Τεχνολογικό Εκπαιδευτικό Ίδρυμα Κεντρικής Μακεδονίας
Σχολή Τεχνολογικών Εφαρμογών
Τμήμα Μηχανικών Πληροφορικής ΤΕ

Πτυχιακή Εργασία

Τσακμάκης Χρήστος

Ανάπτυξη εφαρμογής για την υλοποίηση αλγορίθμου ταξινόμησης σε οπτικοποιημένο περιβάλλον

ΠΕΡΙΕΧΟΜΕΝΑ

Σύντομή Περίληψη

Η ταξινόμηση είναι μία από τις πιο σημαντικές διαδικασίες που εκτελούνται σε ένα υπολογιστή και αυτό οφείλεται στο γεγονός ότι τα ταξινομημένα δεδομένα είναι πιο εύκολο να τα διαχειριστεί κανείς σε σύγκριση με τα τυχαίως διατεταγμένα δεδομένα. Επιπρόσθετα, η ταξινόμηση είναι ιδιαίτερης σημασίας και για τον λόγο ότι έχει στενή σχέση με το πρόβλημα της δρομολόγησης δεδομένων σε επεξεργαστές, που είναι ένα θέμα ιδιαίτερης βαρύτητας για πολλούς παράλληλους αλγορίθμους.

Σε αυτή την εργασία θα παρουσιαστούν και θα περιγραφούν αρκετοί αλγόριθμοι ταξινόμησης και επιπλέον θα συγκριθούν πειραματικά ανά κατηγορίες με τη βοήθεια ενός γραφικού περιβάλλοντος που υλοποιήθηκε για τις ανάγκες αυτής της εργασίας. Έτσι θα καταλήξουμε σε χρήσιμα συμπεράσματα όπως το ποιος είναι ο πιο γρήγορος αλγόριθμος ταξινόμησης ή πως μέσα από μία ποικιλία αλγορίθμων ταξινόμησης θα επιλέγουμε τον καταλληλότερο.

Εισαγωγή

Η ταξινόμηση μπορεί να χαρακτηριστεί ως η διαδικασία της διευθέτησης μίας μη ταξινομημένης συλλογής από στοιχεία σε μία αύξουσα (ή φθίνουσα) ακολουθία. Ειδικότερα, ας θεωρήσουμε μία ακολουθία n στοιχείων S=<a1, a2,…,an> με ακαθόριστη σειρά. Η ταξινόμηση θα μετατρέψει την ακολουθία S σε μία μονοτονικά αύξουσα ακολουθία S=<α1,α2,....,αn> έτσι ώστε αi<αj για 1≤ i ≤ j ≤ n και η ακολουθία S’ να είναι μία μετάθεση του.

Σκοπός της ταξινόμησης είναι η διευκόλυνση της αναζήτησης των εγγραφών του ταξινομηθέντος συνόλου. Για παράδειγμα σε ένα μη ταξινομημένο πίνακα με n εγγραφές η αναζήτηση ενός κλειδιού μπορεί να γίνει σειριακά εκτελώντας O(n) συγκρίσεις, ενώ σε ταξινομημένο πίνακα η δυαδική αναζήτηση και η αναζήτηση παρεμβολής εκτελούν συγκρίσεις της τάξης O(logn) και O(loglogn) αντίστοιχα. Η χρησιμότητα της ταξινόμησης

αποδεικνύεται στην πράξη σε περιπτώσεις αναζήτησης αριθμητικών ή αλφαριθμητικών δεδομένων όπως σε βιβλιοθηκονομικά συστήματα λεξικά, τηλεφωνικούς καταλόγους και γενικά όπου γίνεται αναζήτηση αποθηκευμένων αντικειμένων.

Πέρα όμως από την τεράστια πρακτική σημασία της ταξινόμησης το θέμα παρουσιάζει και πολύ μεγάλο θεωρητικό ενδιαφέρον γιατί έχει αναπτυχθεί πληθώρα αλγορίθμων ταξινόμησης στην συνέχεια της εργασίας θα περιγραφούν πολλοί τέτοιοι αλγόριθμοι ταξινόμησης.

Περιγραφή αλγορίθμων ταξινόμησης

Εισαγωγή στους Αλγορίθμους ταξινόμησης

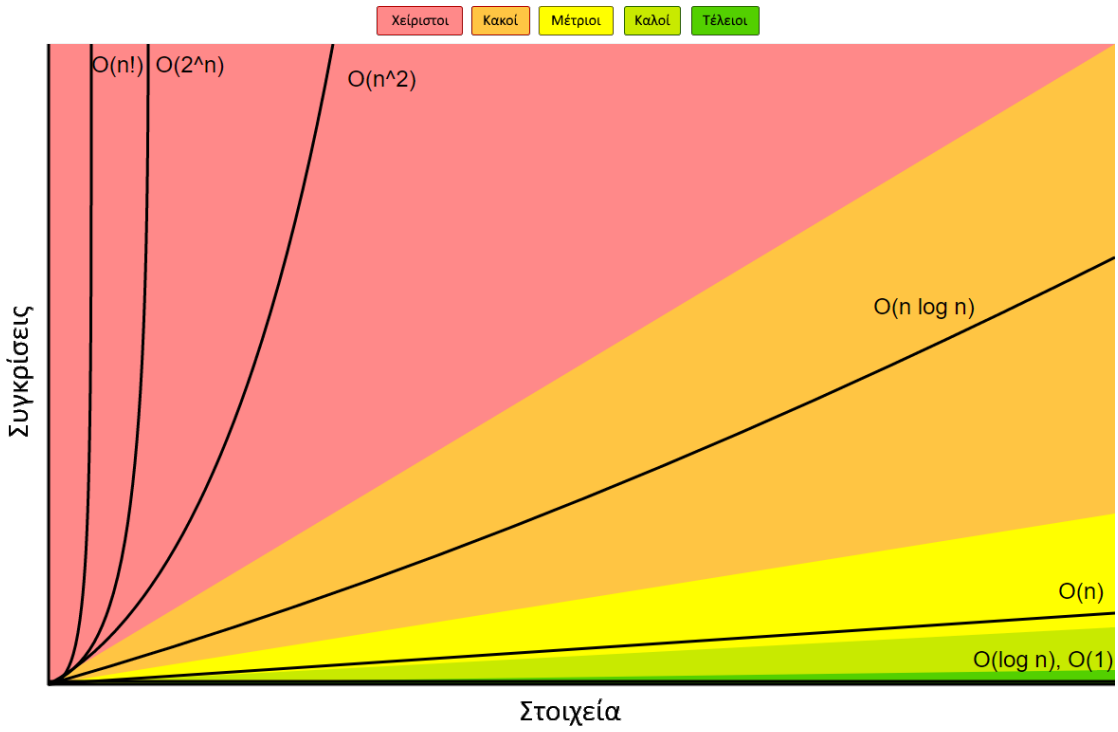
Οι αλγόριθμοι ταξινόμησης είναι αλγόριθμοι που τοποθετούν τα στοιχεία μίας λίστας με συγκεκριμένη σειρά, από τις οποίες οι πιο γνωστές είναι η αριθμητική και η λεξικογραφική σειρά. Όσον αφορά την ταξινομημένη λίστα που προκύπτει πρέπει να τηρούνται δύο κανόνες: Τα στοιχεία της λίστας πρέπει να είναι τοποθετημένα σε αύξουσα σειρά. Το αποτέλεσμα να περιέχει όλα τα στοιχεία της αρχικής λίστας, μόνο που είναι σε διαφορετική σειρά.

Οι αλγόριθμοι ταξινόμησης που χρησιμοποιούνται μέχρι σήμερα στην επιστήμη των υπολογιστών μπορούν να ομαδοποιηθούν σε κατηγορίες ανάλογα με τα χαρακτηριστικά τους. Αυτά τα χαρακτηριστικά που ομαδοποιούν τους αλγορίθμους ταξινόμησης είναι:

Υπολογιστική πολυπλοκότητα των αλγορίθμων ταξινόμησης

Το πιο βασικό κριτήριο της επίδοσης ενός αλγορίθμου είναι ο αριθμός C που μετρά τις απαιτούμενες συγκρίσεις κλειδιών (key comparisons) που εκτελούνται ώστε να γίνει η ταξινόμηση. Οι μέθοδοι ταξινόμησης χωρίζονται με βάση την επίδοσή τους σε δυο μεγάλες κατηγορίες: Στις μεθόδους με επίδοση τάξης O(n2), που ονομάζονται ευθείες(straight) μέθοδοι Στις μεθόδους με επίδοση τάξης O(nlogn).

Πίνακας Πολυπλοκότητας Αλγορίθμων



Παρακάτω παρουσιάζεται ένας πίνακας με τις χρονικές αποδόσεις μερικών αλγορίθμων ταξινόμησης:

Αλγόριθμος	*Πολυπλοκότητα Χρόνου*			*Πολυπλοκότητα Χώρου*
	Καλύτερος	**Μέσος**	**Χειρότερος**	**Χειρότερος**
Quicksort	O(nlog(n))	O(nlog(n))	O(n^2)	O(log(n))
Mergesort	O(nlog(n))	O(nlog(n))	O(nlog(n))	O(n)
Timsort	O(n)	O(nlog(n))	O(nlog(n))	O(n)
Heapsort	O(nlog(n))	O(nlog(n))	O(nlog(n))	O(1)
Bubble Sort	O(n)	O(n^2)	O(n^2)	O(1)
Insertion Sort	O(n)	O(n^2)	O(n^2)	O(1)
Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(1)
Tree Sort	O(nlog(n))	O(nlog(n))	O(n^2)	O(n)
Shell Sort	O(nlog(n))	O(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	O(n+k)	O(n+k)	O(n^2)	O(n)
Radix Sort	O(nk)	O(nk)	O(nk)	O(n+k)
Counting Sort	O(n+k)	O(n+k)	O(n+k)	O(k)
Cubesort	O(n)	O(nlog(n))	O(nlog(n))	O(n)

Υπολογιστική πολυπλοκότητα των ανταλλαγών
Υπάρχουν μέθοδοι ταξινόμησης που για την ταξινόμηση ενός πίνακα δεν χρησιμοποιούν κάποιο βοηθητικό πίνακα ίδιου τύπου αλλά εκτελούν την ταξινόμηση κάνοντας χρήση των ίδιων θέσεων (in place) του πίνακα. Έτσισυχνότατα απαιτείται να γίνει ανταλλαγή (swap) του περιεχομένου δύο θέσεων του πίνακα. Αυτές οι μέθοδοι ταξινόμησης χωρίζονται σε τρεις μεγάλες κατηγορίες: Ταξινομήσεις εισαγωγής Ταξινομήσεις ανταλλαγής ή μετάθεσης Ταξινομήσεις επιλογής Ενώ οι μέθοδοι που δεν χρησιμοποιούν τις ίδιες θέσεις χωρίζονται στις παρακάτω δύο κατηγορίες:

- Ταξινομήσεις συγχώνευσης
- Ταξινομήσεις κατανομής

Πολυπλοκότητα χώρου (μνήμης) - Space Complexity
Είναι η σχέση του μεγέθους των δεδομένων που εισάγονται στον αλγόριθμο και των δεδομένων που χρησιμοποιεί αυτός επιπλέον για να εκτελεστεί. Όταν για παράδειγμα είναι O(1) εννοούμε ότι είναι σταθερή ανεξάρτητα της εισόδου. Όταν είναι O(n) εννοούμε ότι για είσοδο n στοιχείων, χρειάζεται n επιπλέον στοιχεία για να δουλέψει ο αλγόριθμος.

Αναδρομή
Αναδρομή στα μαθηματικά και την επιστήμη υπολογιστών είναι μια μέθοδος για τον ορισμό συναρτήσεων κατά την οποία η οριζόμενη συνάρτηση εμπεριέχει κλήση ή κλήσεις στον ίδιο της τον εαυτό. Μερικοί αλγόριθμοι ταξινόμησης είναι αναδρομικοί ενώ άλλοι αλγόριθμοι δεν είναι.

