Αφαίρεση (Abstraction) (έλλειψη προγενέστερης γνώσης από τον σπουδαστή σχετικά με την οποία θα μπορέσει να "δέσει" την αφηρημένη αντιπροσώπευση).

Πολυπλοκότητα (Complexity Complexity) (πάρα πολλά αλληλένδετα κομμάτια πληροφορίας παρουσιάζονται να υπερφορτώνουν την λειτουργική μνήμη του σπουδαστή).

Μετάφραση (Translation) (οι πληροφορίες της περιοχής μοιράζονται μεταξύ των πολλαπλών αναπαραστάσεων και η μετάφραση μεταξύ τους είναι απαιτητική, υπερφορτώνοντας κατά συνέπεια επίσης τη λειτουργική μνήμη του σπουδαστή)

Κατά την απλή παρακολούθηση μιας πυκνής αντιπροσώπευσης η γνωστική επεξεργασία των πληροφοριών είναι δύσκολη εάν όχι αδύνατη για τους σπουδαστές. Οι εσωτερικές

γνωστικές λειτουργίες τους εμποδίζονται. Παραδείγματος χάριν, η ανάπτυξη ενός διανοητικού προτύπου βασισμένου στις οπτικές πληροφορίες και τη συμπληρωματική αναφερόμενη χαρτογράφηση καθυστερείται επειδή είναι δύσκολο για τον σπουδαστή να προετοιμάσει τις πληροφορίες στη λειτουργική μνήμη και να συμμετέχει αποτελεσματικά στην επιλογή, την οργάνωση, και τις γνωστικές διαδικασίες ολοκλήρωσης[2] .

Σε αυτή την περίπτωση οι σπουδαστές χρειάζονται πρόσθετη εμπειρία εκμάθησης για να αναλύσουν κατάλληλα και να επεξεργαστούν τις σύνθετες πληροφορίες και να κατασκευάσουν το είδος των εσωτερικών αναπαραστάσεων που είναι επαρκείς για την κατανόηση της πυκνής αντιπροσώπευσης (έτσι ώστε να τη μετατρέψει σε μία πιο "λεπτή", πιο διαφανή). Αντιθέτως, όταν οι σπουδαστές παρατηρούν μια αντιπροσώπευση "χαμηλής πυκνότητας" (η αντιπροσώπευση είναι απλή ή οι σπουδαστές εξοικειώνονται ήδη με αυτήν ή με μια κατηγορία παρόμοιων αναπαραστάσεων) οι γνωστικές διαδικασίες διευκολύνονται και η εκμάθηση συμβαίνει με την διανοητική επεξεργασία όλων των διαθέσιμων πληροφοριών. Η αναφερόμενη χαρτογράφηση γίνεται αποτελεσματικά και η διπλή κωδικοποίηση υποστηρίζει καλύτερα την αποστήθιση και την ανάκληση. Είναι και αυτή ενεργός μάθηση, με τη διαφορά ότι δεν υπάρχει καμία απαίτηση για πρόσθετες ενισχυτικές δραστηριότητες εκμάθησης.

Δεδομένου ότι η πυκνότητα μιας αντιπροσώπευσης ποικίλλει ανάλογα με την προγενέστερη γνώση του σπουδαστή και την πείρα του, είναι φυσικό να αναμένεται ότι μια απλή εικονική αναπαράσταση των καθημερινών συσκευών (όπως εκείνες που έχουν χρησιμοποιηθεί στα πειράματα που βεβαιώνουν την ανωτερότητα της συνδυασμένης κειμενικής και οπτικής παρουσίασης) είναι διαφανής για θέματα πειραμάτων, ενώ οι προηγμένες σύνθετες οπτικοποιήσεις στο υλικό της οπτικοποίησης αλγορίθμου είναι σημαντικά πυκνές για τους σπουδαστές που τις χρησιμοποιούν για πρώτη φορά. Επιπλέον, το γεγονός ότι η πυκνότητα αναπαράστασης μπορεί να ελαχιστοποιηθεί όταν ο σπουδαστής εξοικειώνεται με τη σύνταξή της, μας οδηγεί να υποθέσουμε ότι ακόμη και στην περιοχή της οπτικοποίησης αλγορίθμου η εκτενής εξοικείωση του σπουδαστή με το σχήμα αντιπροσώπευσης θα κάνει τις παρατηρήσεις αναπόφευκτα αρκετά διαφανείς έτσι ώστε η απλή τους παρουσίαση και η χρήση τους θα οδηγήσουν στην αποδοτική διπλή κωδικοποίηση και επομένως σε καλύτερη μάθηση. Αυτό είναι μια υπόθεση που μπορεί να εξεταστεί πειραματικά.

[2] Αυτές οι συγκεκριμένες γνωστικές λειτουργίες αναφέρονται στο πρότυπο για την εκμάθηση πολυμέσων βασισμένη στη διπλή υπόθεση κωδικοποίησης.

2.4 Συμπεράσματα

Τα συστήματα οπτικοποίησης αλγορίθμου χρησιμοποιούν τις πολλαπλές δυναμικές οπτικές αναπαραστάσεις για να υποστηρίξουν την εκμάθηση και την διδασκαλία στην περιοχή του αλγορίθμου στους υπολογιστές. Η τρέχουσα έρευνα δείχνει ότι αυτά τα συστήματα μπορούν αποτελεσματικά να οδηγήσουν σε καλύτερες εκβάσεις μάθησης υπό τον όρο ότι χρησιμοποιούνται με τρόπους που προωθούν την ενεργό εμπλοκή των σπουδαστών σε βαθύτερες καταστάσεις επεξεργασίας μάθησης. Η αποστολή των σχεδιαστών και των εκπαιδευτικών θα ήταν, επομένως, να υποστηρίξουν την αποδοτική εκμάθηση των σπουδαστών χρησιμοποιώντας ως μέσο (α) το σχεδιασμό της διεπιφάνειας χρήστη και των διαθέσιμων πολλαπλών αναπαραστάσεων με τέτοιο τρόπο ώστε η υπερφόρτωση των μεταφράσεων να περιορίζεται στο ελάχιστο, (β) σχεδιάζοντας το σύστημα AV με έναν τρόπο που να προωθεί την εμπλοκή των σπουδαστών στις ενεργές καταστάσεις εκμάθησης, και (γ) ενσωματώνοντας εργαλεία AV σε εμπειρίες εκμάθησης εποικοδομιστών για να βοηθήσει τους σπουδαστές να κατασκευάζουν, να μοιράζονται και να διαπραγματεύονται τις σημαντικές γι' αυτούς οπτικοποιήσεις.

ΚΕΦΑΛΑΙΟ 3 : Αλγόριθμοι ταξινόμησης (Sorting Algorithms)

Η ταξινόμηση αποτελεί μια σημαντική δραστηριότητα η οποία συναντάται σε όλους τους λαούς, στην επιστήμη, στην τεχνολογία, στην οικονομική και εμπορική ζωή αλλά και στην καθημερινή ζωή των ανθρώπων (Knuth, 1973; Linderson & Vitter, 1985; Bishop, 1988). Η ταξινόμηση συνδέεται με τη σύγκριση την ποσοτικοποίηση και γενικότερα τη μέτρηση ποιοτήτων που έχουν αξία και ενδιαφέρον και ως εκ τούτου αποκτά μεγάλη κοινωνική ισχύ (Bishop, 1988). Χαρακτηριστικά αναφέρεται ότι το 25% του συνολικού ενεργού χρόνου εργασίας των Η/Υ αφιερώνεται σε ταξινομήσεις ενώ ο καθημερινός χρόνος που διαθέτουν οι μεγάλες τράπεζες σε ταξινομήσεις λογαριασμών ανέρχεται στα δύο ώρες (Knuth, 1975; Linderson & Vitter, 1985). Η αξία της ταξινόμησης έγκειται στο ότι τα διατεταγμένα στοιχεία προσφέρουν αποτελεσματικότερες λύσεις για διάφορα προβλήματα. Τα στοιχεία αυτά μπορεί να είναι αριθμοί, ονόματα, αρχεία, εμβαδά, αντικείμενα. Σε όλες τις περιπτώσεις δεχόμαστε ότι περιέχουν μια μετρήσιμη πληροφορία. Η βασική πράξη στην ταξινόμηση είναι η σύγκριση στοιχείων και η εναλλαγή τους. Η γενικότερη αυτή σημασία της ταξινόμησης επιβάλλει την αυτοματοποίησή της μέσα από την κατασκευή γρήγορων αλγόριθμων εκτελέσιμων από υπολογιστές.

Επειδή η ταξινόμηση είναι πολύ σημαντική και πολύ χρονοβόρα, αποτέλεσε το θέμα εκτεταμένων ερευνών στην επιστήμη των υπολογιστών και αναπτύχθηκαν διάφορες στοιχειώδεις μεθόδους ταξινόμησης, οι οποίες είναι κατάλληλες είτε για μικρά αρχεία δεδομένων είτε για αρχεία με ειδική δομή. Οι αλγόριθμοι ταξινόμησης είναι αλγόριθμοι που τοποθετούν τα στοιχεία μιας λίστας με συγκεκριμένη σειρά, από τις οποίες οι πιο γνωστές είναι η αριθμητική και η λεξικογραφική σειρά.

Σημαντικό ρόλο για την ταξινόμηση των στοιχείων σε κάθε αλγόριθμο παίζει η υπολογιστική πολυπλοκότητα των συγκρίσεων των στοιχείων ανάλογα το μέγεθος της λίστας (n). Εάν το μέγεθος τη λίστας είναι μεγάλο τότε θεωρητικά ένας πιο απλός αλγόριθμος θα χρειαστεί περισσότερο χρόνο για να ταξινομήσει τη λίστα απ' ότι ένας πιο περίπλοκος αλγόριθμος. Για μερικούς αλγορίθμους ταξινόμησης η καλή συμπεριφορά είναι O(nlog(n)) και η κακή συμπεριφορά είναι O(n^2). Η ιδανική συμπεριφορά για μία ταξινόμηση είναι O(n). Οι περισσότεροι αλγόριθμοι ταξινόμησης έχουν χρονική πολυπλοκότητα, που ανήκει σε μία από τις επόμενες κατηγορίες:

- O(1). Κάθε εντολή του προγράμματος εκτελείται μία φορά ή το πολύ μερικές μόνο φορές. Στην περίπτωση αυτή λέγεται ότι ο αλγόριθμος είναι σταθερής πολυπλοκότητας.
- O(logn). Ο αλγόριθμος είναι λογαριθμικής πολυπλοκότητας. Με το "log" συμβολίζεται ο δυαδικός λογάριθμος.
- O(n). Η πολυπλοκότητα λέγεται γραμμική. Αυτή είναι η καλύτερη επίδοση για έναν αλγόριθμο που πρέπει να εξετάσει ή να δώσει στην έξοδο n στοιχεία.
- O(nlogn). Στην κατηγορία αυτή ανήκει μία πολύ σπουδαία οικογένεια αλγορίθμων ταξινόμησης.
- 0(n^2). Τετραγωνική πολυπλοκότητα. Πρέπει να χρησιμοποιείται μόνο για προβλήματα μικρού μεγέθους.
- 0(n^3). Κυβική πολυπλοκότητα. Και αυτοί οι αλγόριθμοι πρέπει να χρησιμοποιούνται μόνο για προβλήματα μικρού μεγέθους.
- 0(2^n).Σπάνια στην πράξη χρησιμοποιούνται αλγόριθμοι εκθετικής πολυπλοκότητας.

Οι τεχνικές ταξινόμησης χωρίζονται σε τρεις κατηγορίες, οι οποίες είναι οι εξής :

1. Τεχνικές ανταλλαγής (Exchange techniques). Σε αυτή τη κατηγορία ανήκουν οι μέθοδοι bubble sort και insertion sort, οι οποίες έχουν χρονική πολυπλοκότητα O(n^2)
2. Τεχνικές βασισμένες σε δομές δένδρου (Tree-based techniques), όπως η δυαδική δενδρική ταξινόμηση (binary tree sorting) οι οποίες έχουν χρονική πολυπλοκότητα O(nlog2n).
3. Αναδρομικές Τεχνικές (Recursive techniques), οι οποίες έχουν επίσης πολυπλοκότητα O(nlog2n), αλλά με δυνατότητα καλύτερης (πιο συμπαγούς) κωδικοποίησης και (δυναμικά) καλύτερης απόδοσης από άλλες τεχνικές.

3.1 Ταξινόμηση φυσαλίδας (Bubble Sort)

Ένας από τους πιο ευρέως χρησιμοποιημένους αλγορίθμους αναζήτησης είναι ο αλγόριθμος φυσαλίδας (bubble sort), ο οποίος είναι ένας απλός και ευθύς αλγόριθμος για την ταξινόμηση δεδομένων. Η ταξινόμηση φυσαλίδας είναι γνωστή για τους αργούς της χρόνους, είναι όμως η πιο απλή από τους υπόλοιπους αλγόριθμους και γι’ αυτό το λόγο είναι η πρώτη μέθοδος ταξινόμησης που μαθαίνουν οι περισσότεροι.

3.1.1 Περιγραφή

Ο αλγόριθμος bubble sort βασίζεται στην αρχή της σύγκρισης και ανταλλαγής ζευγών από γειτονικά στοιχεία, μέχρις ότου ταξινομηθούν όλα τα στοιχεία. Κάθε φορά μετακινείται το μικρότερο στοιχείο της ακολουθίας προς το αριστερό άκρο. Τα βήματα επαναλαμβάνονται μέχρι να ταξινομηθεί ολόκληρη η λίστα. Το όνομα του αλγόριθμου προέρχεται από τον τρόπο ταξινόμησης: τα μεγαλύτερα στοιχεία κατευθύνονται προς το τέλος, όπως οι φυσαλίδες που αναδύονται στην επιφάνεια.

Ο αλγόριθμος αυτός ξεκινά από την αρχή της λίστας και συγκρίνει τα πρώτα δύο στοιχεία. Αν το πρώτο στοιχείο είναι το μεγαλύτερο, τότε εναλλάσσει τη θέση των δύο στοιχείων. Έπειτα συνεχίζει να συγκρίνει τα υπόλοιπα ζεύγη, δηλαδή το δεύτερο στοιχείο με το τρίτο και ούτω καθεξής. Όταν φτάσει στο τέλος της λίστας, ξεκινά πάλι από την αρχή δουλεύοντας με τον ίδιο τρόπο και σταματά όταν σε ένα πέρασμα της λίστας δε γίνει καμία εναλλαγή στοιχείων.

3.1.2 Παράδειγμα λειτουργίας της bubble sort

Η ταξινόμηση φυσαλίδας λειτουργεί ως εξής : Ξεκινάτε από τα αριστερά και συγκρίνετε τα δύο στοιχεία του πίνακα στις θέσεις 1 και 2. Αν το στοιχείο στα αριστερά (στη θέση 1) είναι μεγαλύτερο, τα αντιμεταθέτετε. Αν το στοιχείο στα δεξιά είναι μεγαλύτερο, δεν κάνετε τίποτα. Μετά μετακινείστε μία θέση και συγκρίνετε τα στοιχεία στις θέσεις 2 και 3. Αν πάλι το στοιχείο αριστερά είναι μεγαλύτερο, τα αντιμεταθέτετε. Αν το στοιχείο στα δεξιά είναι μεγαλύτερο, δεν κάνετε τίποτα.

Η διαδικασία αυτή συνεχίζεται σε όλο το μήκος της γραμμής μέχρι να φτάσετε στο τελευταίο στοιχείο του πίνακα. Μετά το πρώτο πέρασμα του πίνακα, έχετε κάνει N – 1 συγκρίσεις (N : το μέγεθος των στοιχείων του πίνακα). Το στοιχείο στο τέλος του πίνακα έχει ταξινομηθεί και δεν θα μετακινηθεί ξανά.

Επιστρέφετε στην αρχή του πίνακα και ξεκινάτε πάλι την ίδια διαδικασία για το δεύτερο, τρίτο... πέρασμα, μόνο που για κάθε πέρασμα το μέγεθος του πίνακα θα μειώνεται κατά ένα.

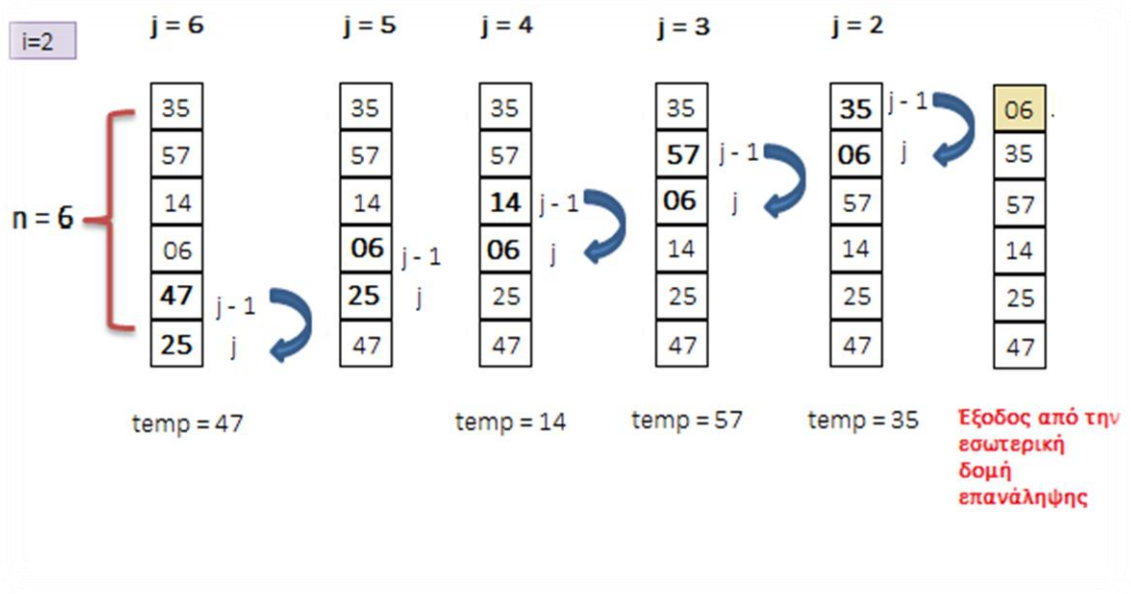
Στην εικόνα, βλέπουμε τον ψευδοκώδικα του αλγορίθμου της ταξινόμησης φυσαλίδας. Η μεταβλητή table είναι ένας πίνακας με n ακεραίους αριθμούς που πρέπει να ταξινομηθούν. Η μέθοδος αυτή αποτελείται από μία διπλή δομή επανάληψης (for loop) και από μία απλή δομή επιλογής (if).

```
Αλγόριθμος Φυσαλίδα
Δεδομένα // table, n //
Για i από 2 μέχρι n
    Για j από n μέχρι i με_βήμα -1
        Αν table[j - 1] > table[j] τότε
            temp ← table[j - 1]
            table[j - 1] ← table[j]
            table[j] ← temp
        Τέλος_Αν
    Τέλος_επανάληψης
Τέλος_επανάληψης
Αποτελέσματα // table //
Τέλος Φυσαλίδα
```

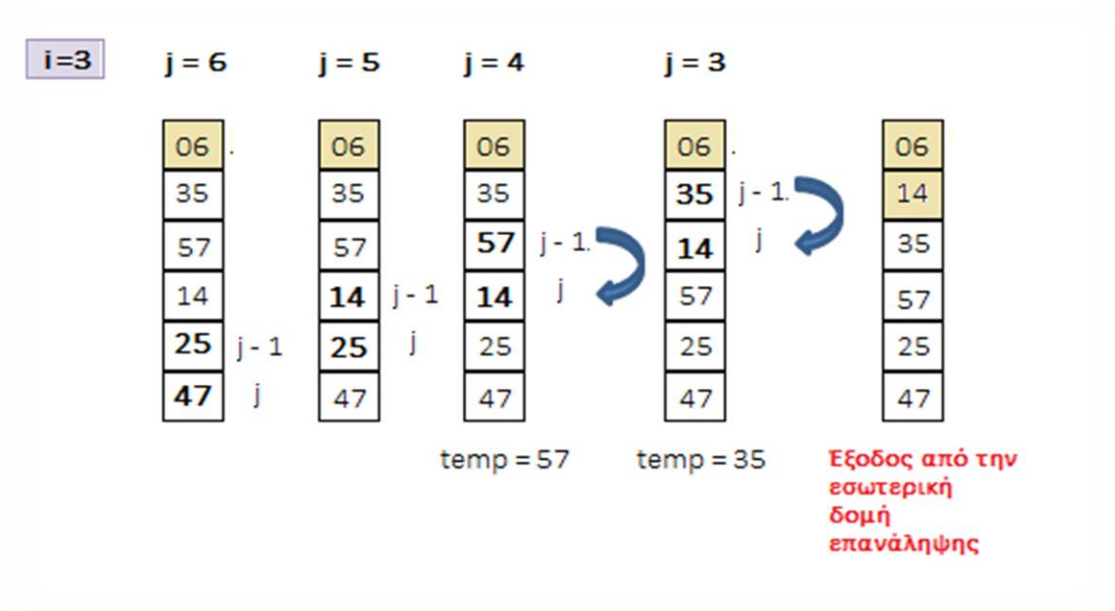
Παρακάτω θα αναλύσουμε ένα παράδειγμα λειτουργίας της bubble sort σε ταξινόμηση κατά αύξουσα σειρά, δηλαδή θα ταξινομηθούν από το μικρότερο προς το μεγαλύτερο. Έχουμε ένα μονοδιάστατο πίνακα N έξι θέσεων :

35	57	14	06	47	25

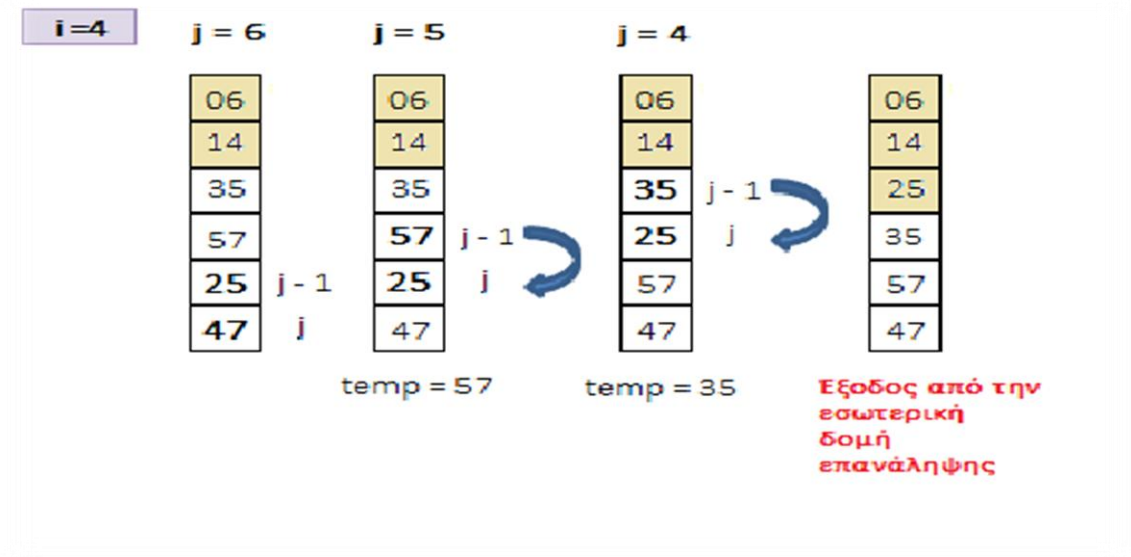
Αρχικά, θα εκτελεστεί η εξωτερική δομή επανάληψης, όπου το i θα πάρει τη τιμή 2. Στην εσωτερική δομή επανάληψης το j θα πάρει την τιμή 6, δηλαδή το μέγεθος του πίνακα και θα μειώνεται σταδιακά μέχρι να φτάσει την τιμή του i.