Ευστάθεια
Μία μέθοδος λέγεται ευσταθής αν η σχετική διάταξη των στοιχείων με ίσα κλειδιά παραμένει αμετάβλητη με την εφαρμογή της μεθόδου. Έτσι μπορούμε

να κάνουμε ένα διαχωρισμό των μεθόδων που είναι ευσταθείς και των μεθόδων που είναι ασταθείς, δηλαδή η σχετική διάταξη των στοιχείων με ίσα κλειδιά μεταβάλλεται μετά την εφαρμογή της μεθόδου.

Συγκρίσεις
Υπάρχουν αλγόριθμοι που βασίζονται στις συγκρίσεις και αλγόριθμοι που δεν χρησιμοποιούν καθόλου συγκρίσεις όπως ο Radixsort.

Προσαρμοστικότητα
Μπορούμε επίσης να ομαδοποιήσουμε τους αλγορίθμους ταξινόμησης από το αν είναι προσαρμοστικοί ή όχι. Προσαρμοστικοί (adaptive) είναι οι αλγόριθμοι ταξινόμησης που βελτιώνουν την επίδοση τους όταν τα στοιχεία στην είσοδο πριν την ταξινόμηση είναι σχεδόν ταξινομημένα.

ΑΛΓΟΡΙΘΜΟΙ ΤΑΞΙΝΟΜΗΣΗΣ	
Ταξινομήσεις ανταλλαγής ή μετάθεσης	Bubble sort Cocktail sort Odd-even sort Comb sort Gnome sort Quicksort
Ταξινομήσεις επιλογής	Selection sort Heapsort Smoothsort Cartesian tree sort Tournament sort Cycle sort
Ταξινομήσεις εισαγωγής	Insertion sort Shell sort Tree sort Library sort Patience sorting
Ταξινομήσεις συγχώνευσης	Merge sort Polyphase merge sort Strand sort Timsort
Γενικοί Αλγόριθμοι	Radix sort Bucket sort Counting sort Pigeonhole sort Burstsort Bead sort

Άλλοι	Topological sorting Sorting network Bitonic sorter Batcher odd-even mergesort Pancake sorting
Μη αποδοτικοί αλγόριθμοι	Bogosort Stooge sort

Από τον παραπάνω πίνακα επιλέχθηκαν 15 μέθοδοι ταξινόμησης, οι οποίες και θα αναλυθούν στο πρόγραμμα :

- Bubble Sort
- Cocktail Sort
- Comb Sort
- Insertion Sort
- Selection Sort
- Counting Sort
- Shell Sort
- Heap Sort
- Merge Sort
- Quick Sort
- Gnome Sort
- Bucket Sort
- Cycle Sort
- Odd-Even Sort
- Pidgeon Hole Sort

Ταξινόμηση φυσαλίδας (**Bubble sort**)

Ταξινόμηση φυσαλίδας (bubble sort) είναι το όνομα ενός απλού αλγόριθμου ταξινόμησης. Λειτουργεί συγκρίνοντας βηματικά τα στοιχεία μιας λίστας και εναλλάσσοντας τα ώστε να βρεθούν σε σωστή σειρά. 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. Αν και απλός, ο αλγόριθμος φυσαλίδας είναι πολύ αναποτελεσματικός. Η πολυπλοκότητα καλύτερης περίπτωσης του συγκεκριμένου αλγόριθμου είναι O(n) και της χειρότερης περίπτωσης O(n2), όπου n είναι το μέγεθος της λίστας.

Η ταξινόμηση φυσαλίδας έχει τα εξής χαρακτηριστικά:

- απλή υλοποίηση
- αποτελεσματικός για (πολύ) μικρά σύνολα δεδομένων
- ευσταθής (stable)
- προσαρμοστική (adaptive)
- επιτόπιος (in-place)
- βελτιώνεται η απόδοση του όταν τα στοιχεία είναι ήδη ταξινομημένα
- γενική μέθοδος ανταλλαγή
- πολυπλοκότητα χειρότερης περίπτωσης $O(n^2)$

Cocktail sort (**Shaker sort**)

Ο αλγόριθμος αυτός είναι μια παραλλαγή του αλγόριθμου φυσαλίδας,. Η διαφορά του με τον αλγόριθμο της φυσαλίδας είναι ότι αντί να διασχίζει τη λίστα από την αρχή προς το τέλος, τη διασχίζει εναλλάξ από την αρχή προς το τέλος και μετά από το τέλος προς την αρχή και ούτω καθεξής. Με τη διαφοροποίηση αυτή ο αλγόριθμος αυτός πετυχαίνει ελαφρά βελτιωμένη απόδοση σε σχέση με τον κλασικό αλγόριθμο φυσαλίδας, γιατί ο αλγόριθμος φυσαλίδας διασχίζει τη λίστα προς μία κατεύθυνση και μπορεί να μετακινεί στοιχεία της λίστας ένα βήμα προς τα πίσω σε κάθε επανάληψη. Η θεωρητική πολυπλοκότητα του αλγορίθμου αυτού είναι στη χειρότερη και στη μέση
περίπτωση O(n2), ενώ πλησιάζει το O(n) στην καλύτερη περίπτωση, που συμβαίνει όταν η αρχή της λίστας είναι ταξινομημένη.

Ο Cocktail sort έχει τα εξής χαρακτηριστικά:

- έχει βελτιωμένη απόδοση σε σχέση με τον Bubble sort
- είναι ευσταθής
- επιτόπιος
- Γενική μέθοδος ανταλλαγή

Comb sort

Αυτή η μέθοδος προήλθε από τη ταξινόμηση φυσαλίδας και λειτουργεί σε σταθερό χρόνο με πολυπλοκότητα O(n2), χωρίς επιπλέον μνήμη. Αυτή συγκρίνει τα στοιχεία όχι ένα ένα αλλά σε μεγαλύτερες αποστάσεις με βάση ένα συντελεστή (shrink factor). Η απόσταση μειώνεται σαν γεωμετρική πρόοδος μέχρι το 1. Έτσι εξαλείφονται τα πολύ "άτακτα" στοιχεία και γίνεται γρήγορα η τελική εκκαθάριση-ταξινόμηση. Η μέθοδος θυμίζει λίγο shellsort αλλά είναι πολύ ανώτερη. Μετά από ελέγχους βρέθηκε ότι ο καλύτερος συντελεστής είναι ο 1,247330950103979.

Ο Comp sort έχει τα εξής χαρακτηριστικά:

- μικρός κώδικας
- επιτόπιος
- ασταθής
- Γενική μέθοδος ανταλλαγή

Ταξινόμηση ευθείας εισαγωγής (**Insertion sort**)

Η ταξινόμηση ευθείας εισαγωγής (insertion sort) είναι ένας πολύ απλός αλγόριθμος ταξινόμησης της κατηγορίας των αλγορίθμων ταξινόμησης με συγκρίσεις. Σύμφωνα με την ταξινόμηση εισαγωγής τα στοιχεία χωρίζονται σχηματικά σε μια ακολουθία προορισμού table[1], table[2],..., table[i-1] και σε μια ακολουθία πηγής table[i], ..., table[n]. Σε κάθε βήμα αρχίζοντας με ι=2 και αυξάνοντας διαδοχικά το ι κατά 1, το στοιχείο με δείκτη ι λαμβάνεται και μεταφέρεται στην κατάλληλη θέση της ακολουθίας προορισμού. Έτσι σε κάθε βήμα αυξάνεται η ακολουθία προορισμού κατά ένα και μειώνεται η ακολουθία πηγής κατά ένα.

Η λειτουργία της εισαγωγής είναι, όμως , «ακριβή» γιατί απαιτεί τη μετατόπιση των υπόλοιπων στοιχείων μία θέση αριστερά (shift left). Ο εν λόγω αλγόριθμος έχει πολυπλοκότητα καλύτερης περίπτωσης O(n) και χειρότερης περίπτωσης $O(n^2)$.

Είναι λιγότερο αποτελεσματικός σε μεγάλους πίνακες από ότι άλλοι αλγόριθμοι ταξινόμησης όπως η γρήγορη ταξινόμηση, η ταξινόμηση σωρού, η ταξινόμηση συνένωσης.

Η ταξινόμηση εισαγωγής έχει τα εξής πλεονεκτήματα:

- απλή υλοποίηση
- αποτελεσματικός για (πολύ) μικρά σύνολα δεδομένων
- προσαρμοστικός
- είναι ευσταθής (stable)
- επιτόπιος (in place)
- Online
- Γενική μέθοδος εισαγωγή

Ταξινόμηση ευθείας επιλογής (**Selection sort**)

Ένας άλλος απλός αλγόριθμος ταξινόμησης είναι ο αλγόριθμος επιλογής (selection sort) που έχει βελτιωμένη απόδοση σε σχέση με τον αλγόριθμο φυσαλίδας. Η ταξινόμηση ευθείας επιλογής έχει πολυπλοκότητα Θ(n2) γεγονός που την κάνει αναποτελεσματική σε μεγάλες λίστες .

Ανήκει στην κατηγορία των αλγορίθμων ταξινόμησης που είναι in-place δηλαδή για την ταξινόμηση ενός πίνακα δεν χρησιμοποιούν κάποιο βοηθητικό πίνακα ίδιου τύπου αλλά εκτελούν την ταξινόμηση κάνοντας χρήση των ίδιων θέσεων του πίνακα. Ο αλγόριθμος επιλογής βασίζεται στις ακόλουθες δυο αρχές: i) επιλογή του στοιχείου με το ελάχιστο κλειδί (για αύξουσα σειρά) ii) ανταλλαγή αυτού του στοιχείου με το πρώτο στοιχείο του πίνακα. Αυτές οι λειτουργίες επαναλαμβάνονται για τα υπόλοιπα n-1 στοιχεία μέχρι στο τέλος να απομείνει το μεγαλύτερο στοιχείο.

Η ταξινόμηση επιλογής έχει τα εξής χαρακτηριστικά:

- απλή υλοποίηση
- αποτελεσματικός για (πολύ) μικρά σύνολα δεδομένων
- ασταθής (unstable)
- επιτόπιος
- γενική μέθοδος επιλογή

Counting sort

Αυτή η μέθοδος βασίζεται στο εύρος των τιμών των στοιχείων που θέλουμε να ταξινομήσουμε. Μετράει πόσες φορές εμφανίζεται κάθε τιμή στον πίνακα και στο τέλος απλά τον δημιουργεί από την αρχή ταξινομημένο. Είναι πολύ βολική μέθοδος για πίνακες ακεραίων με μικρό εύρος τιμών. Ο αλγόριθμος που παρουσιάζεται εδώ αφορά ακέραιους και το k στην πολυπλοκότητα είναι η διαφορά μεταξύ μέγιστου και ελάχιστου στοιχείου.

Ο Counting sort έχει τα εξής χαρακτηριστικά:

- πολύ βολική μέθοδος για πίνακες ακεραίων με μικρό εύρος τιμών
- ευσταθης
- μνήμη O(k)
- πολυπλοκότητα μέσης και χειρότερης περίπτωσης O(n+k)

Ταξινόμηση παρεμβολής με φθίνοντα διαστήματα (**Shell sort**)

Η ταξινόμηση shellsort είναι μια απλή επέκταση της ταξινόμησης με εισαγωγή η οποία κερδίζει σε ταχύτητα επιτρέποντας αντιμεταθέσεις μεταξύ στοιχείων τα οποία μπορεί να βρίσκονται μακριά το ένα από το άλλο. Για την εφαρμογή αυτής της μεθόδου ταξινομούνται σε ένα πρώτο πέρασμα τα στοιχεία που απέχουν ένα διάστημα h1 μεταξύ τους. Στη συνέχεια ταξινομούνται όλα τα στοιχεία αυτά που απέχουν μεταξύ τους διάστημα h2. Μετά όλα τα στοιχεία που απέχουν μεταξύ τους διάστημα h3 κ.ο.κ. μέχρι το ht να γίνει ίσο με 1.

Οποιοδήποτε διάστημα είναι καλό αρκεί το τελευταίο να είναι το h=1. Σύμφωνα με τον Knuth, καλές ακολουθίες είναι 1,4,13,40,121... ή 1,4,15,31..... Επίσης έχει αποδειχθεί (Knuth) ότι η ακολουθία διαστημάτων που είναι δυνάμεις του 2 (1,2,4...) δεν είναι η καλύτερη αλλά δεν είναι και λανθασμένη. Στο παράδειγμα που ακολουθεί ορίζω h3=4, h2=2, h1=1.

Ταξινόμηση σωρού (**Heap sort**)

Ένας άλλος γνωστός αλγόριθμος ταξινόμησης είναι ο αλγόριθμος ταξινόμησης σωρού, που αποτελεί μια βελτιωμένη παραλλαγή του αλγόριθμου επιλογής, που περιγράφηκε προηγουμένως. Όπως και ο αλγόριθμος επιλογής, έτσι και αυτός ο αλγόριθμος δουλεύει ψάχνοντας το μεγαλύτερο (ή μικρότερο) στοιχείο της λίστας, τοποθετώντας το στο τέλος (ή στην αρχή) και συνεχίζει με το υπόλοιπο της λίστας. Η διαφορά είναι, όμως, ότι ο αλγόριθμος ταξινόμησης σωρού εκτελεί αυτή τη διαδικασία πιο αποτελεσματικά χρησιμοποιώντας ένα τύπο δεδομένων που ονομάζεται σωρός, που ουσιαστικά είναι ένας ειδικός τύπος δυαδικού δέντρου.

Μόλις τα στοιχεία της λίστας σχηματίσουν το σωρό, η ρίζα του δέντρου είναι το μεγαλύτερο στοιχείο. Τότε αφαιρείται και τοποθετείται στο τέλος της λίστας και σχηματίζεται ξανά το σωρό με αποτέλεσμα η ρίζα του δέντρου να είναι πάλι το μεγαλύτερο στοιχείο. Χρησιμοποιώντας το σωρό για να βρεθεί το μεγαλύτερο στοιχείο της λίστας απαιτείται O(log n) χρόνος αντί για O(n) που χρειάζεται για μια σειριακή σάρωση στον απλό αλγόριθμο επιλογής. Αυτό επιτρέπει στην ταξινόμηση σωρού να εκτελείται σε χρόνο O(n*logn).

Η ταξινόμηση σωρού έχει τα εξής χαρακτηριστικά:

- παραλλαγή της ταξινόμησης επιλογής
- είναι ασταθής (unstable)
- επιτόπιος (με πολυπλοκότητα χειρότερης περίπτωσης O(n*logn))
- γενική μέθοδος επιλογή

Ταξινόμηση συνένωσης (**Merge sort**)

Ο αλγόριθμος συνένωσης (merge sort) έχει ως πλεονέκτημα την ευκολία που παρουσιάζει η συνένωση ήδη ταξινομημένων λιστών σε μια νέα ταξινομημένη λίστα. Αρχίζει συγκρίνοντας κάθε ζευγάρι στοιχείων της λίστας (δηλαδή το πρώτο με το δεύτερο, το τρίτο με το τέταρτο και ούτω καθεξής) και εναλλάσσει τα στοιχεία αν το πρώτο είναι μεγαλύτερο του δεύτερου. Στη συνέχεια συνενώνει τα ζευγάρια των ταξινομημένων λιστών των δύο στοιχείων σε ταξινομημένες λίστες των τεσσάρων στοιχείων, στη συνέχεια των οχτώ και ούτω καθεξής, μέχρι να συνενωθούν δύο λίστες στην τελική ταξινομημένη λίστα.

Ο εν λόγω αλγόριθμος έχει θεωρητική πολυπλοκότητα O(nlogn). Είναι ο μόνος ευσταθής αλγόριθμος μεγάλης ταχύτητας και αυτό είναι πολύ σημαντικό. Η ταξινόμηση συνένωσης είναι δυνατόν να υλοποιηθεί και χωρίς βοηθητικό πίνακα για μνήμη O(1), αλλά τότε καθυστερεί πολύ.

Ο αλγόριθμος ταξινόμησης συνένωσης έχει τα εξής χαρακτηριστικά:

- δεν είναι προσαρμοστικός
- είναι ευσταθής αν υλοποιηθεί προσεκτικά
- είναι ο μόνος ευσταθής αλγόριθμος μεγάλης ταχύτητας
- χρησιμοποιεί O(n) μνήμη
- γενική μέθοδος συγχώνευση

Γρήγορη Ταξινόμηση (**Quick sort**)

Ο αλγόριθμος γρήγορης ταξινόμησης είναι ένα παράδειγμα αλγορίθμου διαίρει και βασίλευε που στηρίζεται σε μια λειτουργία διαμερισμού. Για διαχωρισμό ενός πίνακα, επιλέγουμε ένα στοιχείο, επονομαζόμενο ως βήμα (pivot) ή οδηγός, και μετακινούμε όλα ταμικρότερα στοιχεία πριν το στοιχείο βήμα και τα μεγαλύτερο μετά από αυτό. Η λειτουργία αυτή μπορεί να γίνει αποτελεσματικά σε γραμμικό χρόνο. Στη συνέχεια επαναληπτικά ταξινομούμε τις μικρότερες και μεγαλύτερες υπολίστες.

Ο εν λόγω αλγόριθμος είναι ένας από τους ταχύτερους αλγόριθμους ταξινόμησης και μάλιστα απαιτεί O(logn) μνήμη. Το πιο περίπλοκο θέμα σε αυτόν τον αλγόριθμο είναι η επιλογή του στοιχείου – βήματος. Λανθασμένες επιλογές αυτού του στοιχείου μπορούν να κοστίσουν σε δραματική μείωση της επίδοσης σε O(n2). Αν όμως σε κάθε βήμα διαλέγουμε το μεσαίο στοιχείο της λίστας τότε ο αλγόριθμος εκτελείται σε O(nlogn).

Gnome sort

Αυτή η μέθοδος είναι παρόμοια με την ταξινόμηση εισαγωγής εκτός από το γεγονός ότι τα στοιχειά πριν τοποθετηθούν στη σωστή θέση κάνουν κάποιες ανταλλαγές όπως και στη ταξινόμηση φυσαλίδας. Συγκρίνοντας κάθε στοιχείο με το προηγούμενο και το επόμενο στοιχείο αντιμεταθέτει τα στοιχεία που δεν είναι στη σωστή σειρά.

Ο χρόνος εκτέλεσης είναι O(n2) άλλα βελτιώνεται όταν τα στοιχεία είναι σχεδόν ταξινομημένα σε O(n). Πρακτικά ο αλγόριθμος μπορεί να τρέξει τόσο γρήγορα όσο και ο αλγόριθμος εισαγωγής.

Ο Gnome sort έχει τα εξής χαρακτηριστικά:

- μικρός κώδικας
- επιτόπιος
- ευσταθής
- προσαρμοστικός
- γενική μέθοδος ανταλλαγή

Ταξινόμηση με κάδους (**Bucket sort**)

Ο αλγόριθμος bucket sort έχει κατώτερο μέσο χρόνο από το κάτω φράγμα του Ω(n logn) για ταξινόμηση βασιζόμενη σε συγκρίσεις. Αυτό οφείλεται στο ότι ο αλγόριθμος θεωρεί ότι τα n στοιχεία προς ταξινόμηση κατανέμονται ομοιόμορφα στο διάστημα [a, b). Αυτός ο αλγόριθμος καλείται **bucket sort** και εκτελείται ως εξής: Το διάστημα [α, b) διαιρείται σε m ίσα υπό-διαστήματα που καλούνται ***κάδοι (buckets).*** Κάθε στοιχείο τοποθετείται στο κατάλληλο κάδο. Επειδή τα n στοιχεία κατανέμονται ομοιόμορφα στο διάστημα [a, b), το πλήθος των στοιχείων σε κάθε κάδο είναι περίπου n/m.

Ο Bucket sort έχει τα εξής χαρακτηριστικά:

- ευσταθής
- επιπλέον μνημη O(n*k)
- πολυπλοκότητα χειρότερης περίπτωσης $O(n^2*k)$

Κυκλική ταξινόμηση (**Cycle sort**)

Ο cycle sort είναι ένας in place και ασταθής (unstable) αλγόριθμος ταξινόμησης. Θεωρητικά είναι ο in place αλγόριθμος με τις λιγότερες εγγραφές στον αρχικό πίνακα. Βασίζεται στην ιδέα ότι το ανακάτεμα των στοιχείων που χρειάζεται για να προκύψει η ταξινόμηση γίνεται κυκλικά. Έχει πολυπλοκότητα μέσης και χειρότερη περίπτωσης O(n2).

Ο Cycle sort έχει τα εξής χαρακτηριστικά:

- Θεωρητικά μικρότερο αριθμό εγγραφών από όλους τους in-place αλγόριθμους ταξινόμησης
- επιτόπιος
- ασταθής
- γενική μέθοδος εισαγωγής
- μέση και χειρότερη πολυπλοκότητα $O(n^2)$.

Odd - Even sort

Ο αλγόριθμος **Odd-even sort** ταξινομεί n στοιχεία σε n φάσεις (το n είναι ζυγό), κάθε μία από τις οποίες απαιτεί n/2 συγκρίσεις-εναλλαγές. Ο αλγόριθμος αυτός εναλλάσσεται μεταξύ δύο φάσεων, που καλούνται περιττή και ζυγή φάση. Έστω <a1, a2,…,an> είναι η προς ταξινόμηση ακολουθία. Κατά τη διάρκεια της περιττής φάσης, στοιχεία με περιττούς δείκτες συγκρίνονται με τους δεξιούς γείτονες τους, και όταν απαιτείται εναλλάσσονται. Επομένως, τα ζευγάρια (a1,a2), (a3,a4),…., (an-1,an) **συγκρίνονται** και εναλλάσσονται. Ομοίως, κατά τη διάρκεια της ζυγής φάσης, στοιχεία με δείκτες ζυγούς συγκρίνονται με τους δεξιούς τους γείτονες και όταν απαιτείται εναλλάσσονται. Επομένως, τα ζευγάρια (a2,a3), (a4,a5),..,(an-2,an-1) συγκρίνονται και εναλλάσσονται. Μετά από n φάσεις, η ακολουθία ταξινομείται. Κάθε φάση απαιτεί Θ(n) συγκρίσεις και έχουμε συνολικά n φάσεις.

Ο Oddeven sort έχει τα εξής χαρακτηριστικά:

- απλή υλοποίηση
- ευσταθής
- επιτόπιος
- γενική μέθοδος ανταλλαγή
- πολυπλοκότητα χειρότερης περίπτωσης $O(n^2)$.

Pigeonhole Sort (**Count sort**)

Ο Pigeonhole Sort γνωστός και ως count sort (δεν είναι ίδιος με τον counting sort) είναι ένας αλγόριθμος ταξινόμησης κατάλληλος για ταξινόμηση λιστών όπου ο αριθμός των στοιχείων εισόδου είναι περίπου ίδιος με το εύρος τιμών και των κλειδιών.

Ο count sort λειτούργει ως εξής :

Αρχικά δημιουργείται ένας βοηθητικός πίνακας με μέγεθος k. Στη συνέχεια, στη θέση που έχει την τιμή key από τον αρχικό πίνακα τοποθετούμε όλες τιμές που έχουν αυτό το κλειδί. Έτσι δημιουργείται λίστες με όλες τις τιμές που έχει το κάθε κλειδί. Τέλος τοποθετούμε τα στοιχεία από τον βοηθητικό πίνακα στον αρχικό πίνακα και έτσι πραγματοποιείται η ταξινόμηση.

Ο Count sort έχει τα εξής χαρακτηριστικά:

- κατάλληλος για ταξινόμηση λιστών με n=k
- ευσταθής
- επιπλέον μνήμη $O(2^k)$
- Πολυπλοκότητα $O(n+2^k)$

Visual Studio 2012 Professional

Το Microsoft Visual Studio, όπως προαναφέρθηκε, είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης (IDE), το οποίο αναπτύχτηκε από τη Microsoft για να μπορέσει να χρησιμοποιηθεί για την ανάπτυξη προγραμμάτων στα λειτουργικά συστήματα των Windows, αλλά και για τον σχεδιασμό και την ανάπτυξη διαδικτυακών ιστοσελίδων, εφαρμογών και υπηρεσιών διαδικτύου.