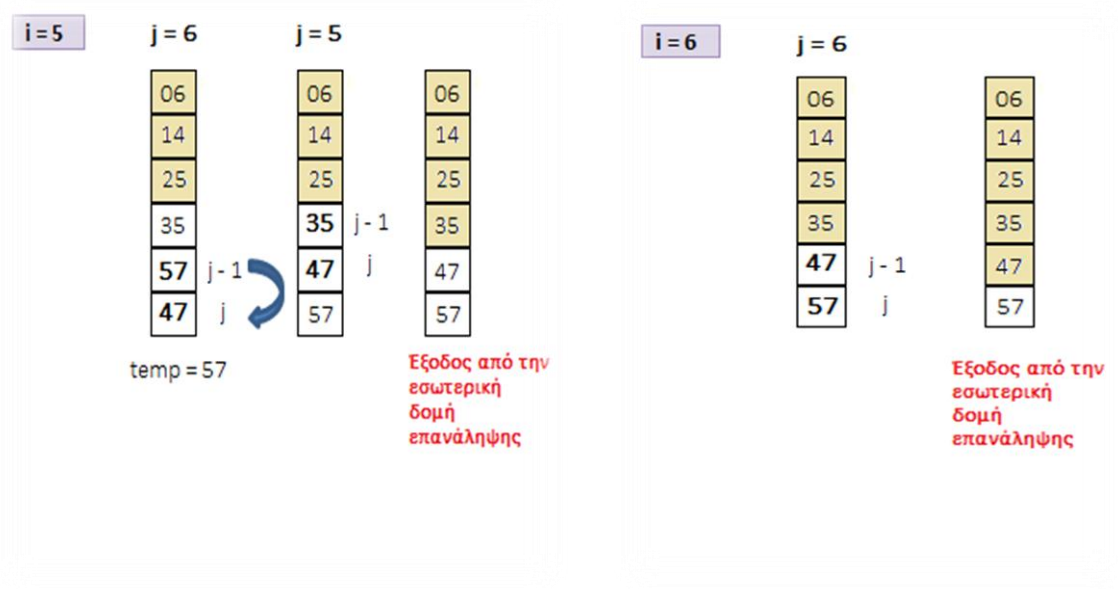
Στο δεύτερο πέρασμα το i θα έχει τιμή 3 και η δεύτερη επαναληπτική δομή θα αρχίσει να εκτελείται μέχρι το j να φτάσει τη τιμή του i.



Στο τρίτο πέρασμα το i θα αυξηθεί και θα έχει τη τιμή 3 και η δεύτερη επαναληπτική δομή θα αρχίσει να εκτελείται μέχρι το j να φτάσει τη τιμή του i.



Στο πέμπτο πέρασμα, η εσωτερική δομή επανάληψης θα εκτελεστεί μόνο μία φορά. Από τη στιγμή που ταξινομήσαμε τα πέντε στοιχεία από τα έξι, ο πίνακας θεωρείται ταξινομημένος.



Τέλος, σκοπός είναι να κατανοήσουμε την χρησιμότητα της μεταβλητής i, η οποία χρησιμοποιείται για να μετράει τα λεγόμενα περάσματα. Στο παραπάνω παράδειγμα, παρατηρούμε ότι στο τέλος κάθε περάσματος ταξινομείται και ένα στοιχείο από τον πίνακα. Το μέγεθος των στοιχείων του πίνακα είναι n=6 και χρειαστήκαμε πέντε περάσματα, δηλαδή n-1 περάσματα για να ταξινομηθεί ο πίνακας.

3.1.3 Υλοποίηση σε Java

Στο πρόγραμμα BubbleSort.java θα δείτε ότι δημιουργεί έναν πίνακα «intArray[]», ο οποίος περιέχει έξι μεταβλητές τύπου «int» (Integer).

```
// Παράδειγμα της Bubble Sort σε Java
// Αυτό το παράδειγμα δείχνει πως ταξινομούμε ένα πίνακα με ακεραίους αριθμούς
// χρησιμοποιώντας τον αλγόριθμο bubble sort. Η ταξινόμηση φυσαλίδας είναι ο πιο απλός
// αλγόριθμος ταξινόμησης.
//-----------------------------------------------------------------------------------------------------
public class BubbleSort {
    public static void main(String[] args) {

//Δημιουργεί ένα πίνακα με 6 ακέραιους αριθμούς σε τυχαία σειρά
        int intArray[] = new int[]{5,90,35,45,150,3};

//Εκτυπώνει τον πίνακα πριν υλοποιηθεί ο αλγόριθμος Bubble Sort
        System.out.println("Ο πίνακας πριν την ταξινόμηση :");
        for(int i=0; i < intArray.length; i++){
```

```
            System.out.print(intArray[i] + " ");
        }

//καλεί την μέθοδο "bubbleSort" για να ταξινομήσει τον πίνακα
        bubbleSort(intArray);
        System.out.println("");

//Εκτυπώνει τον πίνακα μετά την ταξινόμηση
        System.out.println("Ο πίνακας μετά την ταξινόμηση :");
        for(int i=0; i < intArray.length; i++){
            System.out.print(intArray[i] + " ");
        }
    }
//-----------------------------------------------------------------------------------------------------------
    private static void bubbleSort(int[] intArray) {

        int n = intArray.length;
        int temp = 0;

        for(int i=0; i < n; i++){                    // εξωτερικός βρόγχος
            for(int j=1; j < (n-i); j++){            // εσωτερικός βρόγχος
                if(intArray[j-1] > intArray[j]){     // αταξινόμητο ;
                    temp = intArray[j-1];            //κάνε αντιμετάθεση
                    intArray[j-1] = intArray[j];
                    intArray[j] = temp;
                }
            }
        }
    }
}
//-----------------------------------------------------------------------------------------------------------
```

Στη ρουτίνα main() δημιουργούμε ένα πίνακα με έξι στοιχεία σε τυχαία σειρά, εμφανίζει τον πίνακα, καλεί τη bubbleSort(int[] intArray) για να ταξινομήσει τα στοιχεία στον πίνακα και τον εμφανίζει πάλι. Η έξοδος της ταξινόμησης φυσαλίδας θα είναι :

Ο πίνακας πριν την ταξινόμηση :

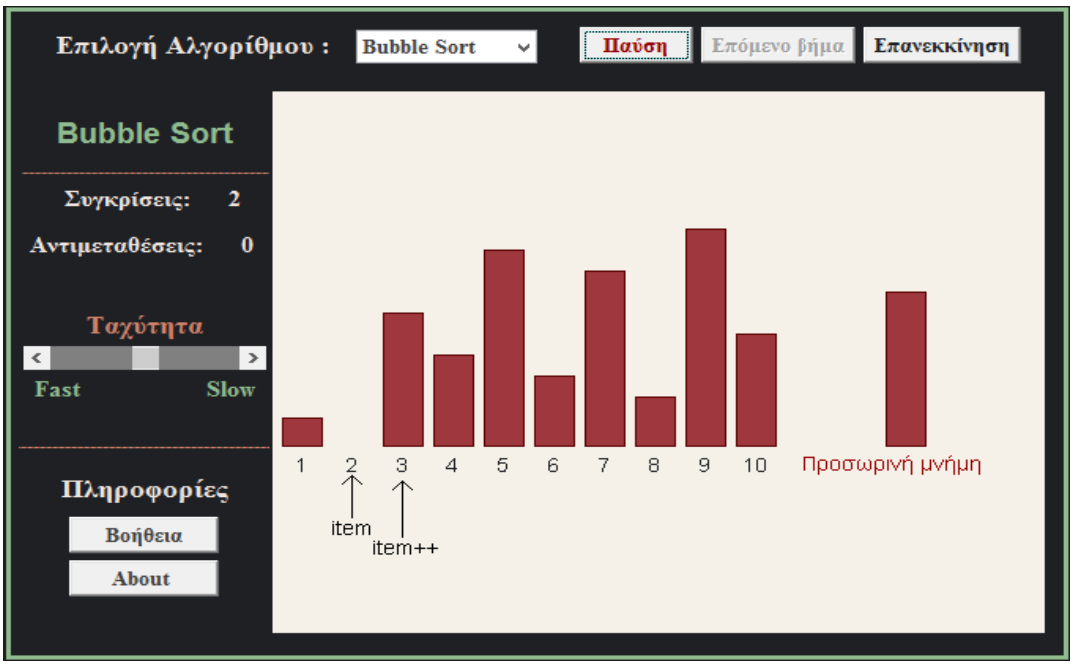
5 90 35 45 150 3

Ο πίνακας μετά την ταξινόμηση :

3 5 35 45 90 150

3.1.4 Η εφαρμογή BubbleSort

Στη εικόνα βλέπετε την εφαρμογή BubbleSort. Αν την εκτελέσετε θα δείτε να μοιάζει με γράφημα ράβδου, με τα ύψη των ράβδων τυχαία διευθετημένα.



Αυτή η εφαρμογή περιέχει ένα γράφημα δύο ταχυτήτων. Μπορείτε να την αφήσετε να εκτελείται μόνη της (κάνοντας κλικ στο κουμπί «Εκκίνηση») ή μπορείτε να προχωράτε βήμα – βήμα (κάνοντας κλικ στο κουμπί «Επόμενο βήμα»). Για να κάνετε άλλη μία ταξινόμηση, πατήστε το κουμπί «Επανεκκίνηση», όπου θα δημιουργήσει ένα νέο σύνολο ράβδων και θα αρχικοποιήσει τη ρουτίνα ταξινόμησης.

Αν εκτελέσουμε την εφαρμογή κάνοντας κλικ το κουμπί «Επόμενο βήμα» μπορούμε να δούμε πραγματικά πως λειτουργεί ο αλγόριθμος BubbleSort. Θα δείτε δύο βέλη που δείχνουν σε διαφορετικούς ράβδους. Τα δύο βέλη ονομάζονται «item» και «item++» και βρίσκονται το ένα δίπλα στο άλλο στην αριστερή άκρη του γραφήματος. Κάθε φορά που κάνετε κλικ στο κουμπί «Επόμενο βήμα» θα δείτε τα βέλη να μετακινούνται μαζί μια θέση προς τα δεξιά, αντιμεταθέτοντας αν χρειάζεται τις ράβδους. Κάθε φορά που χρειάζεται να αντιμετατεθούν δυο ράβδοι, η μία ράβδος τοποθετείται στη «Προσωρινή μνήμη» για να μπορέσει να γίνει η αλλαγή. Αν η ψηλότερη είναι στα αριστερά, θα γίνει η αντιμετάθεση.

Τα μηνύματα στο δεξιό μέρος της εφαρμογής δείχνουν πόσες συγκρίσεις και αντιμεταθέσεις έχουν γίνει μέχρι τώρα. Μία πλήρης ταξινόμηση δέκα ράβδων απαιτεί 45 συγκρίσεις και κατά μέσο όρο 22 αντιμεταθέσεις.

3.1.5 Απόδοση της ταξινόμησης φυσαλίδας

Στην εφαρμογή BubbleSort όπου έχουμε 10 ράβδους βλέπουμε ότι στο πρώτο πέρασμα κάνουν εννέα συγκρίσεις, στο δεύτερο πέρασμα κάνουν οχτώ συγκρίσεις και ούτω καθ' εξής μέχρι να φτάσουμε σε μία σύγκριση στο τελευταίο πέρασμα. Για δέκα στοιχεία θα είναι : 9+8+7+6+5+4+3+2+1 = 45.

Γενικά, όταν N είναι ο αριθμός των στοιχείων στον πίνακα, υπάρχουν N-1 συγκρίσεις στο πρώτο πέρασμα, N-2 συγκρίσεις στο δεύτερο πέρασμα και ούτω καθ' εξής. Ο τύπος για το σύνολο μιας τέτοιας σειράς είναι : (N-1)+(N-2)+(N-3)+...+1 = N* (N-1)/2. Έτσι, ο αλγόριθμος εκτελεί περίπου N^2/2 συγκρίσεις (αγνοώντας το -1, το οποίο δεν προκαλεί μεγάλες διαφορές, ειδικά αν το N είναι μεγάλο).

Οι αντιμεταθέσεις είναι λιγότερες από τις συγκρίσεις επειδή δύο ράβδοι αντιμετατίθενται μόνο αν πρέπει. Αν τα δεδομένα είναι τυχαία, μια αντιμετάθεση είναι απαραίτητη περίπου τις μισές φορές, θα γίνουν λοιπόν περίπου N^2/4 αντιμεταθέσεις. Αν και στη χειρότερη περίπτωση, με τα αρχικά δεδομένα αντιστρόφως ταξινομημένα, μία αντιμετάθεση είναι απαραίτητη με κάθε σύγκριση.

Οι αντιμεταθέσεις και οι συγκρίσεις είναι ανάλογες του N^2. Επειδή οι σταθερές δεν υπολογίζονται, μπορούμε να αγνοήσουμε το 2 και το 4 και να θεωρήσουμε ότι η ταξινόμηση φυσαλίδας εκτελείται σε χρόνο O(N^2). Αν τα στοιχεία είναι περισσότερα, τότε ο χρόνος εκτέλεσης του αλγορίθμου θα είναι μεγαλύτερος.

Οπότε βλέπετε ένα βρόγχο ενσωματωμένο σε άλλο, μπορείτε να καταλάβετε ότι ένας αλγόριθμος εκτελείται σε χρόνο O(N^2). Ο εξωτερικός βρόγχος εκτελείται N φορές και ο εσωτερικός βρόγχος N φορές (ή ίσως N διά κάποιας σταθεράς) για κάθε κύκλο του εξωτερικού βρόγχου.

3.2 Ταξινόμηση εισαγωγής (Insertion Sort)

Η ταξινόμηση “bubble sort” γενικά δεν είναι κατάλληλη γιατί είναι πολύ αργή. Χρειαζόμαστε μία τεχνική η οποία χρησιμοποιεί μόνο λίγο επιπλέον χώρο αλλά είναι πιο γρήγορη από την “bubble sort”. Μία τέτοια αποτελεσματική μέθοδος είναι η ταξινόμηση με εισαγωγή. Ο αλγόριθμος εισαγωγής (insertion sort) είναι ένας απλός αλγόριθμος ταξινόμησης που είναι σχετικά αποτελεσματικός σε μικρές λίστες και σχεδόν ταξινομημένες λίστες και συνήθως χρησιμοποιείται ως μέρος άλλων πιο περίπλοκων αλγόριθμων. Ο αλγόριθμος παίρνει ένα ένα τα στοιχεία της λίστας και τα τοποθετεί σε μια νέα λίστα σε ταξινομημένη σειρά. Η λειτουργία της εισαγωγής είναι, όμως, «ακριβή» γιατί απαιτεί τη μετατόπιση των υπόλοιπων στοιχείων μία θέση αριστερά (shift left).

3.2.1 Περιγραφή

Ο αλγόριθμος ταξινόμησης με εισαγωγή λειτουργεί σε φάσεις. Διατηρεί στο αριστερό μέρος του πίνακα ένα ταξινομημένο υποσύνολο των στοιχείων. Σε κάθε φάση το επόμενο στοιχείο τοποθετείται στη σωστή θέση ανάμεσα στα ταξινομημένα στοιχεία αυξάνοντας το μέγεθος του ταξινομημένου υποσυνόλου κατά ένα. Παίρνουμε κάθε στοιχείο και το τοποθετούμε στη σωστή θέση στον ταξινομημένο πίνακα αριστερά του τρέχοντος στοιχείου. Εάν ξεκινήσουμε από το πρώτο στοιχείο, τότε αφού δεν υπάρχουν άλλα στοιχεία αριστερά, το στοιχείο αυτό βρίσκεται στη σωστή θέση (μέχρι τώρα). Επομένως δε χρειάζεται να ξεκινήσουμε από το πρώτο στοιχείο.

3.2.2 Παράδειγμα λειτουργίας της insertion sort

Η ταξινόμηση εισαγωγής λειτουργεί ως εξής : Το πρόγραμμα χρησιμοποιεί δύο βρόχους επανάληψης για να ταξινομήσει π.χ. έναν πίνακα ακεραίων. Ο εξωτερικός βρόχος, ελέγχει τον δείκτη του πίνακα όπου θα εισαχθεί η επόμενη τιμή. Ο εσωτερικός βρόχος ελέγχει την τρέχουσα τιμή η οποία θα εισαχθεί με τιμές αριστερά του πίνακα. Αν η τρέχουσα τιμή είναι μικρότερη από την τιμή στη θέση i τότε η τιμή αυτή μετακινείτε δεξιά. Η μετακίνηση συνεχίζει και με τις υπόλοιπες τιμές. Κάθε επανάληψη του εξωτερικού βρόχου προσθέτει μια ακόμη τιμή στο ταξινομημένο υποσύνολο του πίνακα.

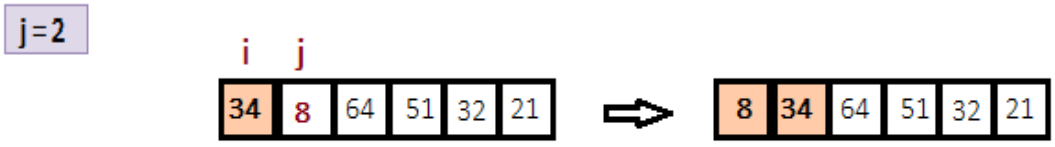
Στην εικόνα, βλέπουμε τον ψευδοκώδικα του αλγορίθμου της ταξινόμησης με εισαγωγή. Η μεταβλητή table είναι ένας πίνακας με n ακεραίους αριθμούς που πρέπει να ταξινομηθούν. Η μέθοδος αυτή αποτελείται από 2 δομές επανάληψης (for loop, while).

```
Αλγόριθμος Εισαγωγής
Δεδομένα // table, n //
Για j από 2 μέχρι n
        key ← table[ j ]
        i ← j - 1
        Ενώ i > 0 και table[ i ] > key κάνε
                table[ i + 1] ← table[ i ]
                i ← i - 1
        table[ i + 1] ← key
Τέλος_επανάληψης
Αποτελέσματα // table //
Τέλος Εισαγωγής
```

Παρακάτω θα αναλύσουμε ένα παράδειγμα λειτουργίας της insertion sort σε ταξινόμηση κατά αύξουσα σειρά, δηλαδή θα ταξινομηθούν από το μικρότερο προς το μεγαλύτερο. Έχουμε ένα μονοδιάστατο πίνακα Ν έξι θέσεων :

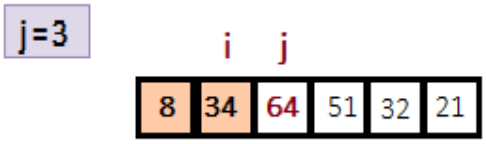
34	08	64	51	32	21

Αρχικά, θα εκτελεστεί η εξωτερική δομή επανάληψης, όπου το j είναι στη $2^{η}$ θέση. Το δεύτερο στοιχείο (8) έχει αριστερά του ένα πίνακα με ένα μόνο στοιχείο (34), ο οποίος αφού έχει ένα μόνο στοιχείο είναι ταξινομημένος. Τοποθετούμε το 8 στη σωστή θέση (πριν το 34) και το 34 μετακινείται μία θέση δεξιά. Έτσι ο πίνακας γίνεται:



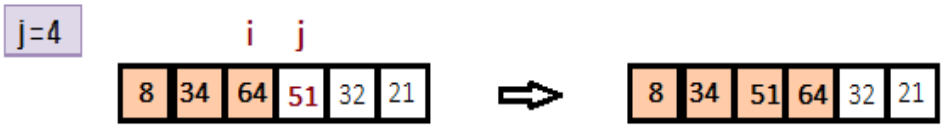
Το i είναι μεγαλύτερο του j (34 > 8). Τοποθετούμε το 8 στη σωστή του θέση και το "34" μετακινείται μια θέση δεξιά.

Συνεχίζουμε με το επόμενο-τρίτο στοιχείο (64) το οποίο πρέπει να μείνει στη θέση που βρίσκεται αφού είναι μεγαλύτερο από το 34.



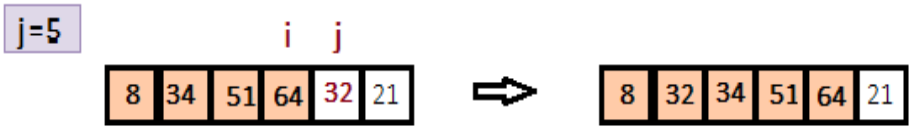
Το i είναι μικρότερο του j (34 < 64). Άρα ο πίνακας παραμένει ίδιος.

Συνεχίζουμε με το τέταρτο στοιχείο (51) το οποίο για να τοποθετηθεί στη σωστή θέση θα πρέπει να πάει μετά το 34 και το 64 να μετακινηθεί δεξιά μέχρι τη θέση του τρέχοντος στοιχείου.



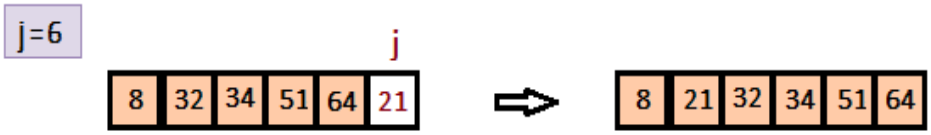
Το j είναι μικρότερο του i (51<64). Τοποθετούμε το στοιχείο "51" στη σωστή του θέση και μετακινούμε μία θέση δεξιά το στοιχείο "64".

Συνεχίζουμε με το επόμενο-πέμπτο στοιχείο (32) το οποίο για να τοποθετηθεί στη σωστή θέση θα πρέπει να πάει μετά το 8 και τα επόμενα στοιχεία πρέπει να μετακινηθούν δεξιά μέχρι τη θέση του τρέχοντος στοιχείου. Έτσι ο πίνακας ταξινομείται σιγά-σιγά, ένα στοιχείο κάθε φορά και γίνεται:



Το 5ο στοιχείο (32) πρέπει να τοποθετηθεί στη 2η θέση του πίνακα και όλα τα υπόλοιπα να μετακινηθούν μία θέση δεξιά μέχρι τη θέση του τρέχοντος στοιχείου.

Τέλος, το έκτο στοιχείο (21), το οποίο για να τοποθετηθεί στη σωστή θέση θα πρέπει να πάει μετά το 8 και τα επόμενα στοιχεία πρέπει να μετακινηθούν δεξιά μέχρι τη θέση του τρέχοντος στοιχείου. Έτσι ο πίνακας ταξινομείται σιγά-σιγά, ένα στοιχείο κάθε φορά και γίνεται:



Το 6ο στοιχείο (21) πρέπει να τοποθετηθεί στη 2η θέση του πίνακα και όλα τα υπόλοιπα στοιχεία να μετακινηθούν μία θέση δεξιά, μέχρι τη θέση του τρέχοντος στοιχείου.

3.2.3 Υλοποίηση σε Java

Στο πρόγραμμα InsertionSort.java θα δείτε ότι δημιουργεί έναν πίνακα «intArray[]», ο οποίος περιέχει έξι μεταβλητές τύπου «int» (Integer).

```
// Παράδειγμα της Insertion Sort σε Java
// Αυτό το παράδειγμα δείχνει πως ταξινομούμε ένα πίνακα με ακεραίους αριθμούς
// χρησιμοποιώντας τον αλγόριθμο Insertion sort.
//-----------------------------------------------------------------------------------------------------
public class InsertionSort {
   public static void main(String[] args) {

//Δημιουργεί ένα πίνακα με 6 ακέραιους αριθμούς σε τυχαία σειρά
        int intArray[] = new int[]{5,90,35,45,150,3};

//Εκτυπώνει τον πίνακα πριν υλοποιηθεί ο αλγόριθμος Insertion Sort
        System.out.println("Ο πίνακας πριν την ταξινόμηση :");
        for(int i=0; i < intArray.length; i++){
            System.out.print(intArray[i] + " ");
        }

//καλεί την μέθοδο "insertionSort" για να ταξινομήσει τον πίνακα
        insertionSort(intArray);
        System.out.println("");

//Εκτυπώνει τον πίνακα μετά την ταξινόμηση
        System.out.println("Ο πίνακας μετά την ταξινόμηση :");
        for(int i=0; i < intArray.length; i++){
            System.out.print(intArray[i] + " ");
        }
    }
//-----------------------------------------------------------------------------------------------------
   private static void insertionSort(int intArray[]){
/*
  Η Insertion Sort χρησιμοποιεί δύο βρόχους επανάληψης για να ταξινομήσει έναν πίνακα
  ακεραίων.  Ο εξωτερικός βρόχος, ελέγχει τον δείκτη του πίνακα όπου θα εισαχθεί η επόμενη
  τιμή.  Ο εσωτερικός βρόχος ελέγχει την τρέχουσα τιμή η οποία θα εισαχθεί με τιμές αριστερά
  του πίνακα.  Αν η τρέχουσα τιμή είναι μικρότερη από την τιμή στη θέση position τότε η τιμή
  αυτή μετακινείτε δεξιά.  Η μετακίνηση συνεχίζει και με τις υπόλοιπες τιμές.
```

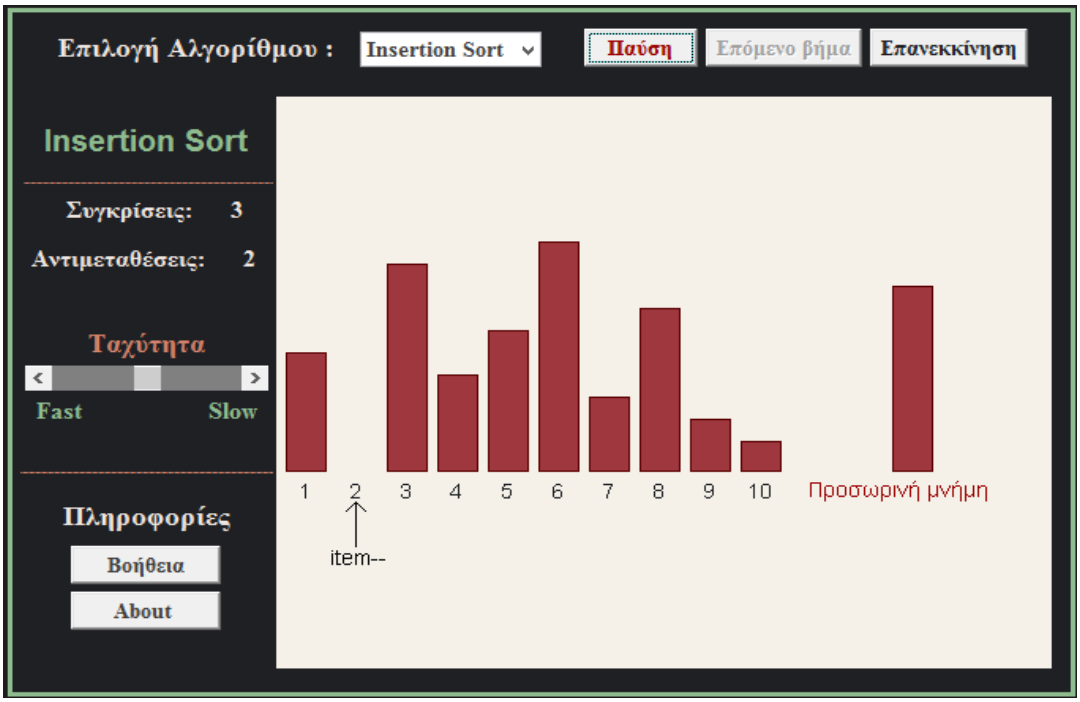
```
   Κάθε επανάληψη του εξωτερικού βρόχου προσθέτει μια ακόμη τιμή στο ταξινομημένο
υποσύνολο του πίνακα.
*/
        int temp;
        int pos;

        for (int i = 1; i < intArray.length; i++){                    // το i χωρίζει τη γραμμή
            temp = intArray[i];                                       // απομάκρυνε το σημαδεμένο στοιχείο
            pos = i - 1;                                              // ξεκίνα τις μετατοπίσεις
            while ((pos >= 0) && (temp < intArray [pos])){            // μέχρι να βρεθεί ένα μικρότερο
                intArray[pos + 1] = intArray[pos];                    // μετατόπισε το στοιχείο δεξιά
                pos--;                                                // πήγαινε μία θέση αριστερά
                intArray[pos + 1] = temp;          // εισάγαγε το απομακρυσμένο στοιχείο
            }
        }
   }
}
//-----------------------------------------------------------------------------------------------------
```

Στη ρουτίνα main() δημιουργούμε ένα πίνακα με έξι στοιχεία σε τυχαία σειρά, εμφανίζει τον πίνακα, καλεί τη bubbleSort(int[] intArray) για να ταξινομήσει τα στοιχεία στον πίνακα και τον εμφανίζει πάλι. Η έξοδος της ταξινόμησης με εισαγωγή θα είναι :

Ο πίνακας πριν την ταξινόμηση :
5 90 35 45 150 3
Ο πίνακας μετά την ταξινόμηση :
3 5 35 45 90 150

3.2.4 Η εφαρμογή InsertionSort

Στη εικόνα βλέπετε την εφαρμογή InsertionSort. Αν την εκτελέσετε θα δείτε να μοιάζει με γράφημα ράβδου, με τα ύψη των ράβδων τυχαία διευθετημένα.



Αυτή η εφαρμογή περιέχει ένα γράφημα δύο ταχυτήτων. Μπορείτε να την αφήσετε να εκτελείται μόνη της (κάνοντας κλικ στο κουμπί «Εκκίνηση») ή μπορείτε να προχωράτε βήμα – βήμα (κάνοντας κλικ στο κουμπί «Επόμενο βήμα»). Για να κάνετε άλλη μία ταξινόμηση, πατήστε το κουμπί «Επανεκκίνηση», όπου θα δημιουργήσει ένα νέο σύνολο ράβδων και θα αρχικοποιήσει τη ρουτίνα ταξινόμησης.

Αν εκτελέσουμε την εφαρμογή κάνοντας κλικ το κουμπί «Επόμενο βήμα» μπορούμε να δούμε πραγματικά πως λειτουργεί ο αλγόριθμος InsertionSort. Θα δείτε ένα βέλος να μετακινείται αριστερά μέχρι να βρεθεί η σωστή θέση για να τοποθετηθεί η ράβδος που βρίσκεται προσωρινά στη μνήμη, μετακινώντας μία θέση δεξιά κάθε ράβδο που είναι μεγαλύτερη από αυτή. Το βέλος ονομάζετε «item--» και μετακινείτε πάντα προς τα αριστερά. Κάθε φορά που κάνετε κλικ στο κουμπί «Επόμενο βήμα» θα δείτε να τοποθετεί μία ράβδο από το δεξιό αταξινόμητο γράφημα στη προσωρινή μνήμη και να τη συγκρίνει με το αριστερό ταξινομημένο γράφημα κάνοντας μία θέση δεξιά τις ράβδους μέχρις ότου βρεθεί η σωστή θέση. Τα μηνύματα στο δεξιό μέρος της εφαρμογής δείχνουν πόσες συγκρίσεις και αντιμεταθέσεις έχουν γίνει μέχρι τώρα.

3.2.5 Απόδοση της ταξινόμησης εισαγωγής

Στο πρώτο πέρασμα, συγκρίνει το πολύ ένα στοιχείο. Στο δεύτερο πέρασμα το πολύ δύο στοιχεία και ούτω καθ' εξής, μέχρι να φτάσει το μέγιστο N-1 συγκρίσεις, δηλαδή 1+2+3+...+ N-1 = N* (N-1)/2.

Επειδή όμως σε κάθε πέρασμα κατά μέσο όρο συγκρίνονται μόνο τα μισά στοιχεία, πριν βρεθεί το σημείο εισαγωγής, μπορούμε να διαιρέσουμε αυτόν τον αριθμό με το 2, που μας δίνει N * (N+1)/4.

Ο αριθμός των αντιγραφών είναι περίπου ίδιος με τον αριθμό των συγκρίσεων. Μια αντιγραφή όμως δεν είναι τόσο χρονοβόρα όσο μία αντιμετάθεση, όταν λοιπόν έχουμε τυχαία δεδομένα, αυτός ο αλγόριθμος εκτελείται σε διπλάσια ταχύτητα από την ταξινόμηση φυσαλίδας. Η ταξινόμηση εισαγωγής εκτελείται σε χρόνο O(N^2) για τυχαία δεδομένα.

Για δεδομένα που είναι ήδη ταξινομημένα ή περίπου ταξινομημένα, η ταξινόμηση εισαγωγής τα καταφέρνει πολύ καλύτερα. Όταν τα δεδομένα είναι σε σειρά, η συνθήκη στο βρόγχο while δεν ισχύει ποτέ, γίνεται λοιπόν μια απλή πρόταση στον εξωτερικό βρόγχο, η οποία εκτελείται N-1 φορές. Σε αυτή τη περίπτωση ο αλγόριθμος εκτελείται σε χρόνο O(N). Αν τα δεδομένα είναι σχεδόν ταξινομημένα, ταξινόμηση εισαγωγής εκτελείται σε χρόνο περίπου O(N), κάτι που την καθιστά έναν απλό και αποτελεσματικό τρόπο ταξινόμησης ενός αρχείου που είναι ελαφρώς αταξινόμητο.

3.3 Ταξινόμηση σωρού

Ο σωρός είναι μια δενδρική δομή και χρησιμοποιείται για την δημιουργία ουρών προτεραιότητας (priority queues). Η ρίζα του δέντρου περιλαμβάνει το μικρότερο-μεγαλύτερο στοιχείο του αναλόγως αν έχουμε σωρό ελαχίστων ή μεγίστων. Τα επόμενα δύο στοιχεία του δένδρου είναι τα παιδιά του. Γενικότερα αν ο πατέρας είναι στη θέση i τα παιδιά του θα είναι στην θέση 2*I (αριστερό παιδί) και 2*i+1 (δεξί παιδί) αντίστοιχα. Αν i η θέση ενός παιδιού i/2 είναι η θέση του πατέρα του. Κάθε σωρός με n στοιχεία έχει ύψος log2n. Ο σωρός, όπως και τα δέντρα γενικότερα, μπορεί να υλοποιηθεί με πίνακα, στον οποίο εισάγονται τα κλειδιά του σωρού από αριστερά προς τα δεξιά και από πάνω προς τα κάτω.

Υπάρχουν δύο είδη σωρών :

- Οι σωροί μεγίστου (maxheap)
- Οι σωροί ελαχίστων(minheap)

Οι βασικές λειτουργίες ενός σωρού είναι:

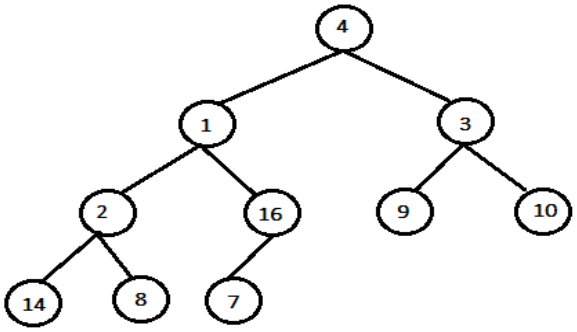
- Η εισαγωγή(Insert) ενός στοιχείου στον σωρό
- Η διαγραφή(Delete) ενός στοιχείου από τον σωρό

3.3.1 Περιγραφή

Ο Heap Sort αρχικά λειτουργεί μετατρέποντας τον αρχικό πίνακα σε ένα σωρό. Ο αλγόριθμος HeapSort χρησιμοποιεί μια διαδικασία αποκαλούμενη Heapify για να ολοκληρώσει την εργασία του. Ο αλγόριθμος Heapify, λαμβάνει ένα δυαδικό δέντρο ως είσοδο και το μετατρέπει σε ένα σωρό. Η ρίζα συγκρίνεται έπειτα με δύο άμεσα παιδιά του, και το μεγαλύτερο παιδί ανταλλάσσεται με αυτήν. Αυτό μπορεί να οδηγήσει σε ένα από το αριστερό ή το δεξιό υποδένδρο. Συνεπώς, ο αλγόριθμος Heapify εφαρμόζεται κατ' επανάληψη στο κατάλληλο υποδένδρο του οποίου η τιμή ανταλλάχθηκε με τη ρίζα και η διαδικασία συνεχίζεται μέχρις ότου είτε φτάσουμε σε ένα στοιχείο του φύλου είτε καθοριστεί ότι η προτεραιότητα του σωρού ικανοποιείται στο συγκεκριμένο υποδένδρο.
Ολόκληρη η μέθοδος HeapSort αποτελείται από δύο σημαντικά βήματα:

1. Κατασκευή του αρχικού σωρού χρησιμοποιώντας και ρυθμίζοντας (N/2) φορές όλα τα στοιχεία που δεν είναι φύλλα.
2. Ταξινόμηση ανταλλάσοντας και ρυθμίζοντας τον σωρό (N-1) φορές.

Στην εικόνα παρακάτω φαίνεται ένας σωρός και η αντίστοιχη υλοποίησή του σε πίνακα :



4	1	3	2	16	9	10	14	8	7

3.3.2 Δομή σωρού μεγίστων (max heap)

Σωρός μεγίστων είναι ένα δένδρο, το οποίο ικανοποιεί δύο συνθήκες:

- ✓ Η τιμή του κλειδιού κάθε κόμβου είναι μεγαλύτερη ή ίση από τις τιμές των κλειδιών των παιδιών του.
- ✓ Είναι ένα συμπληρωμένο δέντρο, που σημαίνει ότι μπορεί να προκύψει από ένα πλήρες δέντρο αφαιρώντας έναν αριθμό στοιχείων από το τέλος.

Εισαγωγή στοιχείου(Insert)

Όταν εισάγουμε ένα καινούργιο στοιχείο στο σωρό αυτό τοποθετείται στην επόμενη διαθέσιμη θέση στο τελευταίο επίπεδο. Αν η τιμή του σωρού δεν παραβιάζει την ιδιότητα του σωρού (δηλαδή η τιμή του είναι μικρότερη η ίση από αυτή του γονέα του) τότε το στοιχείο μένει στη θέση του. Σε διαφορετική περίπτωση παιδί και γονέας ανταλλάσσουν θέσεις. Αυτή η διαδικασία συνεχίζεται μέχρι το καινούργιο στοιχείο να βρει μια θέση τέτοια ώστε να ικανοποιείται η ιδιότητα του σωρού. Έτσι κάθε στοιχείο που εισάγεται στο σωρό θα πρέπει να διασχίσει ένα μονοπάτι από το φύλλο στο οποίο βρισκόταν αρχικά μέχρι κάποιον κατάλληλο εσωτερικό κόμβο. Στη χειρότερη περίπτωση το μονοπάτι θα είναι ως τη ρίζα του δέντρου. Ο χρόνος εισαγωγής ενός στοιχείου στο δέντρο είναι O(logn) όπου n το πλήθος των στοιχείων του σωρού. Στη χειρότερη περίπτωση θα χρειαστούν τόσα βήματα όσο και το ύψος του δέντρου.

Διαγραφή μεγίστου στοιχείου (deletemin)

Το μεγαλύτερο στοιχείο βρίσκεται πάντα στη ρίζα αλλά όταν το απομακρύνουμε δημιουργείται ένα κενό. Το κενό πρέπει να καλυφτεί με τέτοιο τρόπο ώστε ο σωρός να τηρεί την ιδιότητα του σωρού. Μετά την απομάκρυνση του μεγαλύτερου στοιχείου τοποθετείται στη ρίζα το τελευταίο στοιχείο του σωρού και το τρέχον μέγεθος μειώνεται κατά ένα. Αν το στοιχείο που περιέχει τώρα η ρίζα είναι μεγαλύτερο ή ίσο με τα παιδιά του μένει ως έχει. Σε διαφορετική περίπτωση το μεγαλύτερο παιδί παίρνει τη θέση του γονέα και αυτό συνεχίζεται μέχρι το στοιχείο που ήταν στη ρίζα να βρεθεί σε μία θέση στο σωρό που οι τιμές των παιδιών του να μην είναι μεγαλύτερες από τη δικιά του. Και σε αυτήν τη περίπτωση ο χρόνος διαγραφής είναι O(logn) όπου n το πλήθος των στοιχείων του σωρού.

3.3.3 Δομή σωρού ελαχίστων (min heap)

Σωρός ελαχίστων είναι ένα δένδρο, το οποίο ικανοποιεί δύο συνθήκες:

- ✓ Η τιμή του κλειδιού κάθε κόμβου είναι μικρότερη ή ίση από τις τιμές των κλειδιών των παιδιών του.
- ✓ Είναι ένα συμπληρωμένο δέντρο, που σημαίνει ότι μπορεί να προκύψει από ένα πλήρες δέντρο αφαιρώντας έναν αριθμό στοιχείων από το τέλος.

Εισαγωγή στοιχείου(Insert)

Για να εισάγουμε ένα στοιχείο στο σωρό, αρχικά το προσθέτουμε στο τέλος του. Αν το στοιχείο που εισάγουμε είναι μικρότερο από τον γονιό του, τότε αλλάζουν θέση. Επαναλαμβάνουμε αυτή τη διαδικασία μέχρι να μην παραβιάζεται ο κανόνας αυτός.

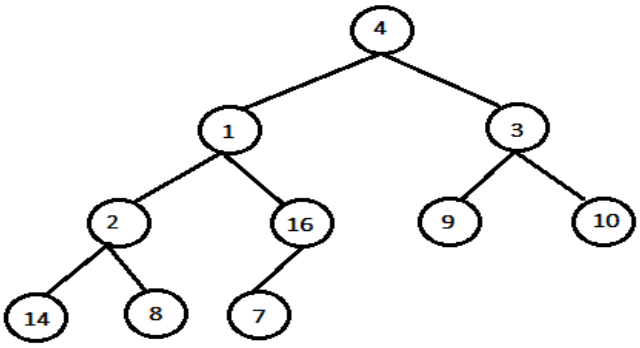
Διαγραφή ελάχιστου στοιχείου (deletemin)

Για να διαγράψουμε το ελάχιστο στοιχείο του σωρού ακολουθούμε την εξής μέθοδο. Πρώτα διαγράφουμε το στοιχείο της κορυφής και βάζουμε στη θέση του το τελευταίο στοιχείο του σωρού. Στη συνέχεια κάνουμε τις κατάλληλες αλλαγές, εφόσον αυτές απαιτούνται για να μην παραβιάζεται ο 1ος κανόνας του σωρού ελαχίστων.

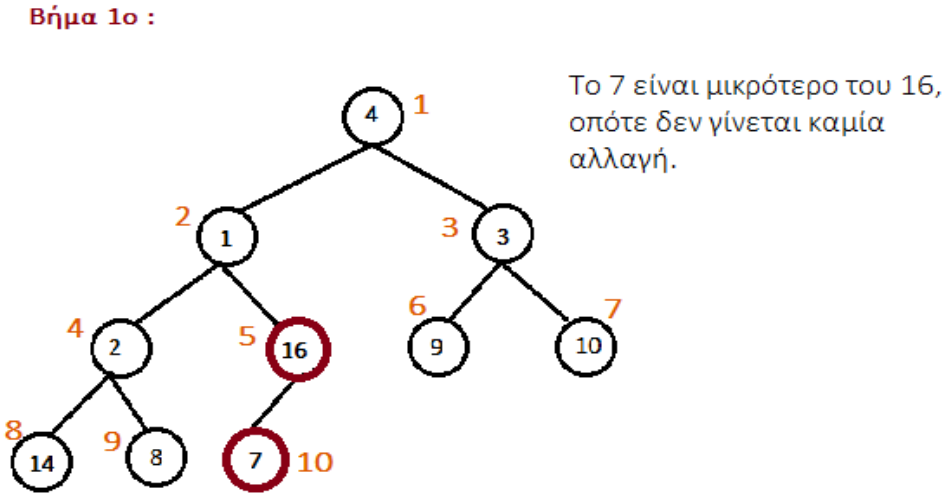
3.3.4 Παράδειγμα δημιουργίας Σωρού Μεγίστων (maxHeap)

Παρακάτω δίνεται ένα παράδειγμα που δείχνει διαγραμματικά βήμα-βήμα τη δημιουργία του σωρού μεγίστων από τα στοιχεία εισόδου :

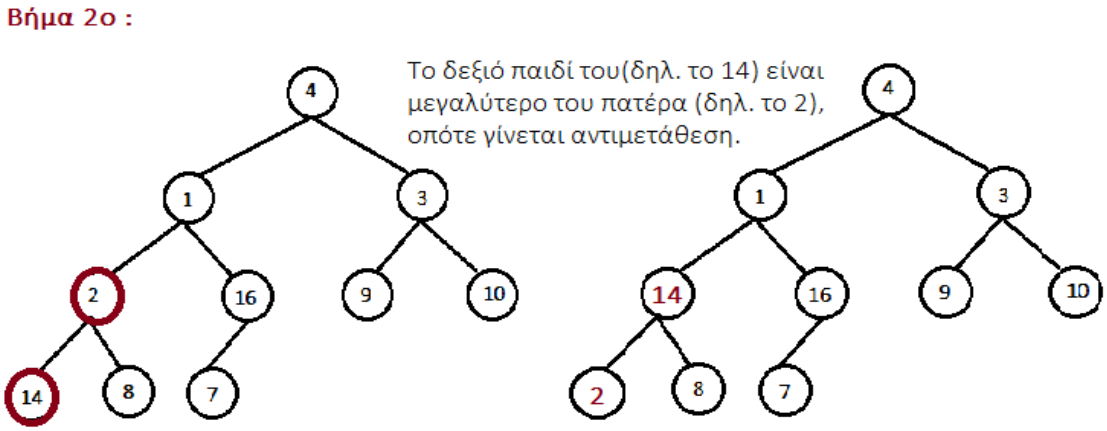
4	1	3	2	16	9	10	14	8	7



Ο πρώτος κόμβος που συγκρίνουμε στη 5η θέση (το μέγεθος του πίνακα διά δύο, 10/2), δηλαδή το 16, με το δεξιό παιδί του που βρίσκεται στη 10η θέση (2*i, όπου i είναι το 5), δηλαδή το 7.

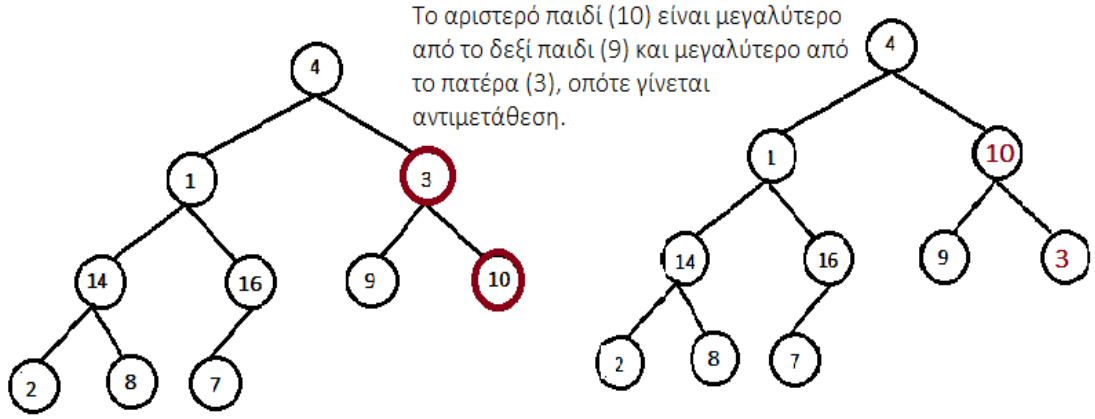


Στη συνέχεια συγκρίνουμε το κόμβο στη 4η θέση, δηλαδή το 2, με το δεξιό παιδί που βρίσκεται στην 8η θέση(2*4), δηλ. το 14, και το αριστερό παιδί που βρίσκεται στην 9η θέση(2*4+1), δηλ. το 8. Το δεξί παιδί είναι μεγαλύτερο από το αριστερό παιδί και μεγαλύτερο από τον πατέρα του, οπότε γίνεται αντιμετάθεση.