Το Visual Studio χρησιμοποιεί διάφορες πλατφόρμες ανάπτυξης λογισμικού της Microsoft, όπως το Windows API, γνωστό και ως win32, που αξιοποιείται στην ανάπτυξη εφαρμογών, ώστε να τρέχουν με επιτυχία σε όλες τις εκδόσεις των Windows. Το Windows Presentation Foundation, γνωστό και ως Avalon, είναι ένα γραφικό υποσύστημα για την απόδοση των διεπαφών του εκάστοτε χρήστη σε εφαρμογές που έχουν μια βάση δεδομένων αλλά και για τον καθορισμό ή την σύνδεση διαφόρων στοιχείων. Επιπλέον, το Microsoft Silverlight είναι ένα πλαίσιο εφαρμογής για τη σύνταξη και τη λειτουργία διαδικτυακών εφαρμογών, με χαρακτηριστικά και σκοπούς παρόμοια με εκείνα του «Adobe Flash» . Το Silverlight είναι, επίσης, μία από τις δύο πλατφόρμες ανάπτυξης εφαρμογών για Windows Phone, και Windows Store.

Το Visual Studio υποστηρίζει διάφορες γλώσσες προγραμματισμού και επιτρέπει την επεξεργασία του κώδικα και τον εντοπισμό σφαλμάτων. Το ολοκληρωμένο πρόγραμμα εντοπισμού σφαλμάτων λειτουργεί 17 τόσο ως ένα πρόγραμμα εντοπισμού σφαλμάτων επιπέδου πηγής, όσο και ως ένα πρόγραμμα εντοπισμού σφαλμάτων επιπέδου μηχανής. Το Visual Studio περιλαμβάνει ένα πρόγραμμα εντοπισμού σφαλμάτων το ονομαζόμενο «Microsoft Visual Studio Debugger», που λειτουργεί τόσο ως ένα πρόγραμμα εντοπισμού σφαλμάτων επιπέδου πηγής, όσο και ως ένα πρόγραμμα εντοπισμού σφαλμάτων επιπέδου μηχανής. Λειτουργεί σε διαχειριζόμενο κώδικα και μπορεί να χρησιμοποιηθεί για τον εντοπισμό σφαλμάτων σε εφαρμογές, οι οποίες είναι γραμμένες σε οποιαδήποτε γλώσσα προγραμματισμού. Επιπλέον, έχει την δυνατότητα να συμπεριληφθεί σε διεργασίες που εκτελούνται για να μπορέσει να παρακολουθήσει καθώς και να εντοπίσει σφάλματα των συγκεκριμενών διαδικασιών. Το πρόγραμμα εντοπισμού σφαλμάτων στο Visual Studio μπορεί επίσης να δημιουργήσει χωματερές μνήμης.

Στην πληροφορική η χωματερή μνήμης αποτελείται από την πραγματική κατάσταση της εργασίας μνήμης ενός προγράμματος ηλεκτρονικού υπολογιστή σε μια συγκεκριμένη χρονική στιγμή. Συνήθως όταν το πρόγραμμα έχει τερματιστεί ανώμαλα χρησιμοποιούνται συχνά για να βοηθήσουν στη διάγνωση και στον εντοπισμό σφάλματων στα προγράμματα ηλεκτρονικών υπολογιστών. Το πρόγραμμα εντοπισμού σφαλμάτων επιτρέπει τον καθορισμό σημείων διακοπής, που αναγκάζουν την εκτέλεση να διακοπεί προσωρινά σε μια ορισμένη θέση και τα ρολόγια που παρακολουθούν τις τιμές των μεταβλητών, καθώς η εκτέλεση προχωρεί.

Στη 1 Αυγούστου 2012 ανακοινώθηκε επίσημα από την Microsoft η τελική κατασκευή του Visual studio 2012, σε αντίθεση με τις προηγούμενες εκδόσεις.

Το Visual Studio 2012 δεν μπορεί να καταγράψει και να παίξει μακροεντολές καθώς και να αφαιρέσει του συντάκτη «macro». Ένα σημαντικό νέο χαρακτηριστικό είναι η υποστήριξη για WinRT ή χρόνου εκτέλεσης Windows, είναι μια πλατφόρμα-ομοιογενή αρχιτεκτονική της εφαρμογής του λειτουργικό συστήματος των Windows 8. Οι εφαρμογές WinRT υποστηρίζονται εν γένει τόσο σε αρχιτεκτονική για x86, όσο και για την αρχιτεκτονική ARM, καθώς επίσης τρέχει μέσα σε ένα sandboxed περιβάλλον για να επιτρέψει μεγαλύτερη ασφάλεια και σταθερότητα.

Αξίζει να σημειωθεί ότι, στα Windows Phone 8 χρησιμοποιείται μια έκδοση του χρόνου εκτέλεσης Windows για να έχει την δυνατότητα να επιτρέπει την ανάπτυξη εφαρμογών με την χρήση γλωσσών προγραμματισμού σε C # καθώς και σε VB.NET. Ο πηγαίος κώδικας του Visual Studio 2012 αποτελείται από περίπου 50 εκατομμύρια γραμμές κώδικα. Η Microsoft παρέχει, τέλος, μια δωρεάν έκδοση του Visual Studio που ονομάζεται «Express» και η συγκεκριμένη έκδοση είναι διαθέσιμη στους φοιτητές μέσω του προγράμματος της Microsoft dreamspark, χωρίς κανένα κόστος.

Γλώσσα Προγραμματισμού C#

Η Γλώσσα C# είναι μια πολύ ευέλικτη και ισχυρή γλώσσα προγραμματισμού με μια ενδιαφέρουσα ιστορία. Θεωρείται ως μια αντικειμενοστραφή γλώσσα υψηλού επιπέδου, που αναπτύχθηκε από την Microsoft σαν μέρος της πρωτοβουλίας του .NET και είναι ευρέως γνωστή για την ανάπτυξη εφαρμογών λογισμικού.

Η C# αποτελεί εξέλιξη των παλαιότερων και πολύ διαδεδομένων γλωσσών προγραμματισμού των C και C++, καθώς επίσης φέρει και πολλά κοινά χαρακτηριστικά με την εξίσου γνωστή γλώσσα προγραμματισμού Java, έχοντας δανειστεί και βελτιώσει ορισμένες δυνατότητες που παρέχονται από αυτές τις γλώσσες. Η Microsoft έχει δώσει μεγάλη βαρύτητα στην ανάπτυξη και στην εξέλιξη της C# και υποστηρίζει αναμφισβήτητα πως είναι πλήρως δομημένη πάνω στον .Net Framework, το όποιο δίνει την δυνατότητα σχεδόν σε όλα τα χαρακτηριστικά που συνδέονται σε σχέση με την γλώσσα να υπάρξουν, καθώς παρέχει άμεση πρόσβαση στις βιβλιοθήκες κλάσεων του .Net Framework.

Η γλώσσα προγραμματισμού C# επιχειρεί να πάρει τα καλύτερα στοιχεία από τις παλαιότερες γλώσσες σχετικά με την ασφάλεια τους. Ένα πρόγραμμα C# μπορεί να περιέχει διάφορα μέρη κώδικα όπου μπορεί να είναι διαχειριζόμενα ή μη διαχειριζόμενα. Ο διαχειριζόμενος κώδικας διαταράσσεται από το σύστημα, το οποίο τρέχει σε αυτό. Με το διαχειριζόμενο κώδικα διασφαλίζεται ότι είναι δύσκολο, αλλά όχι και αδύνατο, να αναγκάσει 23 τον ηλεκτρονικό υπολογιστή να έρθει σε άμεσο τερματισμό. Ωστόσο, με το διαχειριζόμενο κώδικα παρατηρούνται τα προγράμματα που εκτελούνται να τρέχουν πιο αργά. Για την μέγιστη δυνατή απόδοση και την ενεργοποίηση της άμεσης πρόσβασης σε τμήματα του υποκείμενου συστήματος του υπολογιστή,

μπορούν να οριστούν τα προγράμματά ως μη διαχειριζόμενοι κώδικες. Ένα μη διαχειριζόμενο πρόγραμμα εκτελείται πιο γρήγορα από ένα διαχειριζόμενο πρόγραμμα, αλλά με λιγότερη ασφάλεια. Σε περίπτωση που αναγκάσει τον ηλεκτρονικό υπολογιστή να έρθει σε άμεσο τερματισμό είναι δυνατόν να καταστρέψει τον ίδιο τον υπολογιστή. Η αλλαγή σε μη διαχειριζόμενη λειτουργία είναι ανάλογη με την αφαίρεση της ασφάλειας για να μπορέσει να εκτελείται πιο γρήγορα.

Ορισμένα από τα κύρια χαρακτηριστικά της γλώσσας είναι τα εξής :

Type – safe σχεδιασμός : καθιστά αδύνατη την ανάγνωση μεταβλητών που δεν έχουν αρχεικοποιήσει τη χρήση δεικτών που βρίσκονται εκτός ορίων πίνακα καθώς και την εκτέλεση unchecked type casts.

Garbage collection : Αποδεσμεύει αυτόματα τη μνήμη από αντικείμενα που δεν χρησιμοποιούνται πλέον από το πρόγραμμα.

Component – oriented προγραμματισμός : Υποστηρίζει το μοντέλο προγραμματισμού που βασίζεται σε components, δηλαδή σε στοιχεία προγράμματος που μπορούν να χρησιμοποιηθούν από άλλους χρήστες, οι οποίοι χρειάζεται να γνωρίζουν μόνο αυτά που κρίνει απαραίτητα ο δημιουργός του component και χωρίς ο ίδιος να ξέρει για τους τελικούς χρήστες.

Διαχείριση εξαιρέσεων : Παρέχει μια δομημένη και επεκτάσιμη προσέγγιση για την ανίχνευση σφαλμάτων.

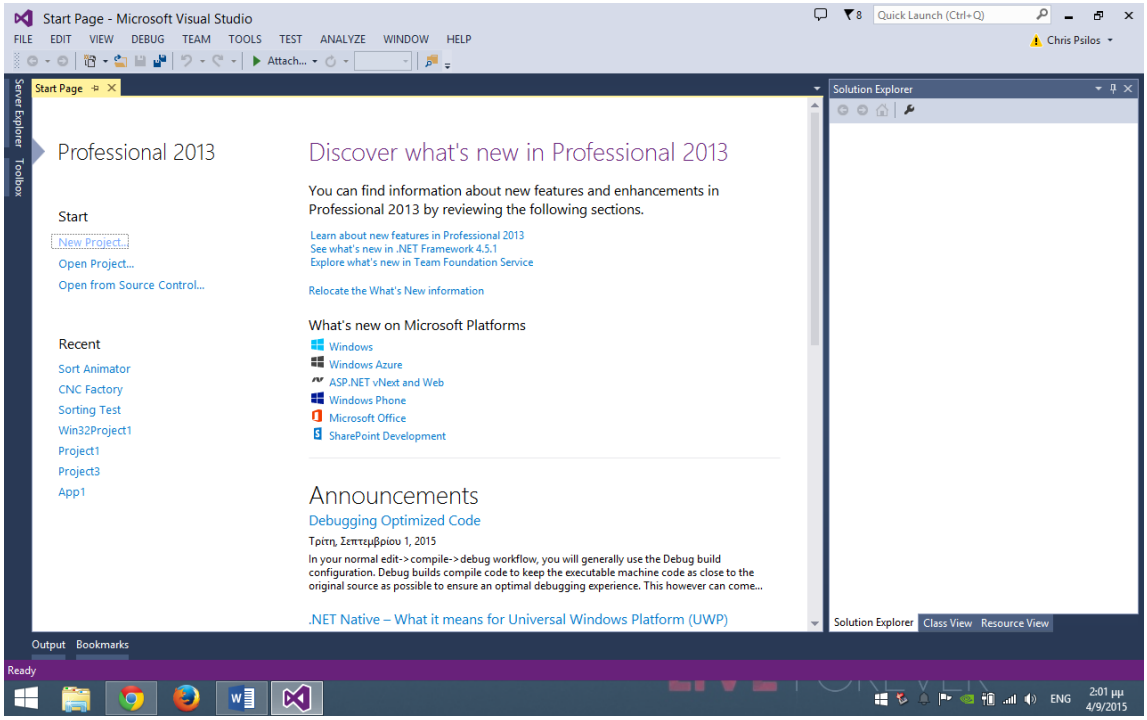
Γλώσσα Προγραμματισμού C++

Η Γλώσσα C++ είναι μια γενικής χρήσης γλώσσα προγραμματισμού που είναι ελεύθερης μορφής, καθώς η τοποθέτηση των χαρακτήρων στη σελίδα στο κείμενο του προγράμματος είναι ασήμαντη. Η γλώσσα προγραμματισμού C++ θεωρείται ως μια γλώσσα ενδιάμεσου επίπεδου, αυτό συμπεραίνεται καθώς περιλαμβάνει τόσο υψηλού επιπέδου, όσο και χαμηλού επιπέδου γλώσσα προγραμματισμού.