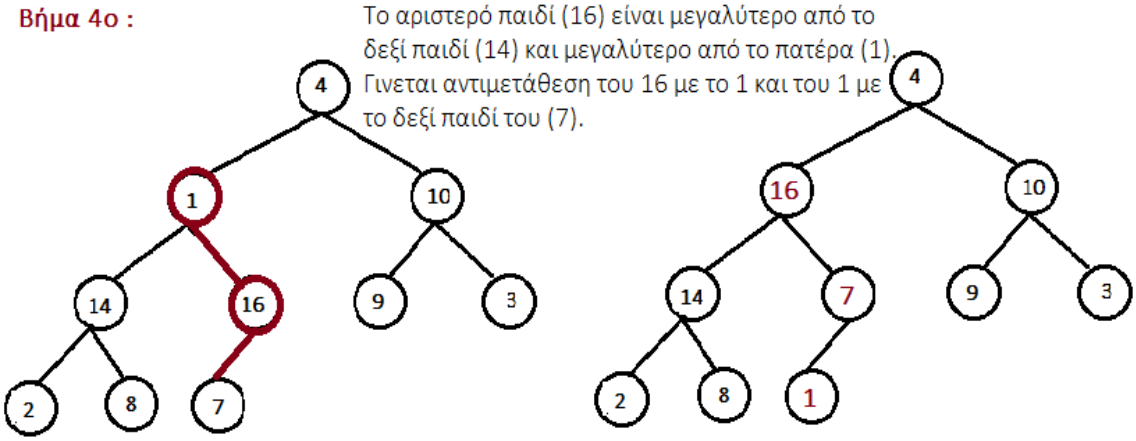
Στο 3ο βήμα συγκρίνουμε το κόμβο στη 3η θέση, δηλαδή το 3, με το δεξιό παιδί που βρίσκεται στην 6η θέση(2*3), δηλ. το 9, και το αριστερό παιδί που βρίσκεται στην 7η θέση(2*3+1), δηλ. το 10. Το αριστερό παιδί είναι μεγαλύτερο από το δεξί παιδί και μεγαλύτερο από τον πατέρα του, οπότε γίνεται αντιμετάθεση.

Βήμα 3ο :



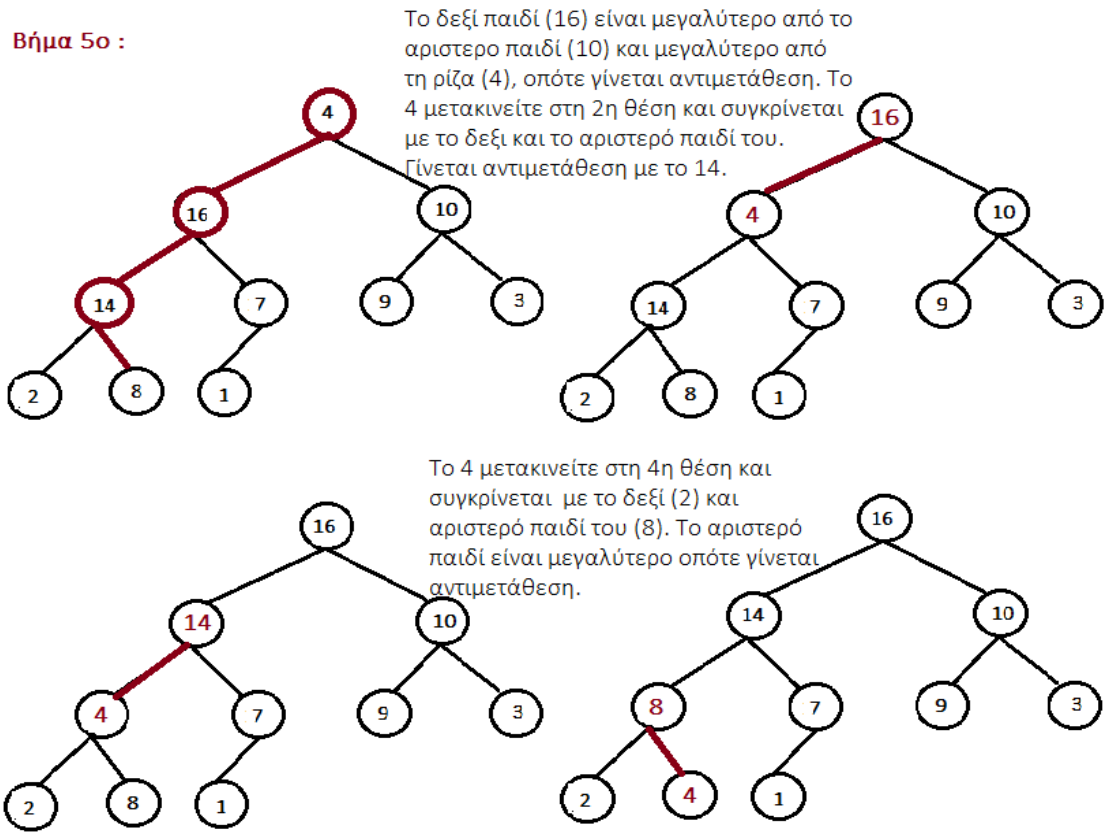
Στο 4^{ο} βήμα έχουμε 2 αντιμεταθέσεις. Πρώτα συγκρίνουμε το κόμβο στη 2^{η} θέση, δηλαδή το 1, με το δεξιό παιδί που βρίσκεται στην 4^{η} θέση(2*2), δηλ. το 14, και το αριστερό παιδί που βρίσκεται στην 5^{η} θέση(2*2+1), δηλ. το 16. Το αριστερό παιδί είναι μεγαλύτερο από το δεξί παιδί και μεγαλύτερο από τον πατέρα του, οπότε γίνεται αντιμετάθεση. Το 1 θα μετακινηθεί στη 5^{η} θέση και θα συγκριθεί με δεξί παιδί του, δηλ. το 7, όπου είναι μεγαλύτερο και γίνεται η 2^{η} αντιμετάθεση.

Βήμα 4ο :



Στο 5^{ο} βήμα γίνονται τρεις αντιμεταθέσεις. Πρώτα συγκρίνουμε τη ρίζα, δηλαδή το 4, με το δεξιό παιδί που βρίσκεται στην 2^{η} θέση(2*1), δηλ. το 16, και το αριστερό παιδί που βρίσκεται στην 3^{η} θέση(2*1+1), δηλ. το 10. Το δεξί παιδί είναι μεγαλύτερο από το αριστερό παιδί και μεγαλύτερο από τη ρίζα, οπότε γίνεται αντιμετάθεση. Το 4 θα μετακινηθεί στη 2^{η} θέση και θα συγκριθεί με δεξί παιδί του, δηλ. το 14 και το αριστερό παιδί, δηλ. το 7 . Γίνεται αντιμετάθεση με το δεξί παιδί, αφού είναι μεγαλύτερο και το 4

μετακινείτε στη 4η θέση. Η τελευταία αντιμετάθεση είναι του 4 με το αριστερό παιδί του, το 8.



Ο πίνακας θα έχει τα εξής δεδομένα μετά τη δημιουργία της σωρού :

16	14	10	8	7	9	3	2	4	1

3.3.5 Υλοποίηση σε Java

Στο πρόγραμμα HeapSort.java θα δείτε ότι στη ρουτίνα main() περιέχει έναν πίνακα «a[]», ο οποίος δημιουργεί δέκα τυχαίες μεταβλητές τύπου «int» (Integer) από 0 έως 50 .

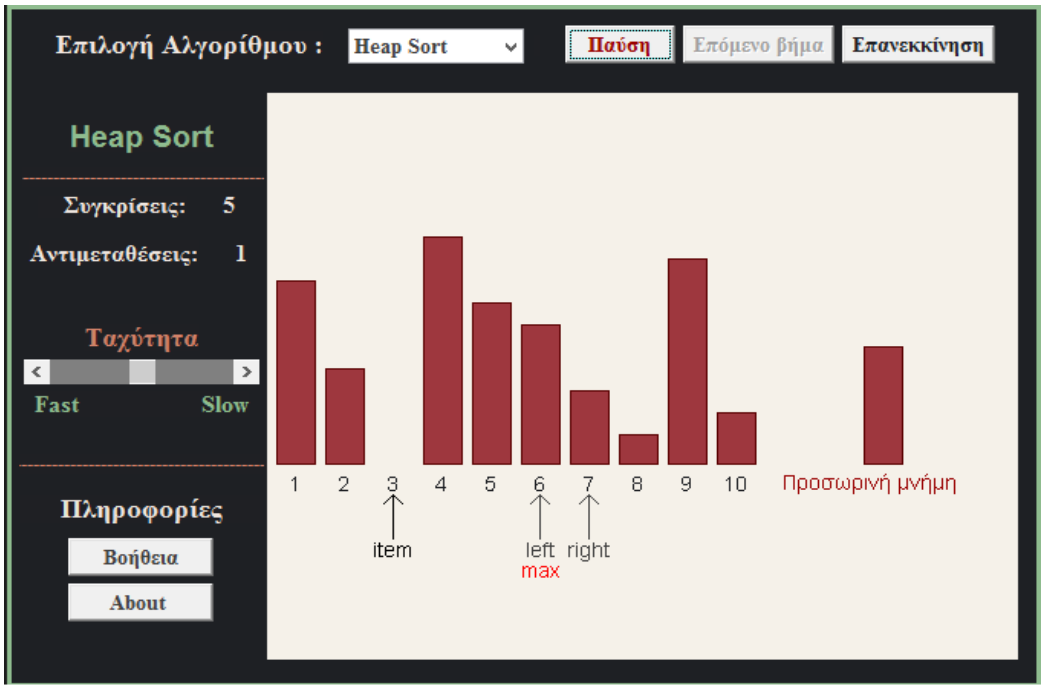
```
// Παράδειγμα της Heap Sort σε Java
// Αυτό το παράδειγμα δείχνει πως ταξινομούμε ένα πίνακα με ακεραίους αριθμούς
// χρησιμοποιώντας τον αλγόριθμο maxHeap.
//-----------------------------------------------------------------------------------------------------
import java.util.Random;
public class HeapSort {
    public  static void heapSort(int array[]) {
     heapSort(array, array.length - 1);
```

```
}
//-----------------------------------------------------------------------------------------------------------
private static void heapSort(int array[], int heapsize) {

   for (int i = heapsize / 2; i >= 0; i--) {            // Δημιουργία Σωρού Μεγίστων
         maxheapify(array, i, heapsize);    // καλεί τη μέθοδο maxheapify για να ταξινομήσει τη
σωρό
   }

   for (int i = heapsize; i >= 0; i--) {              // Διαγραφή μεγίστου στοιχείου
         swap(array, 0, heapsize);          // κάνε αντιμετάθεση της ρίζας με το τελευταίο κόμβο
         heapsize--;                                  // μείωσε κατά 1 το μέγεθος της σωρού
         maxheapify(array, 0, heapsize);       // καλεί τη μέθοδο maxheapify για να ταξινομήσει τη
σωρό
    }
}
//-----------------------------------------------------------------------------------------------------------
private static void maxheapify(int a[], int i, int heapsize) {
      int largest;
      int left = 2 * i;                          // δεξιό παιδί
      int right = 2 * i + 1;                     // αριστερό παιδί

      if (left <= heapsize && a[left] > a[i]) {     // αν το δεξιό παιδί είναι μεγαλύτερο του πατέρα
         largest = left;                         // το μεγαλύτερο στοιχείο είναι το δεξιό παιδί
      }
      else {
         largest = i;                            // αλλιώς το μεγαλύτερο στοιχείο είναι ο πατέρας
      }

      if (right <= heapsize && a[right] > a[largest]) {    // αν το αριστερό παιδί είναι μεγαλύτερο
του largest
         largest = right;
      }

      if (largest != i) {                        // αν το μεγαλύτερο στοιχείο δεν είναι ο πατέρας
         swap(a, i, largest);                          // κάνε αντιμετάθεση
         maxheapify(a, largest, heapsize);       /* αναδρομή */
      }
      return;        /* επιστροφή όταν largest == i */
```

```
    }
//-----------------------------------------------------------------------------------------------------------
public static void main(String[] args) {

// Δημιουργεί ένα πίνακα με 10 τυχαίους αριθμούς
        int a[] = new int[10];
        for(int i = 0;i<a.length;i++){
            a[i] = new Random().nextInt(50);
// Εκτυπώνει τον πίνακα πριν ταξινομηθεί
            System.out.print(a[i]+ " ");
        }
        System.out.println( " ");

// Καλεί την μέθοδο με τα στοιχεία του πίνακα
        heapSort(a);

// Εκτυπώνει τα στοιχεία ταξινομημένα
        for(int i = 0;i<a.length;i++){
            System.out.print(a[i]+ " ");
        }
    }
//-----------------------------------------------------------------------------------------------------------
public static void swap(int []a,int i,int j){
// Κάνει αντιμετάθεση των στοιχείων
        int temp = a[i];
        a[i] = a[j];
        a[j] = temp;
    }
}
```

3.3.6 Η εφαρμογή HeapSort

Στη εικόνα βλέπετε την εφαρμογή HeapSort. Αν την εκτελέσετε θα δείτε να μοιάζει με γράφημα ράβδου, με τα ύψη των ράβδων τυχαία διευθετημένα.



Αυτή η εφαρμογή περιέχει ένα γράφημα δύο ταχυτήτων. Μπορείτε να την αφήσετε να εκτελείται μόνη της (κάνοντας κλικ στο κουμπί «Εκκίνηση») ή μπορείτε να προχωράτε βήμα – βήμα (κάνοντας κλικ στο κουμπί «Επόμενο βήμα»). Για να κάνετε άλλη μία ταξινόμηση, πατήστε το κουμπί «Επανεκκίνηση», όπου θα δημιουργήσει ένα νέο σύνολο ράβδων και θα αρχικοποιήσει τη ρουτίνα ταξινόμησης.

Αν εκτελέσουμε την εφαρμογή κάνοντας κλικ το κουμπί «Επόμενο βήμα» μπορούμε να δούμε πραγματικά πως λειτουργεί ο αλγόριθμος HeapSort. Θα δείτε τρία βέλη που δείχνουν σε διαφορετικούς ράβδους. Το βέλος «item» δείχνει το πατέρα και τα δύο βέλη «left» και «right» που βρίσκονται το ένα δίπλα στο άλλο δείχνουν το αριστερό και δεξιό παιδί του «item». Η ράβδος που βρίσκεται στη θέση (i) που δείχνει το βέλος «item» κάθε φορά συγκρίνεται με το δεξί (2*i) και το αριστερό παιδί του (2*i+1). Στη περίπτωση που κάποιο από τα παιδία του είναι μεγαλύτερο, γίνεται αντιμετάθεση με το πατέρα. Κάθε φορά που χρειάζεται να αντιμετατεθούν δυο ράβδοι, η μία ράβδος τοποθετείται στη «Προσωρινή μνήμη» για να μπορέσει να γίνει η αλλαγή. Τα μηνύματα στο δεξιό μέρος της εφαρμογής δείχνουν πόσες συγκρίσεις και αντιμεταθέσεις έχουν γίνει μέχρι τώρα.

3.3.7 Απόδοση της ταξινόμησης HeapSort

Η HeapSort εκτελείται σε χρόνο O(N*logN), όπου N είναι ο αριθμός των κόμβων. Το κόστος αρχικής κατασκευής της σωρού υπολογίζεται σε O(N), ενώ το κόστος κάθε

εξαγωγής μεγίστου είναι O(logN) και το πλήθος εξαγωγών είναι N-1. Άρα το συνολικό κόστος θα είναι :

$$O(N) + (N-1)*O(logN) = O(N) + O(N*logN) = O(N*logN)$$

Η πολυπλοκότητα του heapsort είναι η ίδια σε κάθε περίπτωση.

3.4 Σύγκριση των αλγορίθμων ταξινόμησης

Οι ταξινομήσεις φυσαλίδα, επιλογής και εισαγωγής έχουν τον ίδιο μέσο χρόνο εκτέλεσης. Παρ' όλα αυτά η ταξινόμηση φυσαλίδας είναι πιο αργή από τις υπόλοιπες, ειδικά όταν το μέγεθος των δεδομένων που χρειάζονται ταξινόμηση είναι μεγάλο. Η ταξινόμηση εισαγωγής έχει καλή απόδοση όταν τα δεδομένα είναι σχεδόν ταξινομημένα και λειτουργεί σε χρόνο O(N) αν τα περισσότερα στοιχεία βρίσκονται στη σωστή θέση. Μία σύνθετη αλλά γρήγορη ταξινόμηση είναι η Heapsort, όπου είναι η καλύτερη ταξινόμηση όταν χρησιμοποιείτε σε δεδομένα που βρίσκονται σε τυχαία σειρά.

Στον παρακάτω πίνακα συνοψίζονται οι χρόνοι εκτέλεσης για τους διάφορους αλγόριθμους ταξινόμησης. Η στήλη «Σύγκριση» επιχειρεί να εκτιμήσει τις διαφορές στη μικρότερη ταχύτητα μεταξύ των αλγορίθμων που έχουν ίδιο μέσο χρόνο.

Ταξινόμηση	**Μέσος όρος**	**Χειρότερος χρόνος**	**Σύγκριση**	**Επιπλέον μνήμη**
Φυσαλίδας	O(N^2)	O(N^2)	Χαμηλή	Όχι
Επιλογής	O(N^2)	O(N^2)	Μέτρια	Όχι
Εισαγωγής	O(N^2)	O(N^2)	Καλή	Όχι
Shellsort	O(N^2/3)	O(N^2/3)	-	Όχι
Quicksort	O(N*logN)	O(N^2)	Καλή	Όχι
Ταξινόμηση σύζευξης	O(N*logN)	O(N*logN)	Μέτρια	Ναι
Heapsort	O(N*logN)	O(N*logN)	Μέτρια	Όχι

ΚΕΦΑΛΑΙΟ 4: Η εφαρμογή «Οπτικοποίηση Αλγορίθμων» (Visualization and Algorithms Animation)

Η ραγδαία εξάπλωση του Internet και του World-Wide Web δημιούργησαν την ανάγκη νέων τρόπων ανάπτυξης και διανομής του λογισμικού. Οι απαιτήσεις αυτές οδήγησαν στην δημιουργία της γλώσσας προγραμματισμού Java, από την εταιρία Sun microsystems TM. Η Java σχεδιάστηκε με σκοπό την ανάπτυξη εφαρμογών που θα τρέχουν σε ετερογενή δικτυακά περιβάλλοντα.

Η Java έχει τα ακόλουθα χαρακτηριστικά :

- *Αντικειμενοστραφής*.
- *Δημιουργία ανεξάρτητων εφαρμογών και applets*. (applet = προγράμματα που περιλαμβάνονται σε HTML σελίδες και εκτελούνται από τον Web Browser).
- *Είναι Interpreted γλώσσα*. Αυτό σημαίνει ότι ο java compiler δεν παράγει εκτελέσιμο κώδικα αλλά μια μορφή ψευδοκώδικα (bytecode) το οποίο από μόνο του δεν τρέχει σε καμία μηχανή. Προκειμένου λοιπόν να εκτελεστεί απαιτείται η χρήση ενός interpreter (=διερμηνέα) για να μετατρέψει το bytecode σε πραγματικό εκτελέσιμο κώδικα. Αυτό το χαρακτηριστικό δίνει τη δυνατότητα στα java bytecodes να μπορούν να τρέξουν σε οποιοδήποτε μηχάνημα, κάτω από οποιοδήποτε λειτουργικό, αρκεί να έχει εγκατασταθεί ένας java interpreter. Επίσης ένα άλλο χαρακτηριστικό του java bytecode είναι το μικρό του μέγεθος, (μόλις λίγα Kilobytes). Αυτό το κάνει ιδανικό για μετάδοση μέσω του δικτύου.
- *Κατανεμημένη (distributed).* Δηλαδή ένα πρόγραμμα σε Java είναι δυνατό να το φέρουμε από το δίκτυο και να το τρέξουμε. Επίσης είναι δυνατό διαφορετικά κομμάτια του προγράμματος να έρθουν από διαφορετικά sites.
- *Ασφαλής (secure).* Στο δίκτυο όμως ελλοχεύουν πολλοί κίνδυνοι για τον χρήστη - παραλήπτη μιας δικτυακής εφαρμογής, γι' αυτό η Java έχει σχεδιαστεί έτσι ώστε να ελαχιστοποιείται η πιθανότητα προσβολής του συστήματος του χρήστη από κάποιο applet γραμμένο για τέτοιο σκοπό.
- *Είναι multithreaded*. Η Java υποστηρίζει εγγενώς την χρήση πολλών threads. Προκειμένου να το επιτύχει αυτό σε συστήματα με έναν επεξεργαστή, το Java runtime system (interpreter) υλοποιεί ένα δικό χρονοδρομολογητή (scheduler), ενώ σε συστήματα που υποστηρίζουν πολυεπεξεργασία η δημιουργία των threads

ανατίθεται στο λειτουργικό σύστημα. Φυσικά όλα αυτά είναι αόρατα τόσο στον προγραμματιστή όσο και στον χρήστη.

- *Υποστηρίζει multimedia εφαρμογές*. Με αυτό εννοούμε ότι η Java παρέχει ευκολίες στη δημιουργία multimedia εφαρμογών. Αυτό επιτυγχάνεται τόσο με την ευελιξίας της σαν γλώσσα όσο και με τις πλούσιες και συνεχώς εμπλουτιζόμενες βιβλιοθήκες της.

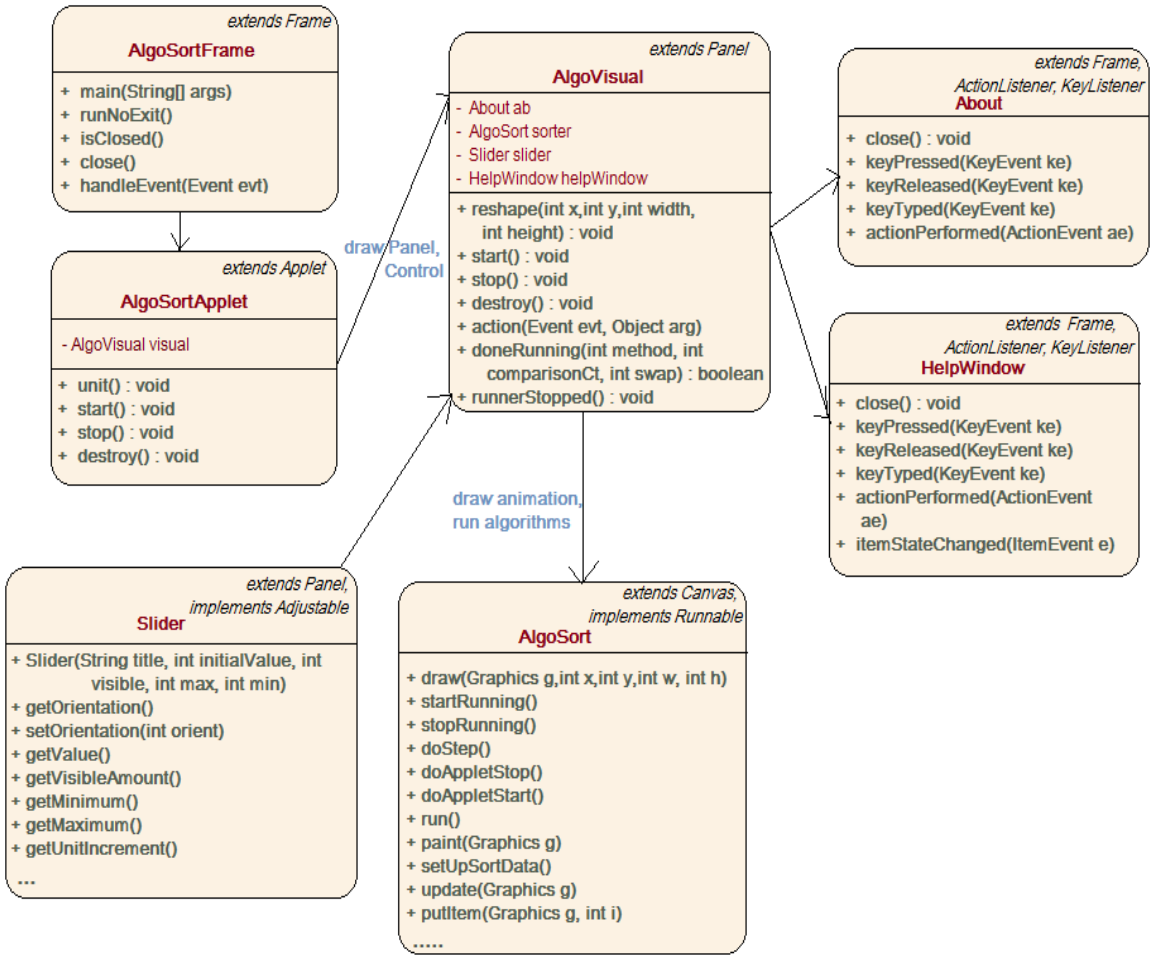
Αυτό το λογισμικό αναπτύχθηκε με στόχο τη χρήση του ως παιδαγωγικό εργαλείο για την εκμάθηση των αλγορίθμων : Bubble Sort, Insertion Sort και Heap Sort. Η εφαρμογή αυτή είναι ενα εργαλείο απεικόνισης που μπορεί να χρησιμοποιηθεί για να συμπληρώσει την διδασκαλία των συγκεκριμένων αλγορίθμων, βοηθώντας έτσι τους σπουδαστές να κατανοήσουν καλύτερα την λειτουργίας τους. Το λογισμικό γράφτηκε στην αντικειμενοστραφή γλώσσα προγραμματισμού Java και αναπτύχθηκε στην πλατφόρμα NetBeans IDE 7.3 Beta.

4.1 Γενική δομή των κλάσεων της εφαρμογής

Η εφαρμογή « Οπτικοποίηση Αλγορίθμων» αποτελείται από επτά κλάσεις, οι οποίες είναι :

1. AlgoSortFrame.java : Περιέχει την main() για να τρέξει το πρόγραμμα.
2. AlgoSortApplet.java : Καλεί την κλάση AlgoVisual.java για να σχεδιάσει το Panel της εφαρμογής.
3. AlgoVisual.java : Στην κλάση αυτή σχεδιάζονται τα κουμπιά ελέγχου της εφαρμογής και καλεί την κλάση AlgoSort.java.
4. AlgoSort.java : Σχεδιάζει τη γραφική απεικόνιση της εφαρμογής και εκτελεί το animation ανάλογα τον αλγόριθμο επιλογής.
5. Slider.java : Σχεδιάζει την μπάρα κύλισης στην κλάση AlgoVisual.java και περιέχει μεθόδους για τον έλεγχο ταχύτητας του animation.
6. About.java : Παρέχει πληροφορίες σχετικά με εμάς.
7. HelpWindow.java : Παρέχει πληροφορίες σχετικά με τους αλγόριθμους : BubbleSort, InsertionSort και HeapSort.

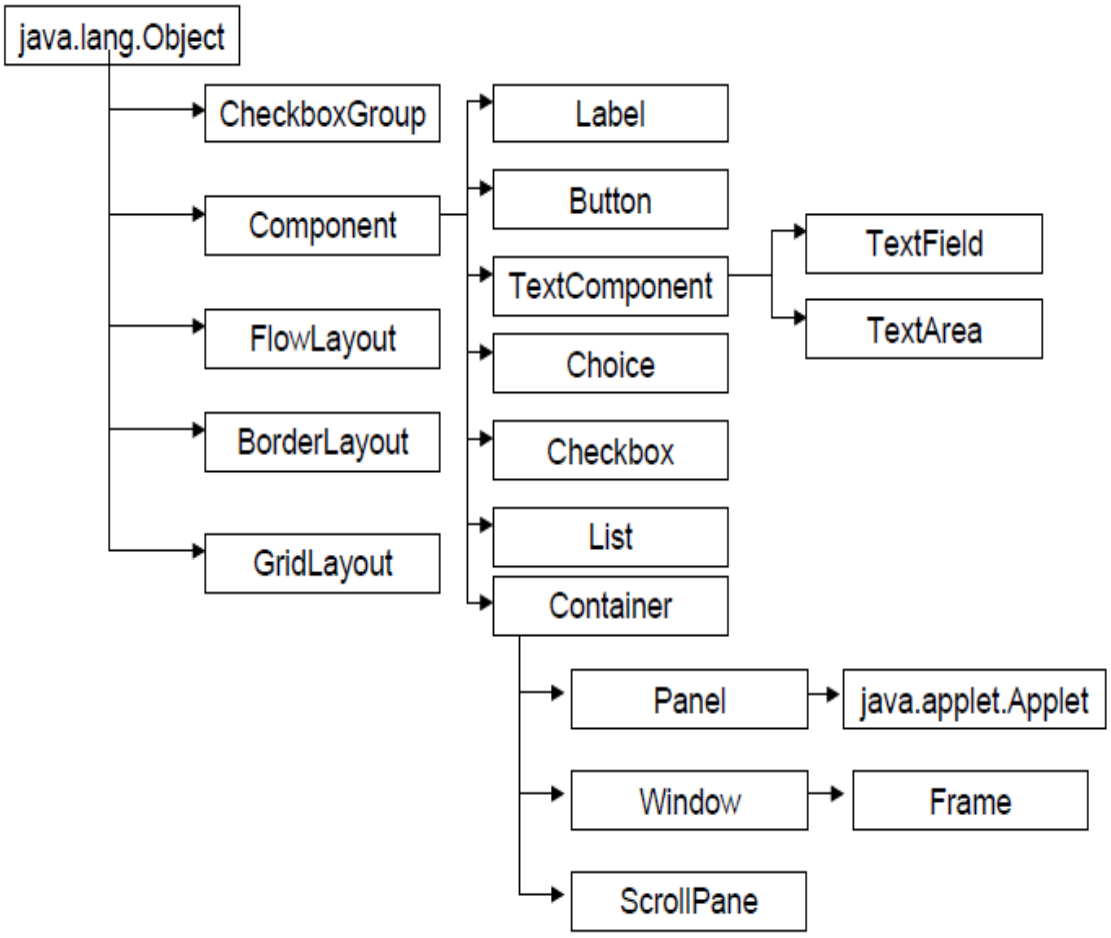
Στην παρακάτω εικόνα βλέπουμε το διάγραμμα πως συνδέονται οι κλάσεις της εφαρμογής «Οπτικοποίηση Αλγορίθμων».



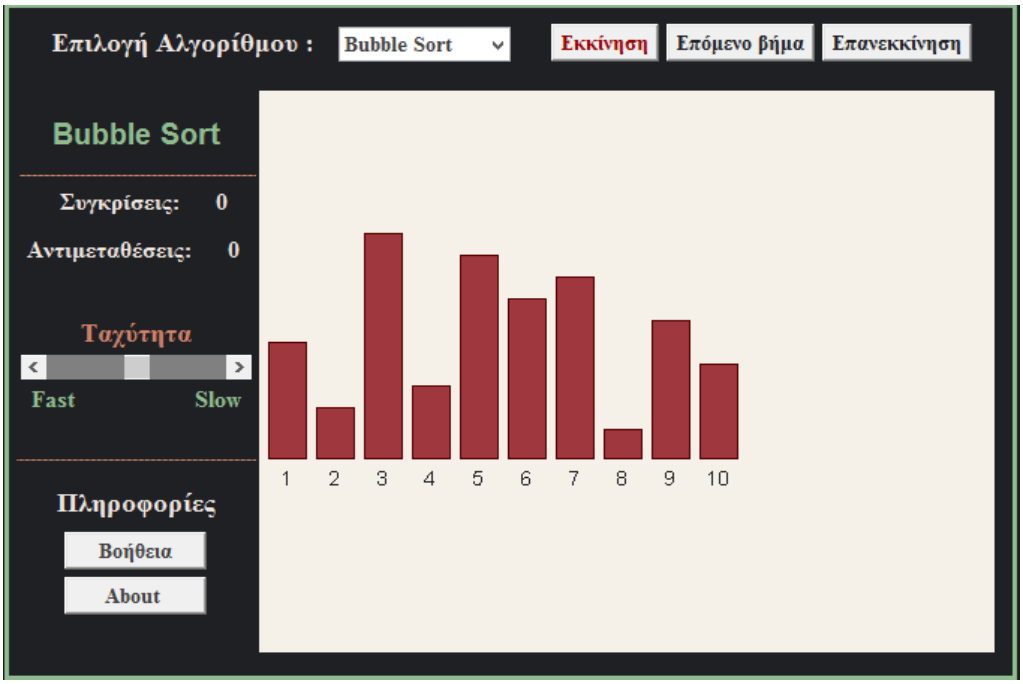
4.2 Το γραφικό περιβάλλον διεπαφής χρήστη (Graphical User Interface, GUI)

Ένα Graphical User Interface-GUI (Γραφική Διεπαφή με το Χρήστη) είναι το μέρος του προγράμματος, που φροντίζει για τον τρόπο εμφάνισης και χειρισμού του προγράμματος από τον χρήστη. Ένα GUI αποτελείται από GUI-components. Οι κλάσεις που χρησιμοποιούνται για την κατασκευή αντικειμένων τύπου GUI-components ανήκουν στο java.awt (Abstract Windowing Toolkit) package. Οι βασικότερες από αυτές είναι η κλάση Component και η κλάση Container.Κάθε κλάση που κληρονομεί την κλάση Component είναι και αυτή ένα Component. Επίσης κάθε κλάση που κληρονομεί την κλάση Container είναι και αυτή ένα Container.

Το επόμενο σχήμα εμφανίζει την ιεραρχία των GUI-components κλάσεων μαζί με κάποιες επιπλέον συμπληρωματικές κλάσεις που διαχειρίζονται ένα GUI.



Κατά το σχεδιασμό του γραφικού περιβάλλοντος (GUI) της εφαρμογής, δόθηκε έμφαση στο να διατηρηθεί απλό, ώστε ο χρήστης να μπορέσει να χρησιμοποιήσει την εφαρμογή χωρίς δυσκολία. Τα κουμπιά ελέγχου βρίσκονται στο πάνω μέρος της οθόνης, δίνοντας τη μορφή σαν μία γραμμή εργαλείων, αφήνοντας το υπόλοιπο παράθυρο για την οπτικοποίηση των αλγορίθμων, τονίζοντας έτσι την διαδραστική οπτικοποιημένη παρουσίαση των αλγορίθμων.



Αρχικά, στη γραμμή εργαλείων υπάρχει ένα πλαίσιο συνδυασμών (combo box) που επιτρέπει στον χρήστη να επιλέξει τον αλγόριθμο που επιθυμεί να εκτελέσει. Σε κάθε αλγόριθμο εμφανίζεται ένα γράφημα, όπου περιέχει δέκα ράβδους διαφορετικού μεγέθους. Οι κινούμενες εικόνες μπορούν να προβληθούν σταδιακά, επιλέγοντας το κουμπί «Επόμενο βήμα», ή συνεχώς, επιλέγοντας το κουμπί «Εκκίνηση». Επιπλέον η μπάρα κύλισης (scroll bar) επιτρέπει στον χρήστη να καθορίσει την ταχύτητα της κίνησης. Επίσης ο χρήστης μπορεί να διακόψει ένα συγκεκριμένο αλγόριθμο από την εκτέλεσή του σε οποιαδήποτε δεδομένη χρονική στιγμή, κάνοντας κλικ στο κουμπί «Παύση», στη περίπτωση που ο αλγόριθμος εκτελείτε για μεγάλο χρονικό διάστημα και ο χρήστης έχει ήδη κατανοήσει την λειτουργία του. Επίσης, ο κάθε αλγόριθμος όταν εκτελείται μετριούνται οι συγκρίσεις που γίνονται μεταξύ των ράβδων και οι αντιμεταθέσεις τους, δείχνοντας έτσι στον χρήστη να καταλάβει αν ο χρόνος εκτέλεσης ενός αλγορίθμου είναι μικρός ή μεγάλος. Κάθε φορά που γίνετε μία αντιμετάθεση δύο ράβδων, η μία ράβδος μετακινείτε στη «Προσωρινή μνήμη» μέχρι η άλλη ράβδος τοποθετηθεί στη σωστή θέση. Ακόμη, στο αριστερό μέρος της εφαρμογής υπάρχει το κουμπί «Βοήθεια», όπου υπάρχουν πληροφορίες σχετικά με τους αλγορίθμους και τη λειτουργία τους. Τέλος, το κουμπί «About» παρέχει πληροφορίες σχετικά με εμάς.

4.2.1 Η δομή του GUI της εφαρμογής

Για να απλοποιηθεί η ανάπτυξη ενός GUI στην Java, χωρίζεται συνήθως σε πολλά Panel components. Αξίζει να αναφερθεί οτι η κλάση Panel κληρονομεί την κλάση Container και η κλάση Applet την κλάση Panel. Έτσι τα Panels και τα Applets είναι Containers και μπορούν να έχουν Components, ακόμα και τύπου Panel.

Σε κάθε Panel τα Components που περιέχει είναι τοποθετημένα με ένα συγκεκριμένο πλάνο ή σχέδιο (layout). Η Java παρέχει διαχειριστές πλάνων (Layout Managers) για την διευθέτηση των Component αντικειμένων μέσα σε ένα Applet ή Panel.

FlowLayout	Είναι ο default Layout Manager για τα Applets και τα Panels. Τοποθετεί τα Component αντικείμενα από αριστερά προς τα δεξιά με την σειρά που προστίθενται.
BorderLayout	Τοποθετεί τα Component αντικείμενα σε πέντε περιοχές: Βόρεια, Νότια, Ανατολική, Δυτική και Κεντρική (North, South, East, West, Center).
GridLayout	Τοποθετεί τα Component αντικείμενα σε γραμμές και στήλες με καθορισμένη σειρά(πρώτα γεμίζει η πρώτη γραμμή μετά η δεύτερη γραμμή κλπ).
CardLayout	Τοποθετεί τα Component αντικείμενα σε στοίβα. Κάθε Container αντικείμενο στη στοίβα μπορεί να χρησιμοποιήσει οποιονδήποτε Layout Manager. Μόνο το Container αντικείμενο που βρίσκεται στην κορυφή της στοίβας είναι ορατό.
GridBagLayout	Είναι παρόμοιος με τον GridLayout Manager. Διαφέρει από αυτόν στο ότι κάθε Component αντικείμενο που του προστίθεται μπορεί να έχει οποιοδήποτε μέγεθος (δηλαδή να καλύπτει περισσότερες από μια γραμμές και στήλες). Επίσης η σειρά με την οποία προστίθενται τα Component αντικείμενα στον GridBagLayout Manager μπορεί να είναι οποιαδήποτε.

Κώδικας AlgoVisual.java

```
class AlgoVisual extends Panel {
.....
```

```
AlgoVisual() {
.....
        setLayout(null);
        titlePanel.setLayout(new GridLayout(2,1,5,5));
        titlePanel.setBackground(new Color(30,32,36));
        add(titlePanel);
        title = new Label("Bubble Sort", Label.CENTER);
        titlePanel.add(title);
        messagePanel.setLayout(new GridLayout(3,2,5,5));
        messagePanel.setLayout(new FlowLayout(1));
        add(messagePanel);
        comp1 = new Label("    Συγκρίσεις:");
        messagePanel.add(comp1);
        comparisons = new Label("0", Label.CENTER);
        messagePanel.add(comparisons);
        swap = new Label("Αντιμεταθέσεις:");
        messagePanel.add(swap);
        swaps = new Label("0", Label.CENTER);
        messagePanel.add(swaps);
        sliderPanel.setLayout(new GridLayout(2,1,5,5));
        sliderPanel.setLayout(new FlowLayout(1));
        add(sliderPanel);
        infoPanel.setLayout(new GridLayout(3,1,5,5));
        infoPanel.setLayout(new FlowLayout(1));
        help = new Button("    Βοήθεια    ");
        about = new Button("     About      ");
        add(infoPanel);
        label2 = new Label("     Πληροφορίες      ", Label.CENTER);
        infoPanel.add(label2);
        infoPanel.add(help);
        infoPanel.add(about);
        sortPanel.setLayout(new GridLayout(1,1,5,5));
        add(sortPanel);
        sortFinished = new Label("");
        sortPanel.add(sortFinished);
.......
```

```
}
.......
// Με τη μέθοδο reshape(int x, int y, int width, int height) τοποθετούμε τα Panel που
//δημιουργήσαμε  παραπάνω στη σωστή θέση μέσα στην εφαρμογή, δίνοντας το πλάτος,
// ύψος και τις διαστάσεις.
public void reshape(int x, int y, int width, int height) {
    super.reshape(x,y,width,height);
    if (width < 300)
        width = 300;
    if (height < 200)
        height = 200;
    controlPanel.reshape(width-120,0,120,0);
    infoPanel.reshape(0,289,160,160);
    messagePanel.reshape(0,110,160,90);
    titlePanel.reshape(0,70,160,52);
    sortPanel.reshape(270,390,240,40);
    sliderPanel.reshape(0,200,160,80);
  }
.......
}
```

4.3 Βασικές λειτουργίες της εφαρμογής

Στα Applets προγράμματα υπάρχουν λειτουργίες που αντιστοιχούν σε γεγονότα που συμβαίνουν κατά τη διάρκεια της ζωής του Applet. Σε κάθε λειτουργία αντιστοιχεί και κάποια μέθοδος η οποία καλείται από τον Browser όταν ένα γεγονός συμβεί. Οι μέθοδοι των λειτουργιών έτσι όπως ορίζονται στην Applet κλάση της Java δεν κάνουν τίποτε. Για να δώσουμε κάποια συμπεριφορά σε κάποιο γεγονός της Applet εφαρμογή μας πρέπει να ξαναορίσουμε την μέθοδο που αντιστοιχεί στο γεγονός (τεχνική method overriding) μέσα στην υποκλάση της Applet που δημιουργήσαμε. Οι τέσσερις πιο βασικές μέθοδοι μιας applet είναι:

1. public void init(){
   . . .
   }

Η μέθοδος αυτή εκτελείται όταν η applet εφαρμογή κατεβαίνει στο τοπικό υπολογιστή για εκτέλεση. Η μέθοδος αυτή μπορεί να περιλαμβάνει την δημιουργία κάποιων αντικειμένων, τον καθορισμό παραμέτρων, το φόρτωμα εικόνων ή font κλπ.

2. public void start(){

        . . .

    }

Η μέθοδος αυτή καλείται αμέσως μετά την init(). Η start() καλείται επίσης όταν η applet εφαρμογή είχε προηγουμένως σταματήσει την εκτέλεσή της. Για παράδειγμα μια applet εφαρμογή σταματά την εκτέλεσή της όταν ο χρήστης αλλάξει μέσω ενός συνδέσμου HTML σελίδα και ξαναρχίζει την εκτέλεσή της όταν ο χρήστης γυρίσει πίσω στη σελίδα της Applet εφαρμογής.

3. public void stop(){

        . . .

    }

Η stop() σταματά την εκτέλεση της Applet εφαρμογής και είναι το συμπλήρωμα της start(). Ο χρήστης μπορεί επίσης να καλέσει από μόνος του την stop για να σταματήσει την Applet εφαρμογή. Στην stop() μπορεί επίσης ο χρήστης να σταματά την εκτέλεση των Threads της Applet εφαρμογής (πράγμα που δεν γίνεται αυτόματα όταν αυτή σταματά να εκτελείται) και να τα ξαναρχίζει όταν αυτή αρχίζει πάλι την εκτέλεσή της.

4. public void destroy(){

        . . .

    }

Δίνει τη δυνατότητα στη Applet εφαρμογή να ελευθερώσει τους πόρους του συστήματος που της είχαν διατεθεί (πχ αντικείμενα, threads κλπ.). Η μέθοδος εκτελείται λίγο πριν η Applet εφαρμογή πάψει οριστικά την εκτέλεσή της ή όταν ο Browser κλείσει. Συνήθως η μέθοδος αυτή δεν χρειάζεται να οριστεί στην Applet εφαρμογή εκτός από πολύ ειδικές περιπτώσεις.

Κώδικας AlgoSortApplet.java

```
public class AlgoSortApplet extends java.applet.Applet {
  AlgoVisual visual;
  public void init() {
    .....
    visual = new AlgoVisual();
```

```
    add(visual);
  }
// Καλεί τη μέθοδο start() από τη κλάση AlgoVisual.java
  public void start() {
    visual.start();
  }
// Καλεί τη μέθοδο stop() από τη κλάση AlgoVisual.java
  public void stop() {
    visual.stop();
  }
// Καλεί τη μέθοδο destroy() από τη κλάση AlgoVisual.java
  public void destroy() {
    visual.destroy();
  }
}
```

Κώδικας AlgoVisual.java

```
class AlgoVisual extends Panel {
        ......
// καλεί τη μέθοδο doAppletStart() από τη κλάση AlgoSort.java για να ξεκινήσει την
// εκτέλεση της  εφαρμογής
        public void start() {
                sorter.doAppletStart();
        }

// καλεί τη μέθοδο doAppletStop() από τη κλάση AlgoSort.java για να σταματήσει την
// εκτέλεση του thread  «runner» της εφαρμογής
        public void stop() {
                sorter.doAppletStop();
        }

// εάν το thread «runner»  εκτελείται, τότε σταμάτησε την εκτέλεση της εφαρμογής
        public void destroy() {
                if (sorter.runner != null && sorter.runner.isAlive())
                        sorter.runner.stop();
        }
```

```
}
```

Κώδικας AlgoSort.java

```
class AlgoSort extends Canvas implements Runnable {
…….
// εάν το thread «runner»  εκτελείται, τότε σταμάτησε την εκτέλεση της εφαρμογή
        synchronized void stopRunning() {
                if (runner != null && runner.isAlive() && state == RUN) {
                        state = STOPPING;
                        notify();
                        while (state == STOPPING){
                                try { wait(); }
                        catch (InterruptedException e) { }
                }
        }

// καλεί τη μέθοδο stopRunning() για να σταματήσει την εκτέλεση του γραφήματος
        synchronized void doAppletStop() {
                oldState = state;
                stopRunning();
                state = APPLETSTOPPED;
                OSC = null;
                OSG = null;
        }
        synchronized void doAppletStart() {
                if (state != APPLETSTOPPED)
                        return;
                state = oldState;
                if (state == RUN || state == STEP)
                        notify();
        }
…….
}
```

4.4 Η εκτέλεση της εφαρμογής

Πολλές γλώσσες προγραμματισμού όπως και η Java διαθέτουν εργαλεία για την υλοποίηση threads στα προγράμματά τους. Αυτές οι γλώσσες καλούνται multithreading languages. Στον παραδοσιακό προγραμματισμό όταν ένα πρόγραμμα εκτελείται ονομάζεται process (διεργασία) και οι εντολές του εκτελούνται σειριακά η μία μετά την άλλη μέχρι το τέλος του. Σ' αυτή την περίπτωση μπορούμε να πούμε ότι το συγκεκριμένο process διαθέτει μόνο ένα thread που εκτελεί τις εντολές του προγράμματος. Στις multithreading γλώσσες προγραμματισμού μπορούμε να ορίσουμε σε κάποιο process να περιέχει ένα ή και περισσότερα threads (ονομάζονται και lightweight processes) οι οποίες εκτελούν κάθε μια ξεχωριστά τον κώδικα του προγράμματος. Είναι δηλαδή σαν να έχουμε πολλά processes που εκτελούνται παράλληλα μέσα στο αρχικό process.

Οι ενέργειες που λαμβάνουν χώρα κατά την εκτέλεση ενός Thread ορίζονται στη μέθοδο run του Thread. Μετά από την δημιουργία και την αρχικοποίηση ενός Thread το σύστημα εκτελεί αυτόματα τον κώδικα που υπάρχει στη μέθοδο run. Συνήθως η μέθοδος αυτή περιέχει κάποια επαναληπτική δομή.