Η γλώσσα C++ είναι μία από τις πιο δημοφιλής γλώσσες προγραμματισμού και υλοποιείται σε μια ευρεία ποικιλία του υλικού και του λειτουργικού συστήματος. Θεωρείται ως μια από τις πιο αποτελεσματικές γλώσσες προγραμματισμού, καθώς μπορεί και χρησιμοποιείται σε πολλά ήδη κωδικοποίησης, όπως σε συστήματα λογισμικού, λογισμικό εφαρμογών, ενσωματωμένο λογισμικό, server υψηλών επιδόσεων, καθώς και σε λογισμικό ψυχαγωγίας. Εφόσον, η C++ μπορεί να χρησιμοποιηθεί σε μια μεγάλη ποικιλία μορφών κωδικοποίησης, ώθησε την εταιρεία της Microsoft να επιτρέπει την δημιουργία εφαρμογών για το λειτουργικό της σύστημα των Windows 8, καθώς με αυτό το τρόπο προσελκύει πολλούε προγραμματιστές να ασχοληθούν με την δημιουργία εφαρμογών σε γλώσσα C++.

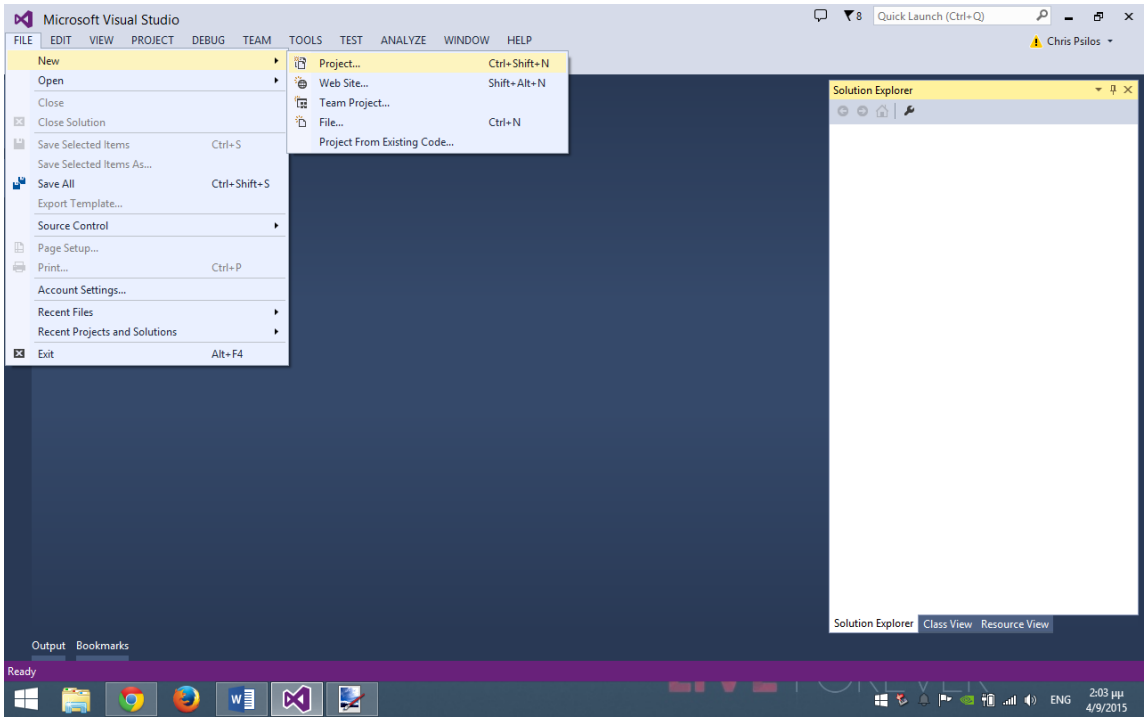
Οδηγιες χρήσης και επεξηγηση του περιβαλλοντος της εφαρμογης

Παρακάτω βλεπουμε την κεντρική σελιδα του περιβάλλοντος προγραμματισμού (IDE)

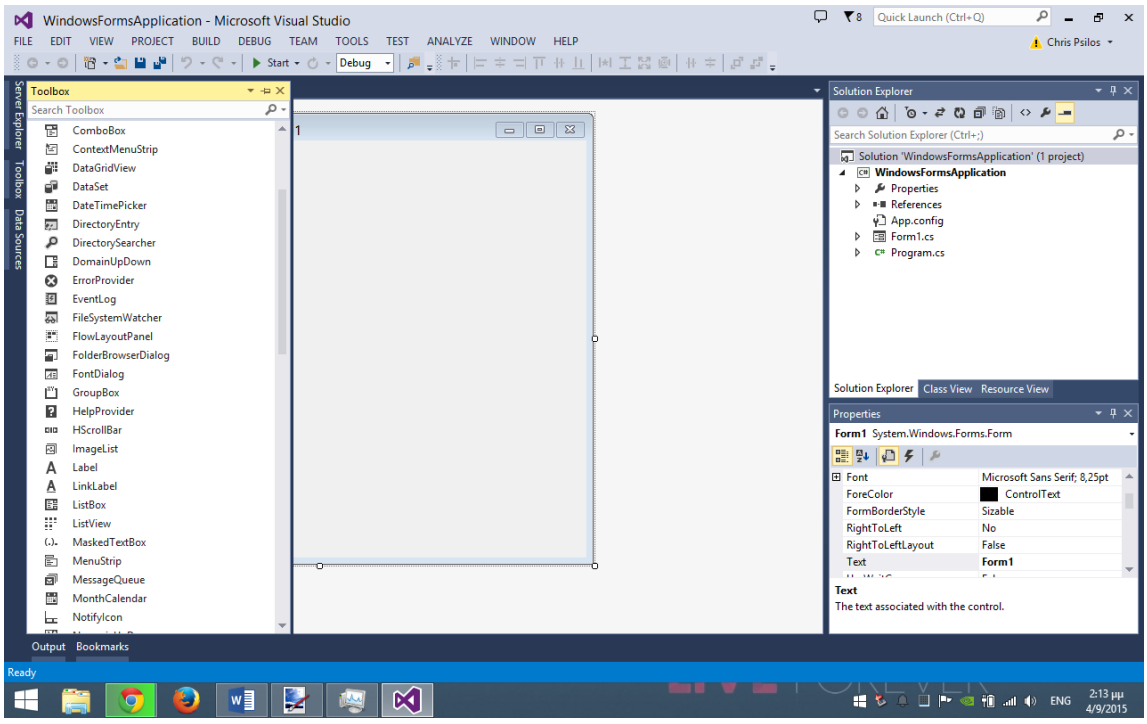


Εδώ παρατηρούμε όλες τις κύριες ενέργειες που μπορούμε να πραγματοποιήσουμε μέσω του περιβάλλοντος προγραμματισμού, όπως την δημιουργία νέου project , την επεξεργασία καποιου ήδη υπάρχοντος καθώς και μια σειρά από βοήθειες σχετικά με τη χρήση του περιβάλλοντος (tutorials).

Για την δημιουργία ενός νέου project χρησιμοποιούμε την επιλογή File -> New -> Project όπως φαίνεται στην παρακάτω εικόνα.



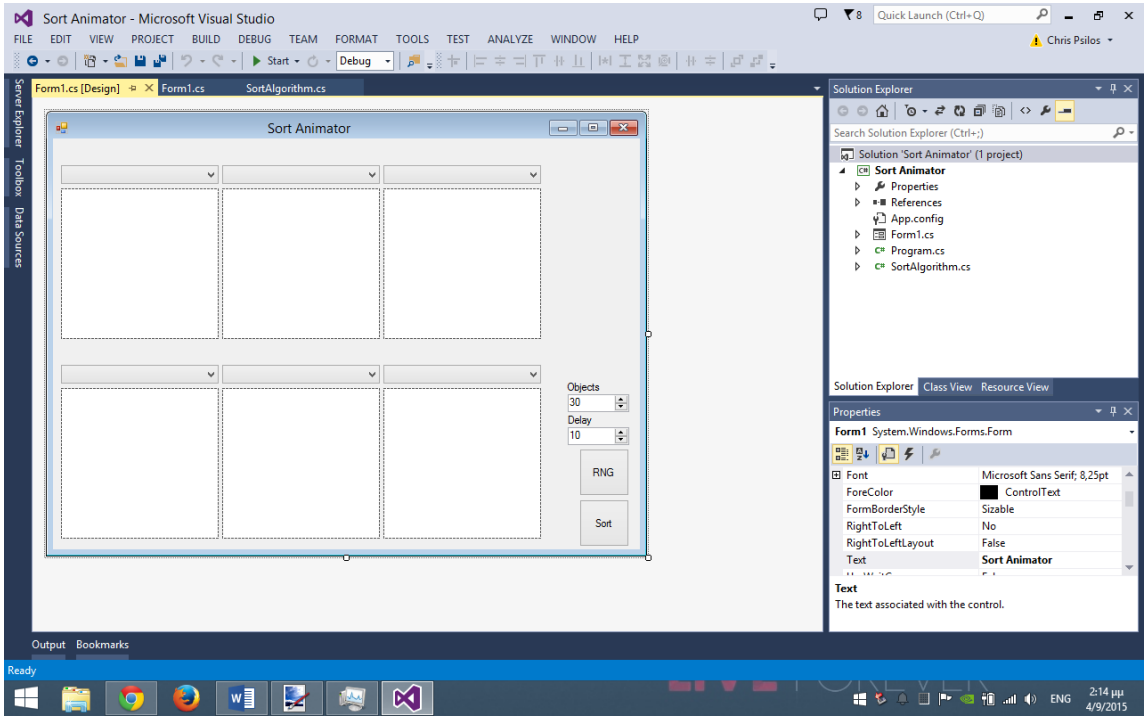
Αυτή η επιλογή μας εμφανίζει ένα χρήσιμο wizard με σκοπό να μας βοηθήσει να δημιουργήσουμε το νέο μας project. Εδώ θα επιλέξουμε τα βασικά χαρακτηριστικά της εφαρμογής μας όπως τη γλώσσα προγραμματισμού που θα χρησιμοποιήσουμε (C, C++, C#, Visual Basic, κλπ.) καθώς και το είδος του project μας (παραθυρική εφαρμογή, εφαρμογή του Windows Store, βιβλιοθήκη, κλπ.). Εμείς θα επιλέξουμε την γλώσσα C# και το είδος της εφαρμογής είναι Windows Forms Application. Επίσης θα χρειαστεί να δώσουμε ένα όνομα στη νέα μας εφαρμογή. Τέλος, πατώντας το κουμπί OK θα εμφανιστεί το κύριο παράθυρο της εφαρμογής μας, όπως φαίνεται παρακάτω.