Η java χρησιμοποιεί τα monitors για να παρέχει συγχρονισμό στα αντικείμενά της. Ένα αντικείμενο που περιέχει **synchnonized** μεθόδους θεωρείται αντικείμενο τύπου monitor. Ένα αντικείμενο τύπου monitor επιτρέπει σε ένα μόνο thread κάθε στιγμή να εκτελεί μια synchronized μέθοδο αυτού του αντικειμένου. Έτσι όταν καλείται κάποια synchronized μέθοδος ενός αντικειμένου τότε το αντικείμενο "κλειδώνεται" για αυτό το thread και δεν μπορεί να κληθεί καμία άλλη synchronized μέθοδος στο συγκεκριμένο αντικείμενο από κάποιο άλλο thread. Μη-synchronized μέθοδοι μπορούν να κληθούν κανονικά. Μετά το τέλος της εκτέλεσης μιας synchronized μεθόδου το αντικείμενο "ξεκλειδώνεται".

Μια thread που εκτελεί μια synchronized μέθοδο μπορεί να σταματήσει την εκτέλεσή της μόνο στην περίπτωση που από τον κώδικα της μεθόδου κληθεί η wait(). Σε μια τέτοια περίπτωση το thread μπαίνει σε κατάσταση wait για το συγκεκριμένο monitor αντικείμενο ενώ το αντικείμενο "ξεκλειδώνεται".

Όταν ένα thread εκτελέσει από τον κώδικα μιας synchronized μεθόδου, την μέθοδο notify(), τότε "ξυπνά". Το "ξυπνημένο" thread, δεν θα είναι σε θέση να εκτελεσθεί, έως ότου το τρέχον thread τελειώσει την εκτέλεση της synchronized μεθόδου, δηλαδή ξεκλειδώσει το monitor αντικείμενο. Εάν πολλά thread περιμένουν σε αυτό το

αντικείμενο, ένα μόνο από αυτά επιλέγεται για να "ξυπνήσει" - η επιλογή γίνεται αυθαίρετα από το σύστημα. Η μέθοδος notifyAll(), "ξυπνά" όλα τα threads που βρίσκονται σε κατάσταση wait για το συγκεκριμένο monitor αντικείμενο. Οι μέθοδοι wait(), notify(), notifyAll() ορίζονται στην κλάση java.lang.Object.

Η κλάση AlgoSort.java εμφανίζει την γραφική απεικόνιση της εφαρμογής και εκτελεί τους τρεις αλγόριθμους ταξινόμησης. Η κλάση αυτή υλοποιεί τη διεπαφή Runnable για να ορίσει την run μέθοδο του Thread «runner» που δημιουργεί. Ο λόγος που χρησιμοποιούμε το thread είναι για να μπορέσουμε να εκτελέσουμε το animation της εφαρμογής.

Παρακάτω θα δούμε τις καταστάσεις στις οποίες μπορεί να βρίσκεται το thread και τις μεθόδους με τις οποίες μπορεί να πηγαίνει από μιά κατάσταση σε μια άλλη. Οι μέθοδοι **startRunning()**, **stopRunning(), doStep(), doAppletStop()** είναι μέθοδοι που δηλώνουν την κατάσταση του thread και χειριζόμαστε κατόπιν αυτές τις καταστάσεις ανάλογα με την περίσταση.

```
class AlgoSort extends Canvas implements Runnable {
        Thread runner;
// Καταστάσεις του thread
        final static int APPLETSTOPPED = -1, IDLE = 0,STARTING = 1,STOPPED =
        2,STEP = 3,RUN = 4,STOPPING = 5;
        private int state = STARTING;
        private int oldState = STARTING;
…….

// Μέθοδος constructor που καλείται από την κλάση AlgoVisual.java και δέχεται ως
// ορίσματα τα χαρακτηριστικά της AlgoVisual, τις ετικέτες «συγκρίσεις», «Αντιμεταθέσεις»,
// «Τίτλος» και την ετικέτα  που εμφανίζει το μήνυμα «Η ταξινόμηση τελείωσε»  όταν μία
// ταξινόμηση τελειώσει, και τέλος το όρισμα slider που καλεί την κλάση Slider.java και
// ελέγχει την ταχύτητα του animation.
AlgoSort(AlgoVisual algo, Label comparisons, Label swaps,Slider slider, Label title, Label
sortFinished)  {
    this.algo = algo;
    this.comparisons = comparisons;
    this.swaps = swaps;
    this.slider = slider;
```

```
    this.title = title;
    this.sortFinished = sortFinished;
    setUpSortData();
  }

synchronized int getState() {
    return state;
  }
synchronized void setState(int state)  {
    this.state = state;
    notify();
  }

// Η μέθοδος setUpSortData() αρχικοποιεί τις τιμές των μεταβλητών κάθε φορά που
// διαλέγουμε έναν καινούργιο  αλγόριθμο να ταξινομήσουμε.
void setUpSortData()  {
    i2Loc = -1;
    iLoc = -1;
    i3Loc = -1;
    lastLoc = -1;
    leftLoc = -1;
    rightLoc = -1;
    maxLoc = -1;
    movingItem = -1;
    tempOn = false;
    for (int i = 1; i <= 10; i++)
      item[i] = i;
    for (int i = 10; i >= 2; i--)  {
      int j = 1 + (int)(Math.random()*i);
      int temp = item[i];
      item[i] = item[j];
      item[j] = temp;
    }
    item[0] = -1;
    for (int i = 11; i < 33; i++)
      item[i] = -1;
  }
```

```
// Κάθε φορά που επιλέγεται έναν καινούργιο αλγόριθμο, τότε εκτελείται η μέθοδος newSort
// (int sortMethod).
synchronized void newSort(int sortMethod) {
// Εάν εκτελείται άλλος αλγόριθμος, τότε καλείται η μέθοδος stopRunning() για να σταματήσει
    if (state == RUN)
        stopRunning();
// Η κατάσταση ορίζεται ως STARTING
    state = STARTING;
    setUpSortData();
    method = sortMethod;
    valid = false;
    if (method == 1)
        title.setText("Bubble Sort");
    else if (method == 2)
        title.setText("Insertion Sort");
    else
        title.setText("Heap Sort");
    sortFinished.setText("");
    comparisons.setText("0");
    swaps.setText("0");
    comparisonCt = 0;
    swap = 0;
// Καλείται η μέθοδος setChangedAll() για να αλλάξει το ύψος και την σειρά των ράβδων.
    setChangedAll();
    repaint();
  }

//  Η μέθοδος startRunning()  εκτελείται όταν πατήσουμε το κουμπί «Εκκίνηση», είτε μετά
// από Παύση είτε μετά από την επιλογή ενός καινούργιου αλγορίθμου.
synchronized void startRunning()  {
    if (state == IDLE)
        newSort(method);
    state = RUN;
    if (runner == null || !runner.isAlive()) {
        runner = new Thread(this);
        runner.start();
    }
```

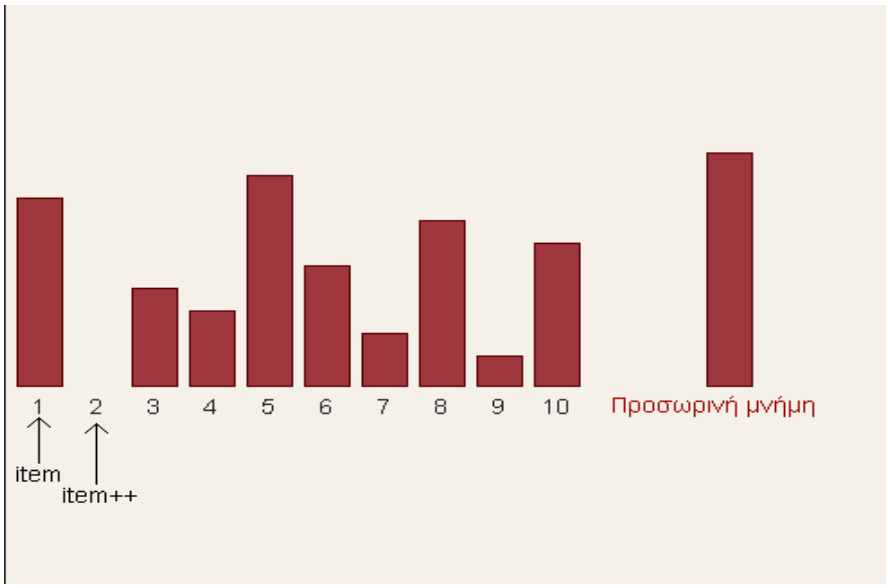
```
    else
        notify();
}

// Η μέθοδος αυτή εκτελείται όταν πατήσουμε το κουμπί «Παύση»
synchronized void stopRunning() {
    if (runner != null && runner.isAlive() && state == RUN) {
        state = STOPPING;
        notify();
        while (state == STOPPING){
            try { wait(); }
        catch (InterruptedException e) { }
        }
}

// Η μέθοδος doStep() εκτελείται όταν επιλέξουμε μία ταξινόμηση να εκτελείται βήμα - βήμα
synchronized void doStep() {
    if (state == IDLE)
        return;
    state = STEP;
    if (runner == null || !runner.isAlive()){
        runner = new Thread(this);
        runner.start();
    }
    else
        notify();
}

// Καλεί τη μέθοδο stopRunning() για να σταματήσει την εκτέλεση του αλγορίθμου
synchronized void doAppletStop() {
    oldState = state;
    stopRunning();
    state = APPLETSTOPPED;
    OSC = null;
    OSG = null;
}

synchronized void doAppletStart() {
```

```
    if (state != APPLETSTOPPED)
        return;
    state = oldState;
    if (state == RUN || state == STEP)
        notify();
}

// Στη μέθοδο run() ορίζονται οι εντολές τις οποίες πρόκειται να εκτελέσει το thread. Καλεί
// τις παραπάνω μεθόδους ανάλογα με την κατάσταση που βρίσκεται κάθε φορά.
public void run() {
    while (true){
        int st;
        synchronized(this){
                while (state <= STOPPED) {
                        try { wait();}
                    catch (InterruptedException e) { }
        }
        st = state;
    }
   if (st == STOPPING) {
        setState(STOPPED);
        algo.runnerStopped();
   }
   else  {
        scriptStep();

// Η μέθοδος repaint()αφού κάνει ένα καθαρισμό της επιφάνειας της οθόνης που αντιστοιχεί
// στο αντικείμενο της  γραφικής κλάσης θα εισάγει την αίτηση επανασχεδιασμού του
// γραφικού πλαισίου( θα δούμε παρακάτω σε  άλλες  μεθόδους να καλείται όταν χρειάζεται
//  να γίνει αλλαγή στην εικόνα)
        repaint();
        if (done) {
                    algo.doneRunning(method,comparisonCt,swap);
                    setState(IDLE);
                    repaint();
        }
        else if (getState() == STOPPING){
```

```
                    setState(STOPPED);
                    algo.runnerStopped();
            }
            else if (st == STEP && getState() != RUN)  {
                    setState(STOPPED);
                    algo.runnerStopped();
            }
            else
                    doWait(slider.getValue());
        }
    }
}
…….
}
```

4.5 Η γραφική απεικόνιση της εφαρμογής

Η κλάση Canvas χρησιμοποιείτε για την παροχή μίας επιφάνειας στην οποία μπορούν να εκτελεστούν λειτουργίες γραφικών.



Οι μέθοδοι που βλέπεται παρακάτω είναι από τη κλάση **AlgoSort.java** και σχεδιάζουν τα γραφικά της εφαρμογής.

```
class AlgoSort extends Canvas implements Runnable {
```

```
…
//  Εκτελεί την κινούμενη εικόνα στην εφαρμογή.
        Thread runner;

// Η Image OSC είναι μία  εικόνα όπου δημιουργείται για να  τοποθετήσει ένα πλαίσιο
// γραφικών.
         Image OSC;

// Τα Graphics OSG (Open Scene Graphics) τα χρησιμοποιούμε για να μπορέσουμε να
//σχεδιάσουμε στην εφαρμογή μας ένα πλαίσιο γραφικών για την OSC.
        Graphics OSG;

……

// Με τη μέθοδο reshape(int x, int y, int width, int height) τοποθετούμε τη γραφική
//  απεικόνιση της  εφαρμογής  στη σωστή θέση μέσα στην εφαρμογή, δίνοντας το πλάτος,
// ύψος και τις διαστάσεις.
    public void reshape(int x, int y, int width, int height) {
        super.reshape(160,50,width-165,height-15);
        if (width != this.width || height != this.height){
                OSC = null;
                OSG = null;
        }
    }

// Η μέθοδος paint(Graphics g) σχεδιάζει σε μία εικόνα  χρησιμοποιώντας ένα πλαίσιο
// γραφικών. Με τη μέθοδο  OSC = createImage(size().width,size().height); δημιουργεί την
// εικόνα,  όπου καθορίζει το πλάτος και το ύψος για να τοποθετήσει την εικόνα αυτή στην
// εφαρμογή. Με τη μέθοδο OSG = OSC.getGraphics(); παίρνει τα γραφικά που
// δημιουργούνται στις άλλες μεθόδους (Graphics g) για να τοποθετήσει μέσα στην
// εικόνα OSC.
    synchronized public void paint(Graphics g) {
// δημιουργεί την εικόνα, εάν δεν υπάρχει
        if (OSC == null || size().width != width || size().height != height)
        {
          try {
            OSC = createImage(size().width,size().height);
            OSG = OSC.getGraphics();
```

```
            }
            catch (OutOfMemoryError e) {
              OSC = null;
              OSG = null;
            }
            font = new Font("Arial",Font.PLAIN,14);
            fm = g.getFontMetrics(font);
            if (OSG != null)
              OSG.setFont(font);
            textAscent = fm.getAscent();
            setSizeData(size().width,size().height);
            setChanged(0,0,width,height);
          }
          if (OSC == null){
            g.setFont(font);
            draw(g,0,0,width,height);
          }
          else {
            if (changed)
              draw(OSG,changed_x,changed_y,changed_width,changed_height);
            g.drawImage(OSC,0,0,this);
          }
        }
```

// Η μέθοδος ***update(Graphics g)*** *εκτελείται κάθε φορά που χρειάζεται να επανασχεδιάσουμε*
// το πλαίσιο γραφικών καλώντας τη μέθοδο ***paint(g).***

```
    public void update(Graphics g) {
        paint(g);
      }
```

// Η μέθοδος ***setSizeData(int w, int h)*** *ορίζει το ύψος ,το πλάτος και τις θέσεις των ράβδων.*

```
    void setSizeData(int w, int h) {
        width = w;
        height = h;
        int x = (width - 20 + barGap)/15;
        barWidth =  x - barGap;
        leftOffset = (width - 15*barWidth - 16*barGap);
```

```
        barHeight = (height - 120 - 2*textAscent) ;
        barIncrement = (barHeight-3)/16;
        minBarHeight = barHeight - 16*barIncrement;
        firstRow_y = barHeight + 10;
        memTemp = barHeight + 10;
      }
```

// Καθώς εκτελείται μία ταξινόμηση η μέθοδος ***putItem(Graphics g, int i)*** *τοποθετεί την*
// κάθε ράβδο στη σωστή θέση και αλλάζει το χρώμα της ανάλογα στη θέση που βρίσκεται.

```
    void putItem(Graphics g, int i) {
        int h = item[i];
        if (h == -1)
          return;
        int x,y,ht;
// Αν η ράβδος έχει ήδη ταξινομηθεί το χρώμα της γίνεται σκούρο κόκκινο
        if (h > 10){
          ht = (h-100)*barIncrement + minBarHeight;
          g.setColor(finishedBarColor);
        }
// Αλλιώς όχι
        else {
          ht = h*barIncrement + minBarHeight;
          g.setColor(barColor);
        }
        if (i == 0){
          x = leftOffset + ((barWidth+barGap)*12);
          y = memTemp - ht;
        }
        else if (i < 11) {
          x = leftOffset + (i-1)*(barWidth+barGap);
          y = firstRow_y - ht;
        }
        else {
          x = leftOffset + (i-11)*(barWidth+barGap);
          y = memTemp - ht;
        }
        g.fillRect(x,y,barWidth,ht);
```

```
            g.setColor(finishedBarColor);
            g.drawRect(x,y,barWidth,ht);
        }
```

*// Η μέθοδος **drawItem(Graphics g)** σχεδιάζει το String «Item» και το βέλος δίνοντας τις*
//διαστάσεις x και y και το χρώμα για να τοποθετηθεί στο σωστό σημείο μέσα στη γραφική
//απεικόνιση της εφαρμογής

```
    void drawItem(Graphics g) {
        int sw = fm.stringWidth("item");
        int x = leftOffset + (iLoc-1)*(barWidth+barGap) + barWidth/2;
        int y = firstRow_y + 38 + textAscent;
        g.setColor(itemColor);
        g.drawString("item",x-sw/2,y+textAscent);
        g.drawLine(x,y,x,y-30);
        g.drawLine(x,y-30,x+6,y-24);
        g.drawLine(x,y-30,x-6,y-24);
    }
```

*// Η μέθοδος **draw(Graphics g, int x, int y, int w, int h)** καλείται από την μέθοδο*
*// **paint(Graphics g)** για να δημιουργήσει τα γραφικά στην εικόνα OSC. Δηλώνει τις*
// διαστάσεις της εικόνας, τα /χρώματα, ορίζει την τοποθεσία των ράβδων, σχεδιάζει και
// τοποθετεί στη σωστή θέση τα νούμερα 1,2,3 ... 10, την Προσωρινή μνήμη.

```
    synchronized void draw(Graphics g, int x, int y, int w, int h) {
        g.setColor(backgroundColor);
        g.fillRect(x,y,w,h);
        g.setColor(borderColor);
        g.drawRect(0,0,width,height);
        g.drawRect(1,1,width-2,height-2);
        g.drawLine(0,height-2,width,height-2);
        g.drawLine(width-2,0,width-2,height);
        int firstBar = (x-10)/(barWidth+barGap) + 1;
        if (firstBar < 1)
          firstBar = 1;
        int lastBar = (x+w-10)/(barWidth+barGap) + 1;
        if (lastBar > 10)
          lastBar = 10;
        if (y <= firstRow_y) {
```

```
          for (int i = firstBar; i <= lastBar; i++)
            putItem(g,i);
        }
        if (y <= firstRow_y + 10 + textAscent && y+h > firstRow_y) {
          g.setColor(borderColor);
          for (int i = firstBar; i <= lastBar; i++) {
            String str = String.valueOf(i);
            int sw = fm.stringWidth(str);
            g.drawString(str,leftOffset+(i-1)*(barWidth+barGap)+(barWidth-
    sw)/2,firstRow_y+6+textAscent);
          }
        }
        if (y <= memTemp  && y+h >= memTemp - barHeight) {
          for (int i = 10 + firstBar; i <= 10 + lastBar; i++)
            putItem(g,i);
        }
       if (tempOn)   {
         g.setColor(barColor1);
         int sw = fm.stringWidth("Προσωρινή μνήμη");
         g.drawString("Προσωρινή μνήμη",leftOffset + (15*barWidth+10*barGap - sw),
    firstRow_y + 5 + textAscent);
         putItem(g,0);
       }
```

// Ανάλογα την ταξινόμηση που εκτελείται δηλώνει τις μεταβλητές ***iLoc, lastLoc, leftLoc***

// ...και καλεί τις αντίστοιχες μεθόδους για να σχεδιάσει τα γραφικά ***item, last, left….***

```
       if (i2Loc >= 0)
         drawItem2(g);
       if (lastLoc >= 0)
         drawLast(g);
       if (leftLoc >= 0)
         drawLeft(g);
       if (rightLoc >= 0)
         drawRight(g);
       if (iLoc >= 0)
         drawItem(g);
       if (i3Loc >= 0)
```

```
            drawItem3(g);
          if (maxLoc >= 0)
            drawMax(g);
          changed = false;
        }
```

*// Η μέθοδος **putItem1(int itemNum)** τοποθετεί κάθε φορά το String «item» και το βέλος να //δείχνει στη ράβδο που γίνεται η σύγκριση.*

```
    void putItem1(int itemNum) {
        int sw = fm.stringWidth("item") + 4;
// Η μεταβλητή «iLoc» δηλώνει τη θέση που βρίσκεται το String «item» και το βέλος
        if (iLoc != -1)
          setChanged(leftOffset + (iLoc-1)*(barWidth+barGap) + (barWidth-
    sw)/2,firstRow_y+1,sw,50+textAscent);
        iLoc = itemNum;
      }
……
}
```

4.6 Η εισαγωγή της εφαρμογής στην Ιστοσελίδα

Το tag < object > δέχεται αρκετές παραμέτρους. Η πιο σημαντική παράμετρος είναι η «type». Άλλες σημαντικές παράμετροι είναι οι WIDTH και HEIGHT, οι οποίες καθορίζουν το πλάτος και το ύψος της περιοχής που θα καταλαμβάνει το Applet στην οθόνη του Browser.

Το tag <PARAM> έχει πολύ πιο απλή σύνταξη: οι μόνες παράμετροι που περιέχει είναι οι NAME και VALUE. Η NAME ορίζει ένα όνομα για μια παράμετρο που θέλουμε να περάσουμε στο Applet ενώ η VALUE ορίζει την τιμή της.

Μέσα στα tags < object > μπορούν να περιληφθούν διάφορα αναγνωριστικά που έχουν να κάνουν με τον τρόπο εμφάνισης της Applet εφαρμογής στη Web σελίδα:

- **archive**: Προσδιορίζει το όνομα του jar αρχείου στο οποίο περιέχεται το applet και οι πόροι του (αν το applet και οι πόροι του περιέχονται σε jar αρχείο).
- **type**: Προσδιορίζει τον τύπο του plugin. Μπορεί να είναι applet ή και bean. Το bean χρησιμοποιείται για την ενσωμάτωση Java Beans σε μία σελίδα .

- **code**: Προσδιορίζει το όνομα της τάξης του applet, καθορίζει το directory στο οποίο βρίσκεται η κλάση που ορίστηκε με την παράμετρο CODE.
- **width**: Προσδιορίζει το πλάτος του applet σε pixels.
- **height**: Προσδιορίζει το ύψος του applet σε pixels.
- **param** : Δηλώνει την παράμετρο που δέχεται η Applet εφαρμογή, στο αναγνωριστικό «name» ορίζεται το όνομα της παραμέτρου ενώ στο «value» η τιμή της.

Το HTML αρχείο που περιέχει την Applet έχει τον παρακάτω κώδικα:

```
<HTML>
<HEAD>
<TITLE> Οπτικοποίηση Αλγορίθμων  </TITLE>
</HEAD>
<BODY>
.....
<div id="applet">
<object type="application/x-java-applet" width="670" height="460">
        <param name="archive" value="SortingAlgorithms.jar">
        <param name="code" value="VisualizeAlgo/AlgoSortApplet.class">
        <param name="background" value="#FFFFFF">
</object>
</div>
.....
</BODY>
</HTML>
```

Στην παρακάτω εικόνα φαίνεται η εφαρμογή στον Web Browser :

http://aetos.it.teithe.gr/~ankri/

ΠΑΡΑΡΤΗΜΑ

Κώδικας Υλοποίησης

Παραθέτονται οι βασικότερες κλάσεις που απαρτίζουν την υλοποίηση της εφαρμογής «Οπτικοποίηση Αλγορίθμων» (Algorithms Visualization).

Κλάση AlgoSortApplet.java

```
import java.awt.*;
import java.applet.*;
public class AlgoSortApplet extends java.applet.Applet {
        AlgoVisual visual;
        public void init() {
                setBackground(new Color(30,32,36));
                setLayout(new BorderLayout());
                visual = new AlgoVisual();
                add(visual);
        }
        public Insets insets() {
                return new Insets(5,5,5,5);
        }
        public void start() {
                visual.start();
        }
        public void stop() {
                 visual.stop();
        }
        public void destroy() {
                visual.destroy();
        }
}
```

Κλάση AlgoSortFrame.java

```
class AlgoSortFrame extends Frame {
        public static void main(String[] args) {
```

```
            AlgoSortFrame frame = new AlgoSortFrame();
        }
        public static void runNoExit() {
            AlgoSortFrame frame = new AlgoSortFrame();
            frame.exitOnClose = false;
        }
        private  AlgoVisual visual;
        private boolean closed = false;
        boolean exitOnClose = true;
        AlgoSortFrame() {
            super("Οπτικοποίηση Αλγορίθμων");
            visual = new AlgoVisual();
            add(visual);
            reshape(20,30,680,480);
            setResizable(false);
            show();
        }
        boolean isClosed() {
            return closed;
        }
         void close() {
            visual.stop();
             visual.destroy();
            closed = true;
            dispose();
            if (exitOnClose)
                System.exit(0);
        }
        public boolean handleEvent(Event evt) {
             if (evt.id == Event.WINDOW_DESTROY) {
                 close();
                return true;
            }
            else
                return super.handleEvent(evt);
        }
}
```

Κλάση AlgoVisual.java

```
class AlgoVisual extends Panel {

    final static String[] sortName = { "Bubble Sort", "Insertion Sort" , "Heap Sort" };
    Choice sortMethodChoice;
    int currentSortMethod = 0;
    Button goButton,stepButton,restartButton,help,about;
    Label  label1, label2,  label3, title, sortFinished;
    Label comparisons, swaps , comp1, swap;
    Panel  controlPanel, infoPanel, messagePanel, titlePanel, sortPanel, sliderPanel;
    About ab;
    AlgoSort sorter;
    Slider slider;
    HelpWindow helpWindow;
    Font myFont = new Font("Serif",Font.BOLD,16);
    Font myFont1 = new Font("Serif",Font.BOLD,14);
    Font myFont2 = new Font("Serif",Font.BOLD,18);
    Font myFont3 = new Font("Arial",Font.BOLD,20);

     AlgoVisual() {
            setBackground(new Color(30,32,36));
            setLayout(null);
            controlPanel = new Panel();
            setLayout(new BorderLayout(5,5));
            add("North",controlPanel);
            label1 = new Label("Επιλογή Αλγορίθμου :");
            label1.setFont(myFont2);
            label1.setForeground(new Color(238,229,222));
            controlPanel.add(label1);
            sortMethodChoice = new Choice();
            for (int i = 0; i < sortName.length; i++)
            sortMethodChoice.addItem(sortName[i]);
            sortMethodChoice.setForeground(Color.darkGray);
            sortMethodChoice.setFont(myFont1);
            controlPanel.add(sortMethodChoice);
            controlPanel.add(label2);
```

```
goButton = new Button("Εκκίνηση");
goButton.setForeground(new Color(153,0,0));
goButton.setFont(myFont1);
controlPanel.add(goButton);
stepButton = new Button("Επόμενο βήμα");
stepButton.setForeground(new Color(30,32,36));
stepButton.setFont(myFont1);
controlPanel.add(stepButton);
restartButton = new Button("Επανεκκίνηση");
restartButton.setForeground(new Color(30,32,36));
restartButton.setFont(myFont1);
controlPanel.add(restartButton);
titlePanel = new Panel();
titlePanel.setLayout(new GridLayout(2,1,5,5));
titlePanel.setBackground(new Color(30,32,36));
add(titlePanel);
title = new Label("Bubble Sort", Label.CENTER);
title.setFont(myFont3);
title.setForeground(new Color(143,188,143));
titlePanel.add(title);
titlePanel.add(label3);
messagePanel = new Panel();
messagePanel.setBackground(new Color(30,32,36));
messagePanel.setLayout(new GridLayout(3,2,5,5));
messagePanel.setLayout(new FlowLayout(1));
add(messagePanel);
comp1 = new Label("    Συγκρίσεις:");
messagePanel.add(comp1);
comp1.setForeground(new Color(238,229,222));
comp1.setFont(myFont);
comparisons = new Label("0", Label.CENTER);
comparisons.setForeground(new Color(238,229,222));
comparisons.setFont(myFont);
messagePanel.add(comparisons);
swap = new Label("Αντιμεταθέσεις:");
messagePanel.add(swap);
swap.setForeground(new Color(238,229,222));
```

```
swap.setFont(myFont);
swaps = new Label("0", Label.CENTER);
swaps.setForeground(new Color(238,229,222));
swaps.setFont(myFont);
messagePanel.add(swaps);
sliderPanel = new Panel();
sliderPanel.setLayout(new GridLayout(2,1,5,5));
sliderPanel.setLayout(new FlowLayout(1));
add(sliderPanel);
slider = new Slider("Ταχύτητα",400, 40, 30, 800);
slider.setBackground(new Color(30,32,36));
sliderPanel.add(slider);
slider.setSize(35,20);
sortPanel = new Panel();
sortPanel.setLayout(new GridLayout(1,1,5,5));
sortPanel.setBackground(new Color(245,241,233));
add(sortPanel);
sortFinished = new Label("");
sortFinished.setForeground(new Color(165,42,42));
sortFinished.setFont(myFont2);
sortPanel.add(sortFinished);
infoPanel = new Panel();
infoPanel.setLayout(new GridLayout(3,1,5,5));
infoPanel.setLayout(new FlowLayout(1));
infoPanel.setBackground(new Color(30,32,36));
infoPanel.setForeground(Color.red);
help = new Button("    Βοήθεια    ");
help.setFont(myFont1);
help.setForeground(Color.darkGray);
about = new Button("     About      ");
about.setFont(myFont1);
about.setForeground(Color.darkGray);
add(infoPanel);
label2 = new Label("      Πληροφορίες      ", Label.CENTER);
label2.setFont(myFont2);
label2.setForeground(new Color(238,229,222));
infoPanel.add(label2);
```

```
        infoPanel.add(help);
        infoPanel.add(about);

        sorter = new AlgoSort(this,comparisons, swaps,slider, title, sortFinished);
        add(sorter);
        ab = new About();
        helpWindow = new HelpWindow();
}
public void reshape(int x, int y, int width, int height) {
        super.reshape(x,y,width,height);
        if (width < 300)
                width = 300;
        if (height < 200)
                height = 200;
        controlPanel.reshape(width-120,0,120,0);
        infoPanel.reshape(0,289,160,160);
        messagePanel.reshape(0,110,160,90);
        titlePanel.reshape(0,70,160,52);
        sortPanel.reshape(270,390,240,40);
        sliderPanel.reshape(0,200,160,80);
}
void doneRunning(int method, int comparisonCt, int swap) {
        goButton.enable();
        goButton.setLabel("Εκκίνηση");
        stepButton.disable();
}
void runnerStopped() {
        goButton.enable();
        goButton.setLabel("Εκκίνηση");
        stepButton.enable();
}
public void start() {
         sorter.doAppletStart();
}
public void stop() {
        sorter.doAppletStop();
}
```

```
public void destroy() {
        if (sorter.runner != null && sorter.runner.isAlive())
                sorter.runner.stop();
}
public boolean action(Event evt, Object arg) {
        if (evt.target == sortMethodChoice) {
                int choice = sortMethodChoice.getSelectedIndex();
                if (choice == currentSortMethod)
                        return true;
                currentSortMethod = choice;
                sorter.newSort(currentSortMethod+1);
                return true;
        }
        else if(evt.target == slider){
                synchronized(sorter) {
                        if (sorter.getState() == AlgoSort.RUN)
                                sorter.doWait(slider.getValue());
                }
        }
        else if (evt.target == goButton) {
                synchronized(sorter) {
                        if (sorter.getState() == AlgoSort.RUN) {
                                goButton.disable();
                                sorter.stopRunning();
                        }
                        else {
                                goButton.setLabel("Παύση");
                                stepButton.disable();
                                sorter.startRunning();
                        }
                }
                return true;
        }
        else if (evt.target == stepButton) {
                sorter.doStep();
                synchronized(sorter) {
                        int st = sorter.getState();
```

```
                    if (st == AlgoSort.STEP) {
                        stepButton.disable();
                        goButton.disable();
                    }
                }
                return true;
            }
            else if (evt.target == restartButton) {
                sorter.newSort(currentSortMethod+1);
                goButton.setLabel("Εκκίνηση");
                stepButton.enable();
                return true;
                }
            else if (evt.target == about) {
                sorter.stopRunning();
                ab.show();
            }
            else if (evt.target == help) {
                sorter.stopRunning();
                helpWindow.show();
            }
            else
                return super.action(evt,arg);
            return false;
        }
}
```

Κλάση AlgoSort.java

```
class AlgoSort extends Canvas implements Runnable {

  Thread runner;
 final static int APPLETSTOPPED = -1, IDLE = 0,STARTING = 1,STOPPED = 2,STEP = 3,RUN
=  4 ,                STOPPING = 5;
  private int state = STARTING;
  private int oldState = STARTING;
```

```
    Image OSC;
    Graphics OSG;
    boolean changed, tempOn;
    Label comparisons, swaps, title, sortFinished;
    int changed_x, changed_y, changed_width, changed_height;
    int width = -1, height;
    int[] item = new int[33];
    int movingItem, iLoc, i2Loc,i3Loc, leftLoc, rightLoc, lastLoc, maxLoc;
    int barWidth, barHeight, minBarHeight, barIncrement, leftOffset, firstRow_y, memTemp,
textAscent;
    int swap,comparisonCt, titl;
    static final int barGap = 7;
    Font font;
    FontMetrics fm;
    static final Color backgroundColor = new Color(245,241,233);
    static final Color borderColor = new Color(30,32,36);
    static final Color barColor = new Color(158,55,62);
    static final Color barColor1 = new Color(153,0,0);
    static final Color finishedBarColor = new Color(89,0,0);
    static final Color itemColor = Color.black;
    static final Color maxColor = Color.red;
    static final Color lastColor = new Color(138,43,226);
    static final Color leftRightColor = Color.DARK_GRAY;
    Slider slider;
    AlgoVisual algo;

    AlgoSort(AlgoVisual algo, Label comparisons, Label swaps,Slider slider, Label title, Label
sortFinished) {
            this.algo = algo;
            this.comparisons = comparisons;
            this.swaps = swaps;
            this.slider = slider;
            this.title = title;
            this.sortFinished = sortFinished;
             setUpSortData();
    }
    public void reshape(int x, int y, int width, int height) {
```

```
        super.reshape(160,50,width-165,height-15);
         if (width != this.width || height != this.height){
                OSC = null;
                OSG = null;
        }
}
synchronized int getState() {
        return state;
}
synchronized void setState(int state)  {
        this.state = state;
        notify();
}
void setUpSortData()  {
        i2Loc = -1;
        iLoc = -1;
        i3Loc = -1;
        lastLoc = -1;
        leftLoc = -1;
        rightLoc = -1;
        maxLoc = -1;
        movingItem = -1;
        tempOn = false;
        for (int i = 1; i <= 10; i++)
                item[i] = i;
        for (int i = 10; i >= 2; i--)  {
                int j = 1 + (int)(Math.random()*i);
                int temp = item[i];
                item[i] = item[j];
                item[j] = temp;
        }
        item[0] = -1;
        for (int i = 11; i < 33; i++)
                item[i] = -1;
}
synchronized void newSort(int sortMethod) {
        if (state == RUN)
```

```
                stopRunning();
        state = STARTING;
        setUpSortData();
        method = sortMethod;
        valid = false;
        if (method == 1)
                title.setText("Bubble Sort");
        else if (method == 2)
                title.setText("Insertion Sort");
        else
                title.setText("Heap Sort");
        sortFinished.setText("");
        comparisons.setText("0");
        swaps.setText("0");
        comparisonCt = 0;
        swap = 0;
        setChangedAll();
        repaint();
}
synchronized public void paint(Graphics g) {
        if (OSC == null || size().width != width || size().height != height) {
                try {
                        OSC = createImage(size().width,size().height);
                        OSG = OSC.getGraphics();
                }
                catch (OutOfMemoryError e) {
                        OSC = null;
                        OSG = null;
                }
                font = new Font("Arial",Font.PLAIN,14);
                fm = g.getFontMetrics(font);
                if (OSG != null)
                        OSG.setFont(font);
                textAscent = fm.getAscent();
                setSizeData(size().width,size().height);
                setChanged(0,0,width,height);
        }
```

```
        if (OSC == null){
                g.setFont(font);
                draw(g,0,0,width,height);
        }
        else {
                if (changed)
                        draw(OSG,changed_x,changed_y,changed_width,changed_height);
                g.drawImage(OSC,0,0,this);
        }
}
public void update(Graphics g) {
        paint(g);
}
void setSizeData(int w, int h) {
        width = w;
        height = h;
        int x = (width - 20 + barGap)/15;
        barWidth =  x - barGap;
        leftOffset = (width - 15*barWidth - 16*barGap);
        barHeight = (height - 120 - 2*textAscent) ;
        barIncrement = (barHeight-3)/16;
        minBarHeight = barHeight - 16*barIncrement;
        firstRow_y = barHeight + 10;
        memTemp = barHeight + 10;
}

synchronized void setChanged(int x, int y, int w, int h) {
        if (changed){
                int x1 = Math.min(x,changed_x);
                int y1 = Math.min(y,changed_y);
                int x2 = Math.max(x+w,changed_x+changed_width);
                int y2 = Math.max(y+h,changed_y+changed_height);
                changed_x = x1;
                changed_y = y1;
                changed_width = x2 - x1;
                changed_height = y2-y1;
        }
```

```
        else   {
                changed_x = x;
                changed_y = y;
                changed_width = w;
                changed_height = h;
        }
        changed = true;
}
void setChangedAll() {
        setChanged(0,0,size().width,size().height);
}
void putItem(Graphics g, int i) {
        int h = item[i];
        if (h == -1)
                return;
        int x,y,ht;
        if (h > 10){
                ht = (h-100)*barIncrement + minBarHeight;
                g.setColor(finishedBarColor);
        }
        else {
                ht = h*barIncrement + minBarHeight;
                g.setColor(barColor);
        }
        if (i == 0){
                 x = leftOffset + ((barWidth+barGap)*12);
                y = memTemp - ht;
        }
        else if (i < 11) {
                x = leftOffset + (i-1)*(barWidth+barGap);
                y = firstRow_y - ht;
        }
        else {
                x = leftOffset + (i-11)*(barWidth+barGap);
                y = memTemp - ht;
        }
        g.fillRect(x,y,barWidth,ht);
```

```
        g.setColor(finishedBarColor);
        g.drawRect(x,y,barWidth,ht);
}
void drawLeft(Graphics g) {
        int sw = fm.stringWidth("left");
        int x = leftOffset + (leftLoc-1)*(barWidth+barGap) + barWidth/2;
        int y = firstRow_y + 38 + textAscent;
        g.setColor(leftRightColor);
        g.drawString("left",x-sw/2,y+textAscent);
        g.drawLine(x,y,x,y-30);
        g.drawLine(x,y-30,x+6,y-24);
        g.drawLine(x,y-30,x-6,y-24);
}
void drawLast(Graphics g) {
        int sw = fm.stringWidth("last");
        int x = leftOffset + (lastLoc-1)*(barWidth+barGap) + barWidth/2;
        int y = firstRow_y + 38 + textAscent;
        g.setColor(lastColor);
        g.drawString("last",x-sw/2,y+textAscent);
        g.drawLine(x,y,x,y-30);
        g.drawLine(x,y-30,x+6,y-24);
        g.drawLine(x,y-30,x-6,y-24);
}
void drawRight(Graphics g) {
        int sw = fm.stringWidth("right");
        int x = leftOffset + (rightLoc-1)*(barWidth+barGap) + barWidth/2;
        int y = firstRow_y + 38 + textAscent;
        g.setColor(leftRightColor);
        g.drawString("right",x-sw/2,y+textAscent);
        g.drawLine(x,y,x,y-30);
        g.drawLine(x,y-30,x+6,y-24);
        g.drawLine(x,y-30,x-6,y-24);
}
void drawItem(Graphics g) {
        int sw = fm.stringWidth("item");
        int x = leftOffset + (iLoc-1)*(barWidth+barGap) + barWidth/2;
        int y = firstRow_y + 38 + textAscent;
```

```
        g.setColor(itemColor);
        g.drawString("item",x-sw/2,y+textAscent);
        g.drawLine(x,y,x,y-30);
        g.drawLine(x,y-30,x+6,y-24);
        g.drawLine(x,y-30,x-6,y-24);
}
void drawItem2(Graphics g) {
        int sw = fm.stringWidth(" item++");
        int x = leftOffset + (i2Loc-1)*(barWidth+barGap) + barWidth/2;
        int y = firstRow_y + 52 + textAscent;
        g.setColor(itemColor);
        g.drawString(" item++",x-sw/2,y+textAscent);
        g.drawLine(x,y,x,y-40);
        g.drawLine(x,y-40,x+6,y-34);
        g.drawLine(x,y-40,x-6,y-34);
}
void drawItem3(Graphics g) {
        int sw = fm.stringWidth(" item--");
        int x = leftOffset + (i3Loc-1)*(barWidth+barGap) + barWidth/2;
        int y = firstRow_y + 38 + textAscent;
        g.setColor(itemColor);
        g.drawString(" item--",x-sw/2,y+textAscent);
        g.drawLine(x,y,x,y-30);
        g.drawLine(x,y-30,x+6,y-24);
        g.drawLine(x,y-30,x-6,y-24);
}
void drawMax(Graphics g) {
        int sw = fm.stringWidth("max");
        int x = leftOffset + (maxLoc-1)*(barWidth+barGap) + barWidth/2;
        int y = firstRow_y + 52 + textAscent;
        g.setColor(maxColor);
        g.drawString("max",x-sw/2,y+textAscent);
}

synchronized void draw(Graphics g, int x, int y, int w, int h) {
        g.setColor(backgroundColor);
        g.fillRect(x,y,w,h);
```

```
g.setColor(borderColor);
g.drawRect(0,0,width,height);
g.drawRect(1,1,width-2,height-2);
g.drawLine(0,height-2,width,height-2);
g.drawLine(width-2,0,width-2,height);
int firstBar = (x-10)/(barWidth+barGap) + 1;
if (firstBar < 1)
        firstBar = 1;
int lastBar = (x+w-10)/(barWidth+barGap) + 1;
if (lastBar > 10)
        lastBar = 10;
if (y <= firstRow_y) {
        for (int i = firstBar; i <= lastBar; i++)
                putItem(g,i);
}
if (y <= firstRow_y + 10 + textAscent && y+h > firstRow_y) {
        g.setColor(borderColor);
        for (int i = firstBar; i <= lastBar; i++) {
                String str = String.valueOf(i);
                int sw = fm.stringWidth(str);
                g.drawString(str,leftOffset+(i-1)*(barWidth+barGap)+(barWidth-sw)/2
                                        ,firstRow_y +6+
                                          textAscent);
        }
}
if (y <= memTemp  && y+h >= memTemp - barHeight) {
        for (int i = 10 + firstBar; i <= 10 + lastBar; i++)
                putItem(g,i);
}
if (tempOn)   {
        g.setColor(barColor1);
        int sw = fm.stringWidth("Προσωρινή μνήμη");
        g.drawString("Προσωρινή μνήμη",leftOffset + (15*barWidth+10*barGap - sw),
                                firstRow_y + 5 + textAscent);
        putItem(g,0);
}
if (i2Loc >= 0)
```

```
                drawItem2(g);
        if (lastLoc >= 0)
                drawLast(g);
        if (leftLoc >= 0)
                drawLeft(g);
        if (rightLoc >= 0)
                drawRight(g);
        if (iLoc >= 0)
                drawItem(g);
        if (i3Loc >= 0)
                drawItem3(g);
        if (maxLoc >= 0)
                drawMax(g);
        changed = false;
}
synchronized void startRunning()  {
        if (state == IDLE)
                newSort(method);
        state = RUN;
        if (runner == null || !runner.isAlive()) {
                runner = new Thread(this);
                runner.start();
        }
        else
                notify();
}
synchronized void stopRunning() {
        if (runner != null && runner.isAlive() && state == RUN) {
                state = STOPPING;
                notify();
                while (state == STOPPING){
                        try { wait(); }
                        catch (InterruptedException e) { }
                }
        }
}
synchronized void doStep(){
```

```
        if (state == IDLE)
                return;
        state = STEP;
        if (runner == null || !runner.isAlive()){
                runner = new Thread(this);
                runner.start();
        }
        else
                notify();
}
synchronized void doAppletStop(){
        oldState = state;
        stopRunning();
        state = APPLETSTOPPED;
        OSC = null;
        OSG = null;
}
synchronized void doAppletStart(){
        if (state != APPLETSTOPPED)
                 return;
        state = oldState;
        if (state == RUN || state == STEP)
                notify();
}
synchronized void doWait(int millis){
        try { wait(millis); }
        catch (InterruptedException e) { }
}
public void run() {
        while (true){
                int st;
                synchronized(this){
                        while (state <= STOPPED) {
                                try { wait();}
                                catch (InterruptedException e) { }
                }
                st = state;
```

```
        }
        if (st == STOPPING) {
                setState(STOPPED);
                algo.runnerStopped();
        }
        else  {
                scriptStep();
                 repaint();
                if (done) {
                                algo.doneRunning(method,comparisonCt,swap);
                                setState(IDLE);
                                repaint();
                }
                else if (getState() == STOPPING){
                                setState(STOPPED);
                                algo.runnerStopped();
                }
                else if (st == STEP && getState() != RUN)  {
                                setState(STOPPED);
                                algo.runnerStopped();
                }
                else
                                doWait(slider.getValue());
        }
        }
}
void invalidate(int itemNumber, int expandedBy) {
        doWait(slider.getValue());
        if (itemNumber < 0)
                return;
        int x,y;
        if (itemNumber == 0) {
                x = leftOffset + ((barWidth+barGap)*12);
                y = memTemp - barHeight;
        }
        else if (itemNumber < 11) {
                x = leftOffset + (itemNumber-1)*(barWidth+barGap);
```

```
                y = firstRow_y - barHeight;
        }
        else  {
                x = leftOffset + (itemNumber-11)*(barWidth+barGap);
                y = memTemp - barHeight;
        }
        setChanged(x-expandedBy,y-
expandedBy,barWidth+1+2*expandedBy,barHeight+1+2*expandedBy);
  }

  void putTemp(boolean on) {
        if (tempOn == on)
                return;
        tempOn = on;
        setChanged(0,memTemp+1,width,height-memTemp);
  }
  void putLeft(int itemNum) {
        int sw = fm.stringWidth("left") + 4;
        if (leftLoc != -1)
                  setChanged(leftOffset + (leftLoc-1)*(barWidth+barGap) + (barWidth-sw)/2,
                                firstRow_y+1, sw,50+textAscent);
        leftLoc = itemNum;
  }
  void putLast(int itemNum) {
        int sw = fm.stringWidth("last") + 4;
        if (lastLoc != -1)
                  setChanged(leftOffset + (lastLoc-1)*(barWidth+barGap) + (barWidth-
                                sw)/2,firstRow_y+1,sw,50+ textAscent);
        lastLoc = itemNum;
  }
  void putRight(int itemNum) {
        int sw = fm.stringWidth("right") + 4;
        if (rightLoc != -1)
                  setChanged(leftOffset + (rightLoc-1)*(barWidth+barGap) + (barWidth-
                                sw)/2,firstRow_y+1,sw,54+ textAscent);
        rightLoc = itemNum;
  }
```

```
  void putItem1(int itemNum) {
        int sw = fm.stringWidth("item") + 4;
        if (iLoc != -1)
              setChanged(leftOffset + (iLoc-1)*(barWidth+barGap) + (barWidth-
sw)/2,firstRow_y+1,sw,50+       textAscent);
        iLoc = itemNum;
  }
  void putItem2(int itemNum) {
        int sw = fm.stringWidth(" item++") + 4;
        if (i2Loc != -1)
                    setChanged(leftOffset + (i2Loc)*(barWidth+barGap) + (barWidth-
                          sw)/2,firstRow_y+1,sw,64+ textAscent);
        i2Loc = itemNum;
  }
  void putItem3(int itemNum) {
        int sw = fm.stringWidth(" item--") + 4;
        if (i3Loc != -1)
                    setChanged(leftOffset + (i3Loc)*(barWidth+barGap) + (barWidth-
                          sw)/2,firstRow_y+1,sw,50+ textAscent);
        i3Loc = itemNum;
  }
  void putMax(int itemNum) {
        int sw = fm.stringWidth("max") + 4;
        if (maxLoc != -1)
              setChanged(leftOffset + (maxLoc-1)*(barWidth+barGap) + (barWidth-
                    sw)/2,firstRow_y+1,sw,68+ textAscent);
        maxLoc = itemNum;
  }
  void itemChanged(int itemNum) {
        invalidate(itemNum,0);
        repaint();
  }

  int method = 1;
  boolean done;
  int i, j, k;
  boolean valid = false;
```

```
  int heapSize, node, child;
  int left, right, largest;

void copyItem(int toItem, int fromItem) {
        doWait(slider.getValue());
        movingItem = item[fromItem];
        item[fromItem] = -1;
        invalidate(fromItem,0);
        int x1, y1, x2, y2;
        if (toItem == 0) {
                x2 = leftOffset + ((barWidth+barGap)*12);
                y2 = memTemp;
        }
        else if (toItem < 11){
                x2 = leftOffset + (toItem-1)*(barWidth+barGap);
                y2 = firstRow_y;
        }
        else{
                x2 = leftOffset + (toItem-11)*(barWidth+barGap);
                y2 = memTemp;
        }
        if (fromItem == 0) {
                x1 = leftOffset + ((barWidth+barGap)*12);
                y1 = memTemp;
        }
        else if (fromItem < 11) {
                 x1 = leftOffset + (fromItem-1)*(barWidth+barGap);
                 y1 = firstRow_y;
        }
        else {
                x1 = leftOffset + (fromItem-11)*(barWidth+barGap);
                y1 = memTemp;
        }
        doWait(slider.getValue());
        item[toItem] = movingItem;
        movingItem = -1;
        invalidate(toItem,0);
```

```
            repaint();
    }
boolean greaterThan(int itemA, int itemB){
            comparisonCt++;
            comparisons.setText(String.valueOf(comparisonCt));
            return (item[itemA] > item[itemB]);
}
void swapItems (int a, int b) {
            copyItem(0, a);
            if (getState() == STARTING)
                        return;
            copyItem(a, b);
            if (getState() == STARTING)
                        return;
            copyItem(b, 0);
            swap++;
            swaps.setText(String.valueOf(swap));
}

synchronized void scriptSetup(){
            switch (method)  {
                        case 1: {
                                    j = 10;
                                    i = 1;
                                    putTemp(true);
                                    break;
                        }
                        case 2: {
                                    j = 0;
                                    putTemp(true);
                                    break;
                        }
                        case 3: {
                                    heapSize=10;
                                    node = heapSize/2;
                                    putTemp(true);
                                    break;
```

```
                }
        }
}
synchronized void scriptStep() {
        if (!valid) {
                scriptSetup();
                valid = true;
                done = false;
                putItem1(-5);
                putItem2(-5);
                putItem3(-5);
                putLeft(-5);
                putRight(-5);
                putLast(-5);
                return;
        }
        switch (method) {
                case 1:
                        if (i==j){
                                if (j==2){
                                        done = true;
                                        putTemp(false);
                                        item[1] = 100+item[1];
                                        itemChanged(1);
                                        putItem1(-1);
                                        putItem2(-1);
                                        sortFinished.setText("Η ταξινόμηση τελείωσε !!!");
                                }
                                else {
                                        putItem2(-1);
                                        putItem1(-1);
                                        j = j - 1;
                                        i = 1;
                                }
                        }
                        else {
                                i2Loc = i + 1 ;
```

```
                putItem1(i);
                putItem2(i2Loc);
                if (greaterThan(i, i + 1)){
                        swapItems(i, i + 1);
                }
                else{
                        putItem2(i2Loc);
                        putItem1(i);
                }
                i = i + 1;
                if (i==j){
                        putItem2(-1);
                        putItem1(-1);
                        item[j] = 100+item[j];
                        itemChanged(j);
                }
        }
break;
case 2:
        if (j==0){
                copyItem(0, 2);
                j = 2;
                i = 1;
                iLoc = j;
                putItem1(iLoc);
                putItem3(-3);
                putTemp(true);
        }
        else if (j==11) {
                putItem1(-3);
                putItem3(-3);
                for (int i = 1; i <= 10; i++)
                        item[i] += 100;
                setChangedAll();
                sortFinished.setText("Η ταξινόμηση τελείωσε !!!");
                done = true;
                putTemp(false);
```

```
            }
            else if (i==0){
                        copyItem(1, 0);
                        i = -1;
            }
            else if (i == -1){
                        j = j + 1;
                        iLoc = j;
                        putItem1(iLoc);
                        i = -2;
                        putItem3(-3);
            }
            else if (i == -2) {
                        i3Loc = j;
                        putItem3(i3Loc);
                        putItem1(-3);
                        copyItem(0, j);
                        i = j - 1;
                        putItem3(-3);
            }
            else if (greaterThan(i, 0)){
                        i3Loc = i;
                        putItem3(i3Loc);
                        putItem1(iLoc);
                        copyItem(i + 1, i);
                        i = i - 1;
                        swap++;
                        swaps.setText(String.valueOf(swap));
            }
            else {
                        iLoc = i+1;
                        putItem3(-3);
                        putItem1(iLoc);
                        copyItem(i + 1, 0);
                        i = -1;
            }
break;
```

```
case 3:
        if (node<=5 && node>0) {
                i = node ;
                 if (node==5){
                        left = 2 * i;
                        putItem1(i);   putLeft(left);putRight(-4);
                        if (greaterThan(left,i)) {
                                putMax(left);  largest = left;
                        }
                        else {
                                putMax(i); largest = i;
                        }
                        if (largest != i) {
                                swapItems( i, largest);
                                putMax(-4);
                        }
                        node --;
                        return;
                }
                else if (node==4){
                        left = 2 * i ;
                        right = 2 * i +1 ;
                        putLeft(left);   putRight(right);  putItem1(i);
                        if (greaterThan(left,i)) {
                                putMax(left);  largest = left;
                        }
                        else {
                        putMax(i); largest = i;
                        }
                        if (greaterThan(right,largest)){
                                putMax(right); largest = right;
                        }
                        if (largest != i) {
                                swapItems( i, largest);
                                putMax(-4);
                        }
                        node --;
```

```
            return;
}
else if (node==3){
            left = 2 * i ;
            right = 2 * i + 1;
            putLeft(left); putRight(right); putItem1(i);
            if (greaterThan(left,i)){
                        putMax(left); largest = left;
            }
            else {
                        putMax(i);largest = i;
            }
            if (greaterThan(right,largest)) {
                        putMax(right);  largest = right;
            }
            if (largest != i) {
                        swapItems( i, largest);
            }
            node --;
            return;
}
else if (node==2){
            left = 2 * i ;
            right = 2 * i + 1;
            putLeft(left); putRight(right); putItem1(i);
            if (greaterThan(left,i)) {
                        putMax(left); largest = left;
            }
            else {
                        putMax(i); largest = i;
            }
            if (greaterThan(right,largest)) {
                        putMax(right);largest = right;
            }
             if (largest != i) {
                        swapItems( i, largest);
                        putMax(-4);
```

```
                    i = largest ;
                    if (largest==4){
                            left = 2 * i;
                            right = 2 * i + 1;
                            putLeft(left); putRight(right); putItem1(i);
                            if (greaterThan(left,i)) {
                                    putMax(left);  largest = left;
                            }
                            else{
                                    putMax(i);   largest = i;
                            }
                            if (greaterThan(right,largest)) {
                                    putMax(right);largest = right;
                            }
                            if (largest != i) {
                                    swapItems( i, largest);
                                    putMax(-4);
                            }
                    }
                    else if (largest == 5){
                            left = 2 * i;
                            putLeft(left); putItem1(i);putRight(-4);
                            if (greaterThan(left,i)) {
                                    putMax(left);  largest = left;
                            }
                            else{
                                    putMax(i);  largest = i;
                            }
                            if (largest != i) {
                                    swapItems( i, largest);
                                    putMax(-4);
                            }
                    }
            }
    }
    node --;
    return;
}
```

```
else if (node==1){
        left = 2 * i ;
        right = 2 * i + 1;
        putLeft(left); putRight(right);  putItem1(i);
        if (greaterThan(left,i)) {
                putMax(left);   largest = left;
        }
        else {
                putMax(i);largest = i;
        }
        if (greaterThan(right,largest)) {
                putMax(right);  largest = right;
        }
         if (largest != i) {
                swapItems( i, largest);
                i = largest ;
                if (largest==2){
                        left = 2 * i;
                        right = 2 * i + 1;
                        putLeft(left);   putRight(right);  putItem1(i);
                if (greaterThan(left,i)) {
                        putMax(left); largest = left;
                }
                else{
                        putMax(i);  largest = i;
                }
                if (greaterThan(right,largest)) {
                        putMax(right);  largest = right;
                }
                if (largest != i) {
                        swapItems( i, largest);
                        putMax(-4);
                        i = largest ;
                        if (largest==4){
                                        left = 2 * i;
                                        right = 2 * i + 1;
                                        putLeft(left);  putRight(right);  putItem1(i);
```

```
                                        if (greaterThan(left,i)) {
                                        putMax(left);  largest = left;
                                        }
                                        else{
                                                putMax(i);  largest = i;
                                        }
                                        if (greaterThan(right,largest)) {
                                        putMax(right); largest = right;
                                        }
                                        if (largest != i) {
                                        swapItems( i, largest);
                                        }
                                }
                                else if (largest==5){
                                        left = 2 * i;
                                        putLeft(left);  putItem1(i); putRight(-4);
                                        if (greaterThan(left,i)) {
                                        putMax(left); largest = left;
                                        }
                                        else{
                                        putMax(i);  largest = i;
                                        }
                                        if (largest != i) {
                                        swapItems( i, largest);
                                        putMax(-4);
                                        }
                                }
                        }
        }
else if (largest==3){
        left = 2 * i;
        right = 2 * i + 1;
        putLeft(left);  putRight(right);  putItem1(i);
        if (greaterThan(left,i)) {
                putMax(left);  largest = left;
        }
        else{
```

```
                    putMax(i);  largest = i;
                }
                if (greaterThan(right,largest)) {
                    putMax(right); largest = right;
                }
                if (largest != i) {
                    swapItems( i, largest);
                }
            }
            }
          node --;
          return;
        }
  }


item[heapSize] = 100+item[heapSize];
        itemChanged(heapSize);
        putLast(-4);
        putLeft(-4);
        putRight(-4);
        putItem1(-4);
        done = true;
        putTemp(false);
        sortFinished.setText("Η ταξινόμηση τελείωσε !!!");
     break;
   }//switch
 }//script setup
}//algosort
```