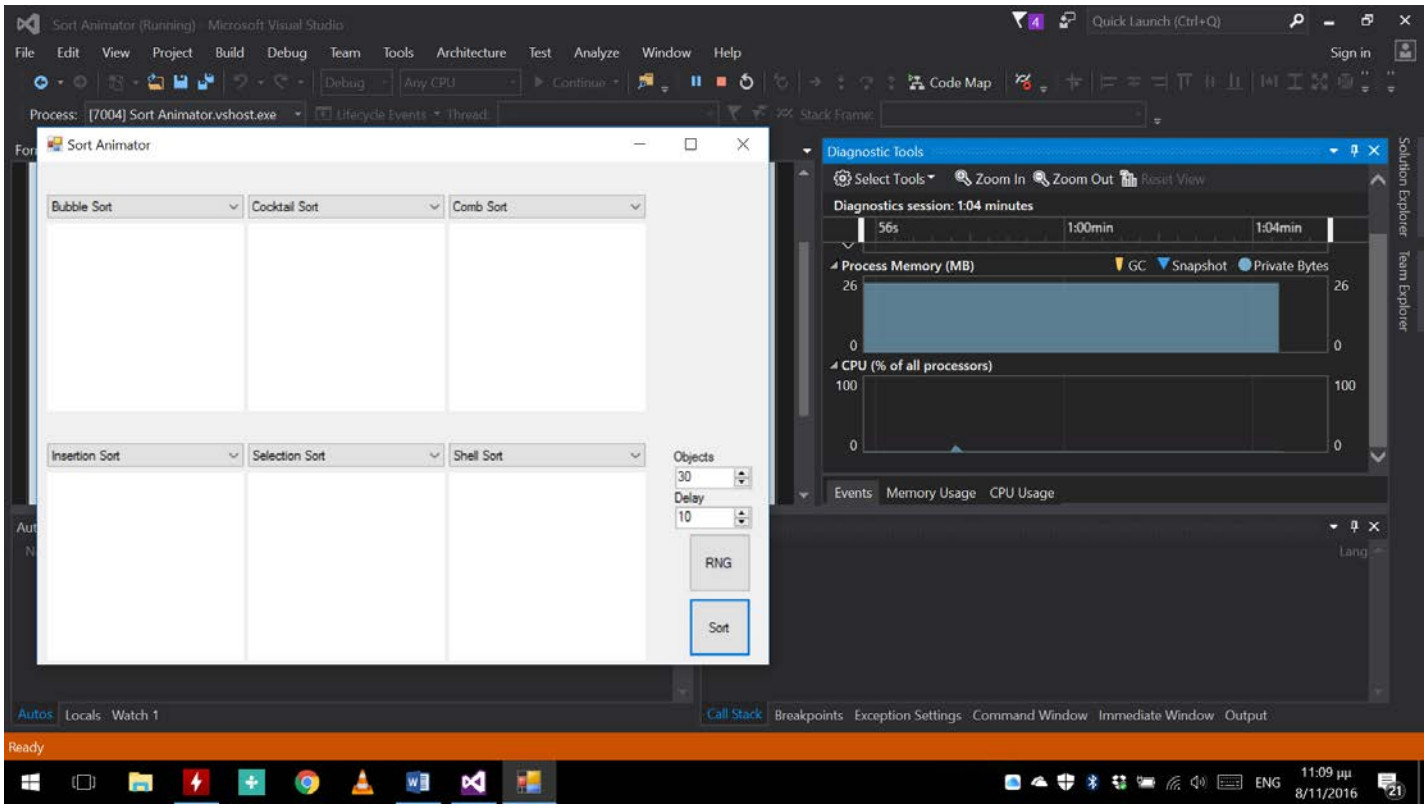
Εδώ πατώντας επάνω στην ταμπέλα Toolbox μπορούμε να προσθέσουμε όλα τα visual elements της εφαρμογής μας, όπως labels, buttons και λοιπά. Για την εφαρμογή θα χρειαστούμε:

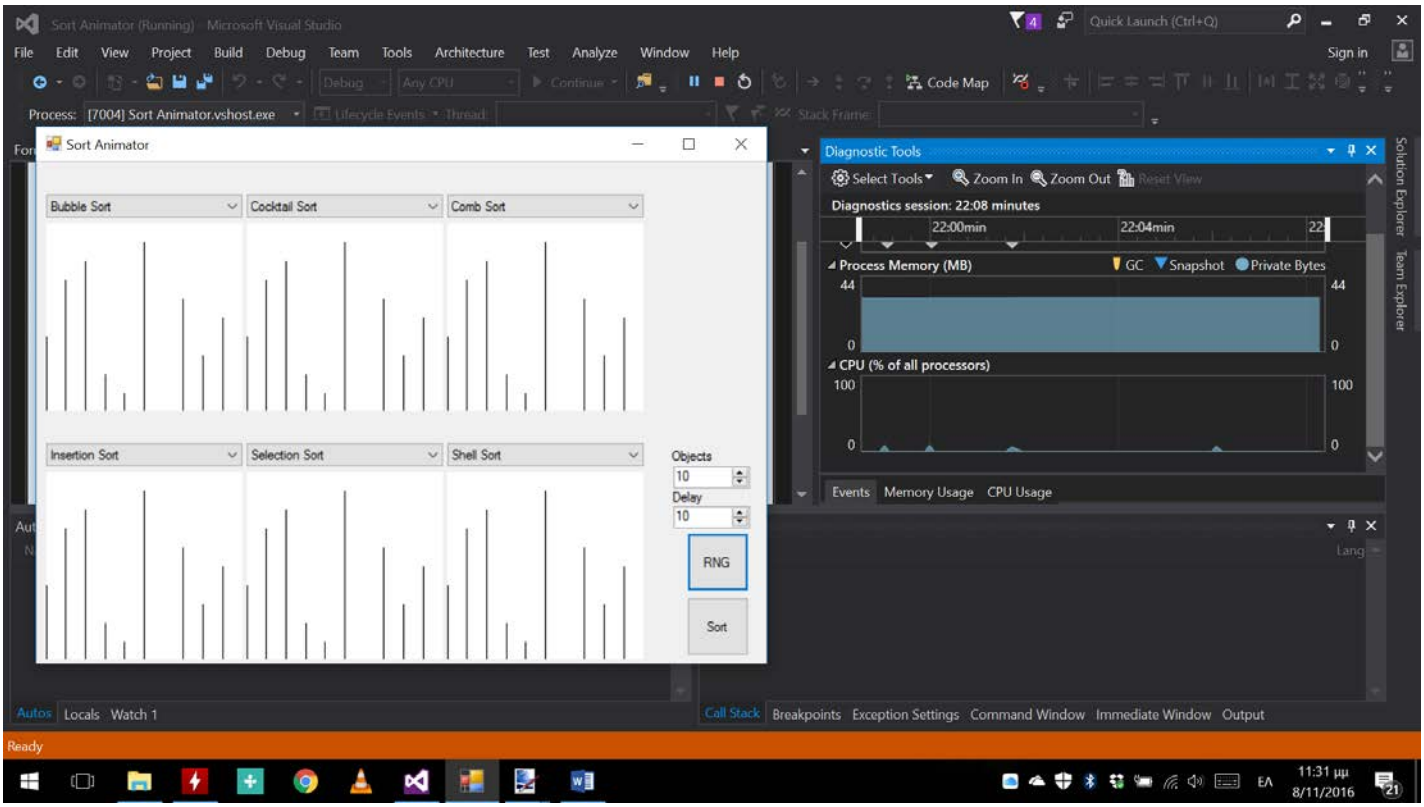
- 6 Picture Box
- 6 Combo Box
- 2 Label
- 2 NumericUpDown
- 2 Button

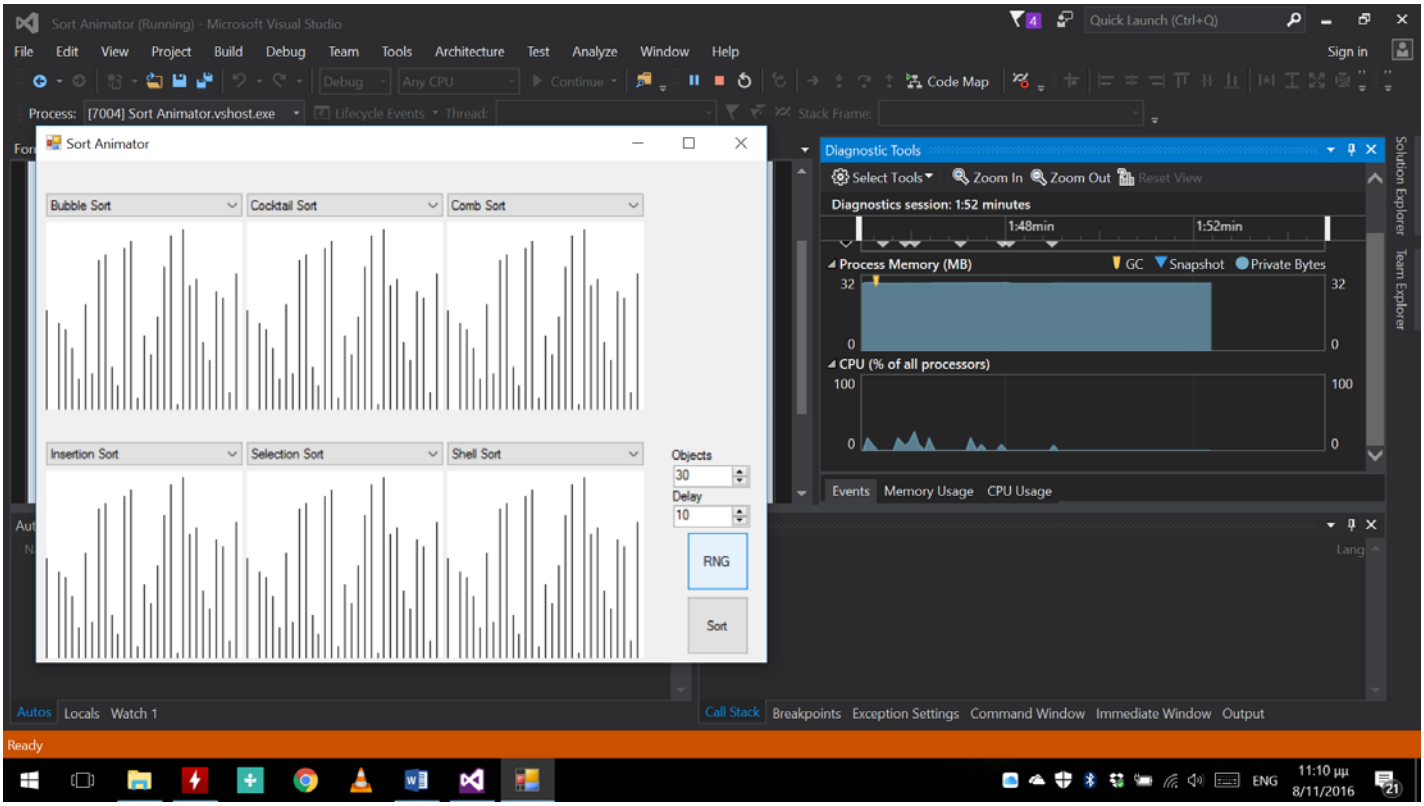
Το τελικό αποτέλεσμα πρέπει να μοιάζει όπως στην παρακάτω εικόνα