Κλάση Slider.java

```
public class Slider extends Panel implements Adjustable{
   Scrollbar scrollbar;
   Label valueLabel, titleLabel;
   Font myFont1 = new Font("Serif",Font.BOLD,18);
```

```
Font myFont2 = new Font("Serif",Font.BOLD,16);

public Slider(String title, int initialValue, int visible, int max, int min){
        String initialValueStr = Integer.toString(initialValue);
        valueLabel = new Label("Fast                Slow", Label.CENTER);
        valueLabel.setForeground(new Color(143,188,143));
        valueLabel.setFont(myFont2);
        titleLabel = new Label(title, Label.CENTER);
        titleLabel.setFont(myFont1);
        titleLabel.setForeground(new Color(209,129,105));
        scrollbar  = new Scrollbar(Scrollbar.HORIZONTAL, initialValue, visible, max, min);
        scrollbar.setSize(10, 10);
        scrollbar.setBackground(Color.gray);
        setLayout(new BorderLayout());
        add(valueLabel, "South");
        add(scrollbar, "Center");
        add(titleLabel, "North");
}
public void addAdjustmentListener(AdjustmentListener l) {
  scrollbar.addAdjustmentListener(l);
}
public void removeAdjustmentListener(AdjustmentListener l) {
  scrollbar.removeAdjustmentListener(l);
}
public int getOrientation() {
  return scrollbar.getOrientation();
}
public void setOrientation(int orient) {
  scrollbar.setOrientation(orient);
}
public int getValue() {
  return scrollbar.getValue();
}
public int getVisibleAmount() {
  return scrollbar.getVisibleAmount();
}
public int getMinimum() {
```

```
    return scrollbar.getMinimum();
  }
  public int getMaximum() {
    return scrollbar.getMaximum();
  }
  public int getUnitIncrement() {
    return scrollbar.getUnitIncrement();
  }
  public int getBlockIncrement() {
    return scrollbar.getBlockIncrement();
  }
  public void setValue(int value) {
    scrollbar.setValue(value);
    valueLabel.setText(Integer.toString(value));
  }
  public void setVisibleAmount(int value) {
    scrollbar.setVisibleAmount(value);
  }
  public void setMinimum(int min) {
    scrollbar.setMinimum(min);
  }
  public void setMaximum(int max) {
    scrollbar.setMaximum(max);
  }
  public void setUnitIncrement(int inc){
    scrollbar.setUnitIncrement(inc);
  }
  public void setBlockIncrement(int inc){
    scrollbar.setBlockIncrement(inc);
  }
}
```

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] A. Korhonen. World wide web (www) tietorakenteiden ja algoritmien tie-tokoneavusteisessa opetuksessa. Master's thesis, Helsinki University of Tech-nology, 1997.

[2] Ainsworth, S. (1999). The functions of multiple representations. *Computers & Education*, 33,131-152.

[3] Ainsworth, S., & Van Labeke, N. (2004). Multiple forms of dynamic representation. *Learningand Instruction*, 14(3), 235-357.

[4] Anderson, J.R. (1995). *Cognitive Psychology and Its Implications* (Fourth Edition). New York:Freeman.Ausserhofer, A. (2000). Dynamic and Parameterised

[5] Animation of Baecker, R.M. (1998). Sorting Out Sorting: A Case Study of Software Visualization forTeaching Computer Science. In M. Brown, J. Domingue, B. Price, and J. Stasko (Eds.),"Software Visualization: Programming as a Multimedia Experience". The MIT Press,Cambridge, MA, pp. 369-382.

[6] Baecker, R.M. (1998). Sorting Out Sorting: A Case Study of Software Visualization forTeaching Computer Science. In M. Brown, J. Domingue, B. Price, and J. Stasko (Eds.),"Software Visualization: Programming as a Multimedia Experience". The MIT Press,Cambridge, MA, pp. 369-382.

[7] Baloukas, T. (2012), JAVENGA: JAva-based Visualization Environment for Network and Graph Algorithms. Comput. Appl. Eng. Educ., 20: 255–268. doi: 10.1002/cae.20392

[8] Bazik, J., Tamassia, R., Reiss, S., & Van Dam, A. (1998). Software visualization in teaching at Brown University. In M. Brown, J. Domingue, B. Price, and J. Stasko (Eds.) "Software Visualization: Programming as a Multimedia Experience". The MIT Press, Cambridge, MA, pp.383-398.

[9] Brown, M. H., & Hershberger, J. (1998). Fundamental Techniques for Algorithm AnimationDisplays. In M. Brown, J. Domingue, B. Price, and J. Stasko (Eds.)

"Software Visualization:Programming as a Multimedia Experience". The MIT Press, Cambridge, MA, pp. 81-101.

[10] Byrne, M. D., Catrambone, R., & Stasko, J.T. (1999). Evaluating Animations as student aids in learning computer algorithms, *Computers & Education*, 33, 253-278.

[11] Chi, M. T. H., Feltovich, P. J., & Glaser, R. (1981). Categorization and representation of physics problems by experts and novices. *Cognitive Science*, 5, 121-152.

[12] Colaso V, Kamal A, Saraiya P, North C, McCrickard S, Shaffer CA. Learning and Retention in Data Structures: A Comparison of Visualization, Text, and Combined Methods. In: World Conference on Educational Multimedia/Hypermedia and Educational Telecommunications (ED-MEDIA 2002).; 2002.

[13] Demetriadis, S., Triantafillou, E., & Pombortsis, A. (2003). A phenomenographic study of students' attitudes toward the use of multiple media for learning, ACM SIGCSE Bulletin, *Proceedings of ITiCSE 2003, 8th Annual Conference on Innovation and Technology in Computer Science Education*, Thessaloniki, University of Macedonia, 35 (3), 183-187.

[14] Douglas, S., Hundhausen, C., & McKeown, D. (1996). Exploring human visualization of algorithms. In *Proceedings of Graphics Interface '96* (pp. 9-16).

[15] Fouh E, Akbar M, Shaffer CA. The Role of Visualization in Computer Science Education. Computers in the Schools. 2012;29(1-2):95-117.

[16] G. Rößling. Overview of the Animation Repository. http://www.animal.ahrgr.de/available-animations/, Accessed November 9, 2005.

[17] G. Rößling and B. Freisleben. Animal: A System for Supporting Multiple Roles in Algorithm Animation. *Journal of Visual Languages and Computing*, 13(3):341–354, 2002.

[18] Kann, C., Lindeman, R.W. & Heller R. (1997). Integrating algorithm animation into a learning environment, *Computers & Education*, 28(4), 223-228.

[19] Kehoe, C. M., & Stasko, J. T. (1996). Using animations to learn about algorithms: An ethnographic case study. (Technical Report No. GIT-GVU-96-20). Atlanta, GA: Georgia Institute of Technology.

[20] Kehoe, C., Stasko, J., & Taylor, A. (2001). Rethinking the evaluation of algorithm animations as learning aids: an observational study, *Int. J. Human-Computer Studies*, 54, 265-284. (Available online at http://www.idealibrary.com)

[21] Kevin Wayne – Robert Sedgewick. Introduction to Programming in JAVA. An interdisciplinary Approach. Lawrence, A. W., Badre, A. M., & Stasko, J. T. (1994). Empirically evaluating the use of animations to teach algorithms. In *Proceedings of the 1994 IEEE Symposium on Visual Languages* (pp. 48-54).

[22] Malmi, L., V. Karavirta, A. Korhonen, J. Nikander, O. Seppälä, P. Silvasti (2004). Visual Algorithm Simulation Exercise System with Automatic Assessment: TRAKLA2. (Available online at http://www.cs.hut.fi/Research/SVG/publications/infoinedu.pdf)

[23] Mayer, R. E. & Anderson, R. B. (1991). Animations need narrations: An experimental test of a dual-coding hypothesis. *Journal of Educational Psychology*, 83, 484-490.

[24] Mayer, R.E. (2003). The promise of multimedia learning: using the same instructional design methods across different media. *Learning and Instruction*, 13, 125-139.

[25] Park, O., & Gittelman, S. S. (1992). Selective use of animation and feedback in computer-based instruction. *Educational Technology Research & Development*, 40(4), 27-38.

[26] Price, B.A., Baecker, R.M., & Small, I.S (1998). A Principled Taxonomy of SoftwareVisualization. *Journal of Visual Languages and Computing*, 4(3), 211-266.

[27] Purvi Saraiya, Clifford A. Shaffer, D. Scott Mccrickard, Chris North (2004). “Effective features of algorithm visualizations”. ACM SIGCSE V.36 Issue 1 Pages 382-386

[28] S. Diehl. Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software. Springer-Verlag New York, Inc., 2007.

[29] Saraiya P, Shaffer CA, McCrickard DS, North C. Effective Features of Algorithm Visualizations. In: SIGCSE '04: Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education. Norfolk, VA: ACM; 2004. p. 382-6.

[30] Shaffer CA, Karavirta V, Korhonen A, Naps TL. OpenDSA: beginning a community active-eBook project. In: Proceedings of the 11th Koli Calling International Conference on Computing Education Research. Koli National Park, Finland: ACM; 2011. p. 112-7. (Koli Calling '11).

[31] Shaffer CA, Akbar M, Alon AJ, Stewart M, Edwards SH. Getting algorithm visualizations into the classroom. In: Proceedings of the 42nd ACM Technical Symposium on Computer Science Education - SIGCSE '11. Dallas, TX, USA: ACM Press; 2011. 129.

[32] Shaffer CA, Naps TL, Fouh E. Truly Interactive Textbooks Shaffer CA, Naps TL, Fouh E. Truly Interactive Textbooks for Computer Science Education. In: Proceedings of the Sixth Program Visualization Workshop. Darmstadt, Germany; 2011. p. 97-103.

[33] Shaffer CA, Naps TL, Rodger SH, Edwards SH. Building an online educational community for algorithm visualization. In: Proceedings of the 41st ACM technical symposium on Computer science education - SIGCSE '10. Milwaukee, Wisconsin, USA: ACM Press; 2010. 475.

[34] Shaffer CA, Cooper ML, Alon AJ, Akbar M, Stewart M, Ponce S, et al. Algorithm Visualization: The State of the Field. ACM Transactions on Computing Education. 2010;10(3):1-22.

[35] Shaffer CA, Cooper M, Edwards SH. Algorithm visualization: a report on the state of the field. ACM SIGCSE Bulletin. 2007;39(1):150-4.

[36] Shaffer CA, Heath LS, Yang J. Using the Swan data structure visualization system for computer science education. In: SIGCSE '96: Proceedings of the twenty-seventh SIGCSE technical symposium on Computer science education. New York, NY, USA: ACM Press; 1996. p. 140-4.

[37] Stasko, J. (1990) "TANGO: A framework and system for algorithm animation", *Computer*,23(9), 27-39.

[38] Stasko, J. (1998). Smooth continuous animation for portraying algorithms and processes. In M.Brown, J. Domingue, B. Price, and J. Stasko (Eds.), "Software Visualization: Programming asa Multimedia Experience". The MIT Press, Cambridge, MA, pp. 103-118.

[39] Stasko, J., & Lawrence, A. (1998). Empirically assessing algorithm animations as learning aids.

[40] Stasko, J., Eugene Zhang. Focus+context display and navigation techniques for enhancing radial, space-filling hierarchy visualizations. In *Proceedings of the Symposium on Information Visualization (Info- Vis'00), Salt Lake City, UT*, pages 57–65, Washington, DC, 2000. IEEE Computer Society Press.

[41] Tarantula. http://www.cc.gatech.edu/aristotle/Tools/tarantula.

[42] In M. Brown, J. Domingue, B. Price, and J. Stasko (Eds.) "Software Visualization: Programming as a Multimedia Experience". The MIT Press, Cambridge, MA, pp. 417-438.

[43] J. Hyvönen and L. Malmi. TRAKLA – a system for teaching algorithms usingemail and a graphical editor. In Proceedings of HYPERMEDIA in Vaasa , pages141–147, 1993.

[44] Robert Lafore. Δομές δεδομένων και Αλγόριθμοι στη JAVA. Δεύτερη έκδοση.

[45] Robert Sedgewick. Αλγόριθμοι σε JAVA.

[46] Kevin Wayne – Robert Sedgewick. Introduction to Programming in JAVA. An interdisciplinary Approach.

[47] Robert Sedgewick – Philippe Flazolet. An introduction to the analysis of Algorithms. Δεύτερη έκδοση.

[48] Robert Sedgewick. Algorithms. Τέταρτη έκδοση.

[49] Wikipedia the free encyclopedia, Ταξινόμηση Φυσαλίδας
http://el.wikipedia.org/wiki/Ταξινόμηση_φυσαλίδας
http://en.wikipedia.org/wiki/Bubble_sort

[50] Wikipedia the free encyclopedia, Ταξινόμηση Εισαγωγής
http://en.wikipedia.org/wiki/Insertion_sort

[51] Wikipedia the free encyclopedia, Ταξινόμηση Σωρού
http://en.wikipedia.org/wiki/Heap_sort
http://en.wikipedia.org/wiki/Heap_(data_structure)

[52] Yang J, Shaffer CA, Heath LS. SWAN: A Student-Controllable Data Structure Visualization System. In: Proceedings of Graph Drawing '95. Springer Lecture Notes in Computer Science 1027; 1995. p. 520-3.