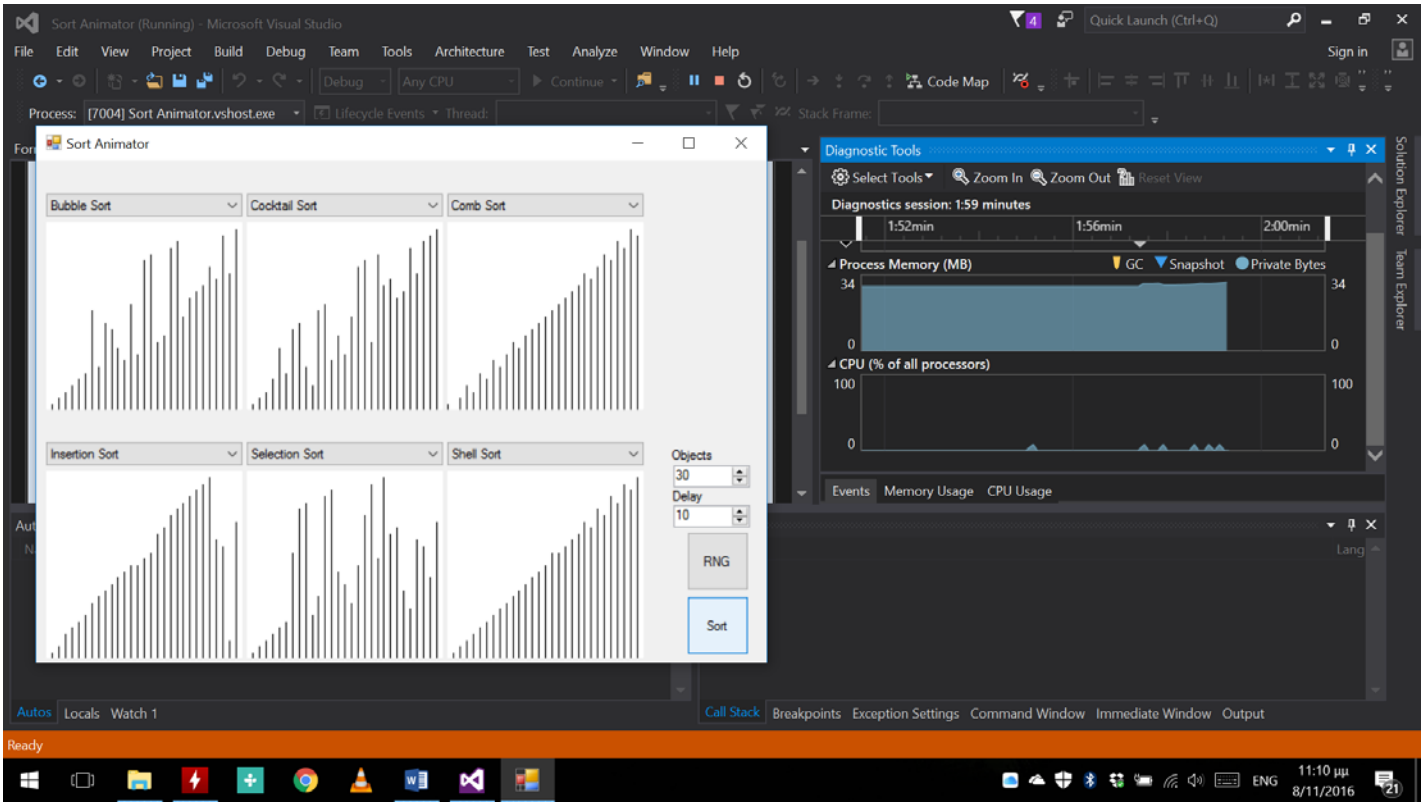


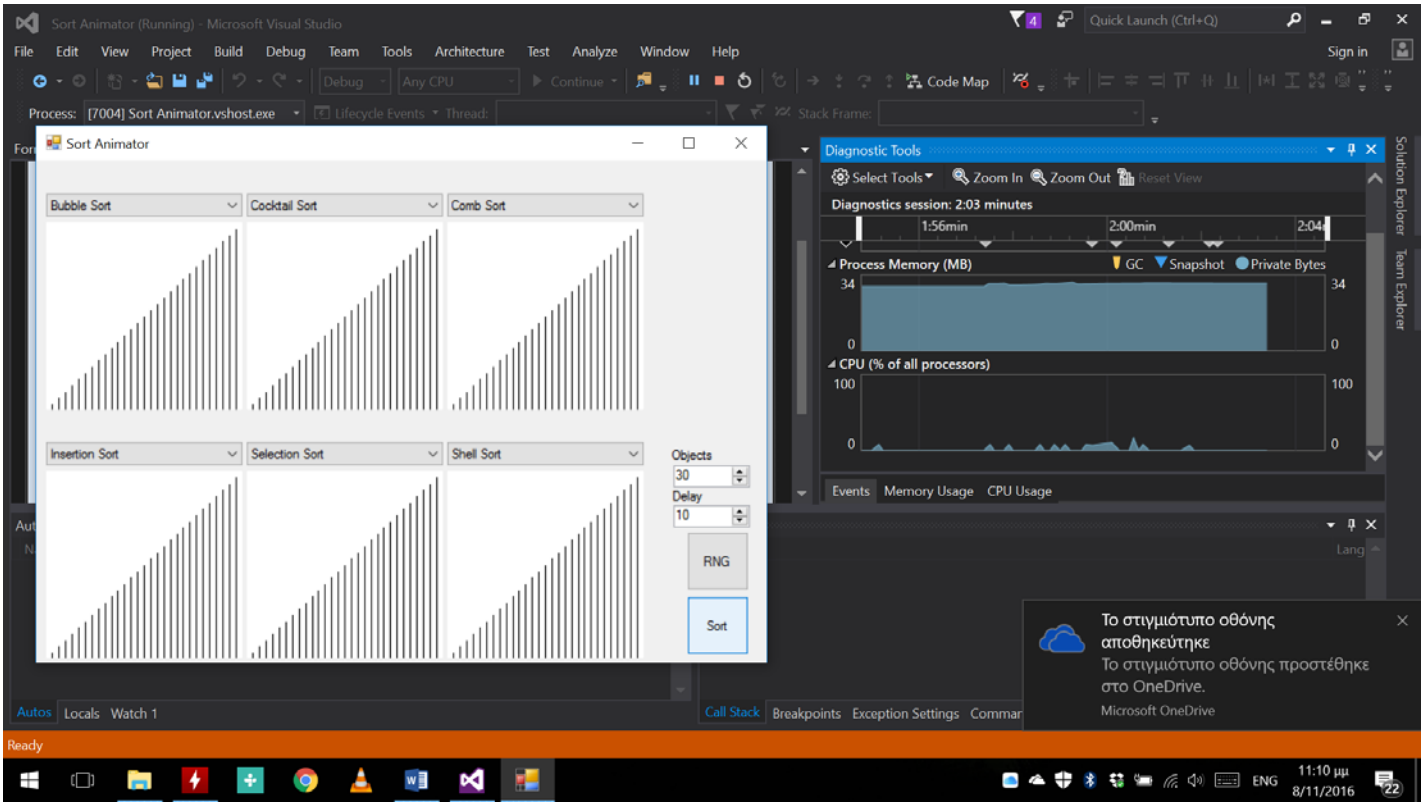
Παρακάτω παραθέτω κάποιες ενδεικτικές εικόνες από την εκτέλεση του προγράμματος:

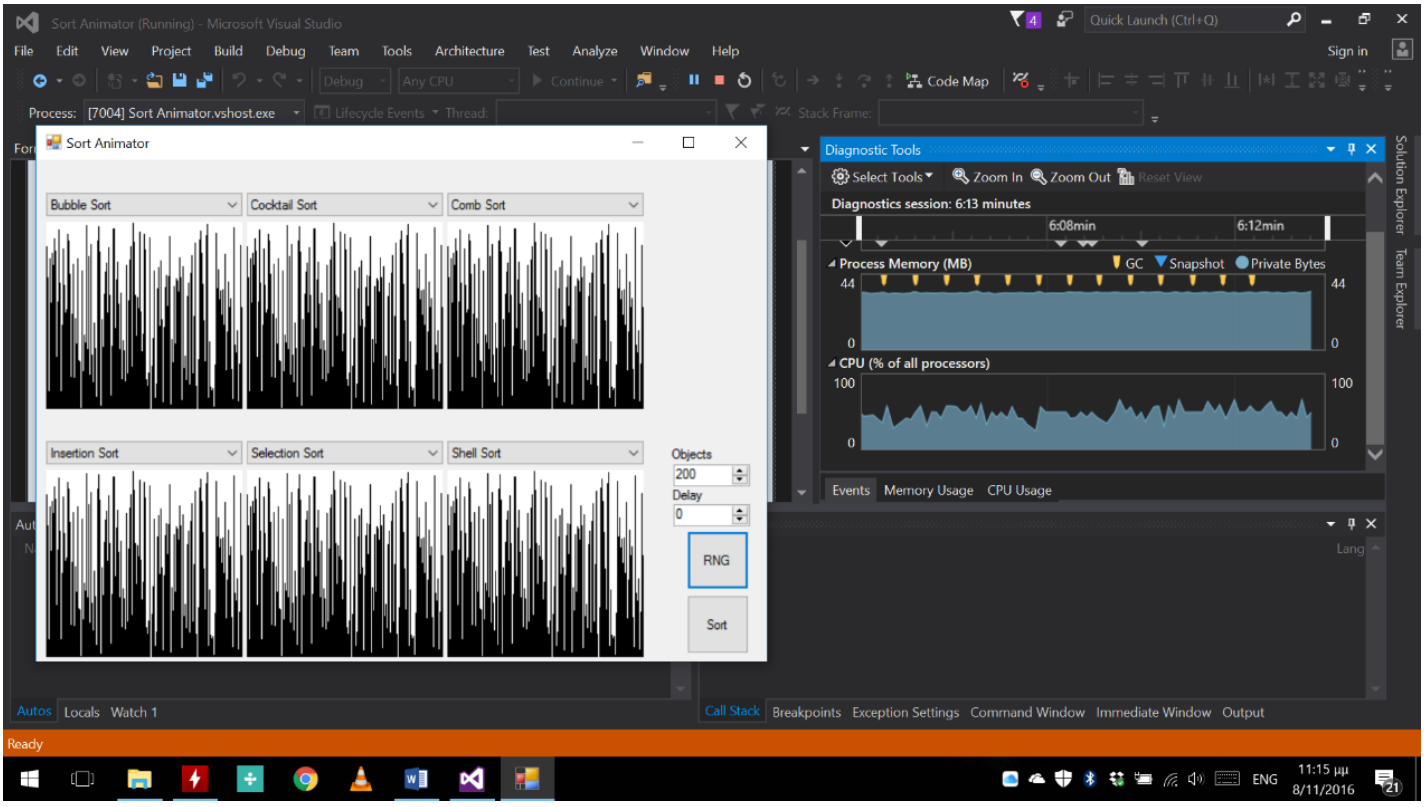


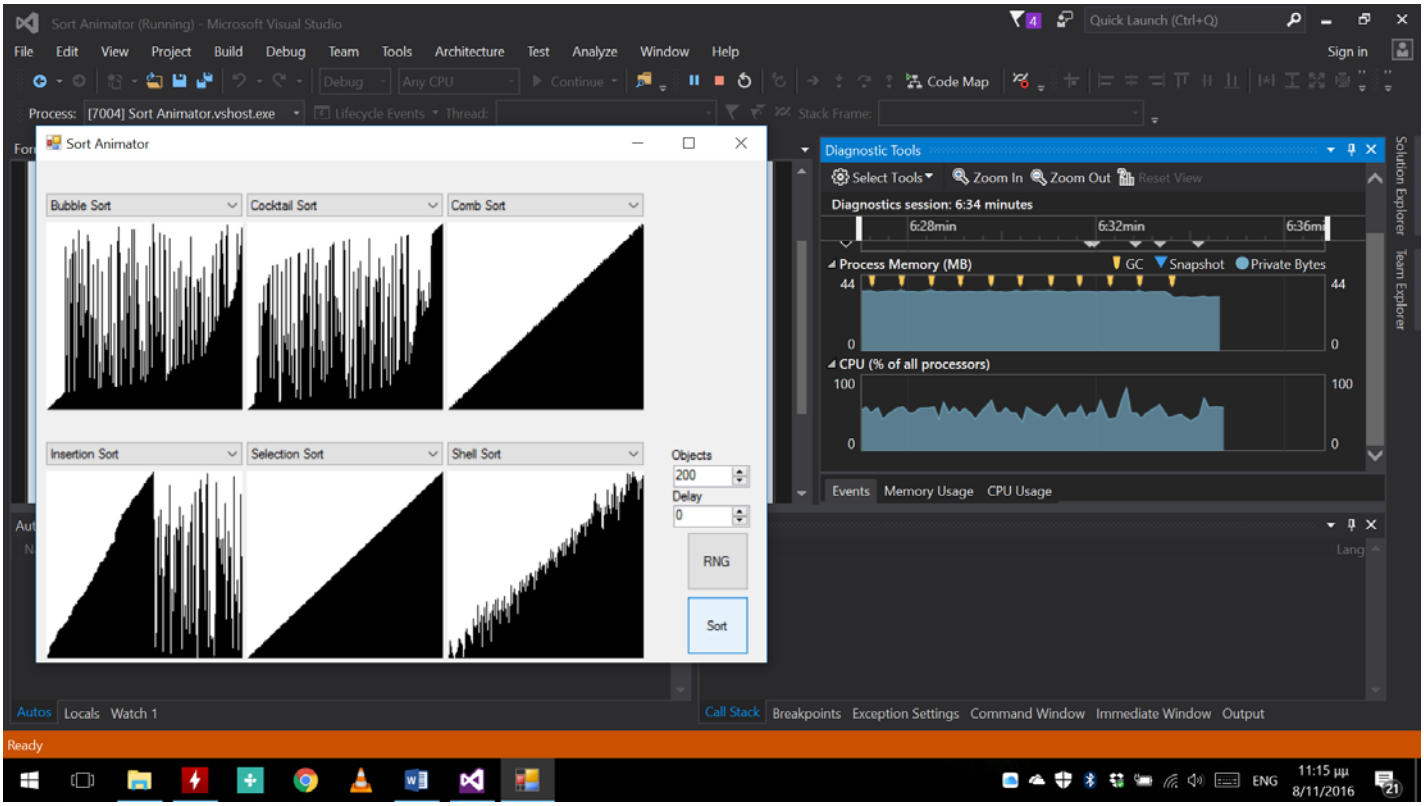


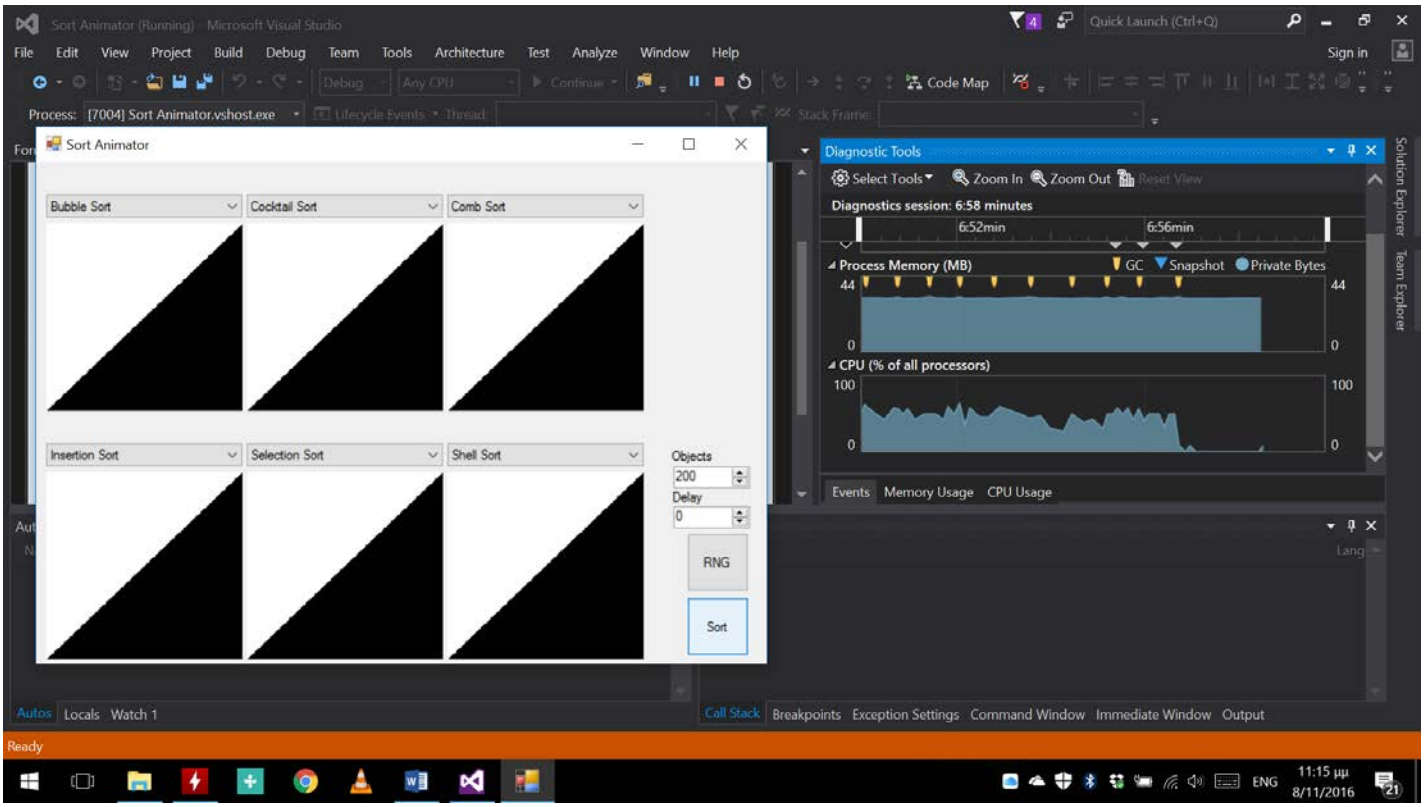

Sort Animator (Running) - Microsoft Visual Studio
Sort Animator
Bubble Sort
Cocktail Sort
Comb Sort
Insertion Sort
Selection Sort
Shell Sort
Objects
30
Delay
10
RNG
Sort
Diagnostic Tools
Diagnostics session: 1:52 minutes
1:48min
1:52min
Process Memory (MB)
GC
Snapshot
Private Bytes
CPU (% of all processors)
Events
Memory Usage
CPU Usage
Ready


Sort Animator (Running) - Microsoft Visual Studio
Sort Animator
Bubble Sort
Cocktail Sort
Comb Sort
Insertion Sort
Selection Sort
Shell Sort
Objects
30
Delay
10
RNG
Sort
Diagnostic Tools
Diagnostics session: 1:59 minutes
1:52min
1:56min
2:00min
Process Memory (MB)
GC
Snapshot
Private Bytes
CPU (% of all processors)
Events
Memory Usage
CPU Usage
Ready


Sort Animator (Running) - Microsoft Visual Studio
Sort Animator
Bubble Sort
Cocktail Sort
Comb Sort
Insertion Sort
Selection Sort
Shell Sort
Objects
30
Delay
10
RNG
Sort
Diagnostic Tools
Diagnostics session: 2:03 minutes
Process Memory (MB)
CPU (% of all processors)
Το στιγμιότυπο οθόνης αποθηκεύτηκε
Το στιγμιότυπο οθόνης προστέθηκε στο OneDrive.
Microsoft OneDrive
Ready


Sort Animator (Running) - Microsoft Visual Studio
Sort Animator
Bubble Sort
Cocktail Sort
Comb Sort
Insertion Sort
Selection Sort
Shell Sort
Objects
200
Delay
0
RNG
Sort
Diagnostic Tools
Diagnostics session: 6:13 minutes
Process Memory (MB)
CPU (% of all processors)
Ready


Sort Animator (Running) - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Architecture Test Analyze Window Help
Sort Animator
Bubble Sort
Cocktail Sort
Comb Sort
Insertion Sort
Selection Sort
Shell Sort
Objects
200
Delay
0
RNG
Sort
Diagnostic Tools
Diagnostics session: 6:34 minutes
Process Memory (MB)
CPU (% of all processors)
Events
Memory Usage
CPU Usage
Autos
Locals
Watch 1
Call Stack
Breakpoints
Exception Settings
Command Window
Immediate Window
Output
Ready


Sort Animator (Running) Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Architecture Test Analyze Window Help
Sort Animator
Bubble Sort
Cocktail Sort
Comb Sort
Insertion Sort
Selection Sort
Shell Sort
Objects
200
Delay
0
RNG
Sort
Diagnostic Tools
Diagnostics session: 6:58 minutes
Process Memory (MB)
CPU (% of all processors)
Events
Memory Usage
CPU Usage
Autos
Locals
Watch 1
Call Stack
Breakpoints
Exception Settings
Command Window
Immediate Window
Output
Ready

Κώδικας εφαρμογής

Επεξήγηση κώδικα εφαρμογής

```
public partial class Form1 : Form
  {
     Graphics g1;
     ArrayList array1,array2,array3,array4,array5,array6;
     Bitmap bmpsave;
     ArrayList sorter = new ArrayList();

     static Random rng = new Random();

     public Form1()
     {
        InitializeComponent();

        sorter.Add("Bubble Sort");
        sorter.Add("Cocktail Sort");
        sorter.Add("Comb Sort");
        sorter.Add("Insertion Sort");
        sorter.Add("Selection Sort");
        sorter.Add("Counting Sort");
        sorter.Add("Shell Sort");
        sorter.Add("Heap Sort");
        sorter.Add("Merge Sort");
        sorter.Add("Quick Sort");
        sorter.Add("Gnome Sort");
        sorter.Add("Bucket Sort");
        sorter.Add("Cycle Sort");
        sorter.Add("Odd-Even Sort");
        sorter.Add("Pidgeon Hole Sort");

        comboBox1.DataSource = new ArrayList(sorter);
        comboBox2.DataSource = new ArrayList(sorter);
        comboBox3.DataSource = new ArrayList(sorter);
        comboBox4.DataSource = new ArrayList(sorter);
        comboBox5.DataSource = new ArrayList(sorter);
        comboBox6.DataSource = new ArrayList(sorter);

        comboBox2.SelectedIndex = 1;
        comboBox3.SelectedIndex = 2;
        comboBox4.SelectedIndex = 3;
        comboBox5.SelectedIndex = 4;
        comboBox6.SelectedIndex = 5;
```

Στο παραπάνω κομμάτι κώδικά βλέπουμε τη βασική κλάση της φόρμας μας (Form1) οπού και γίνονται όλες οι απαραίτητες αρχικοποιήσεις των οπτικών αντικειμένων που βρίσκονται σε αυτήν. Αρχικά προσθέτουμε όλες τις επιλογές του χρήστη στα ComboBoxes και στην συνέχεις δίνουμε μια αρχική επιλογή στο καθένα.

```
private void rng_Click(object sender, EventArgs e)
```

```
        {
            bmpsave = new Bitmap(pnlSort1.Width, pnlSort1.Height);

            g1 = Graphics.FromImage(bmpsave);

            pnlSort1.Image = bmpsave;
            pnlSort2.Image = bmpsave;
            pnlSort3.Image = bmpsave;
            pnlSort4.Image = bmpsave;
            pnlSort5.Image = bmpsave;
            pnlSort6.Image = bmpsave;

            array1 = new ArrayList((int)numericUpDownObjects.Value);

            for (int i = 0; i < array1.Capacity; i++)
            {
                int y = (int)((double)i / array1.Capacity * pnlSort1.Height);
                array1.Add(y);
            }
            Randomize(array1);

            DrawSamples(g1);
        }

public void Randomize(IList list)
        {


            for (int i = list.Count - 1; i > 0; i--)
            {
                int swapIndex = rng.Next(i + 1);
                if (swapIndex != i)
                {
                    object tmp = list[swapIndex];
                    list[swapIndex] = list[i];
                    list[i] = tmp;
                }
            }
        }

private void DrawSamples(Graphics g)
        {
            g.Clear(Color.White);
            //g.Clear(Color.White);

            for (int i = 0; i < array1.Count; i++)
            {
                int x = (int)(((double)pnlSort1.Width / array1.Count) * i);

                Pen pen = new Pen(Color.Black);
                g.DrawLine(pen, new Point(x, pnlSort1.Height), new Point(x, (int)(pnlSort1.Height -
(int)array1[i])));
            }
        }
```

Παραπάνω βλέπουμε τη μέθοδο που καλείται όταν πατήσουμε το RNG Button καθώς και τις συμπληρωματικές μεθόδους που καλλούνται από αυτή. Αρχικά δημιουργούμε ένα bitmap στο μέγεθος του πάνελ και ένα νέο ArrayList με όσες θέσεις έχει ορίσει ο χρήστης. Στη συνέχεια γεμίζουμε τον πίνακα με ψευδοτυχαίες τιμές, με μια ομοιόμορφη δομή έτσι ώστε κανένα element να μην υπερβαίνει τα όρια του πάνελ αργότερα όταν θα σκιαγραφηθεί. Η λογική που χρησιμοποιείται είναι έτσι ώστε το κάθε item να χρησιμοποιεί ένα κλάσμα του

διαθέσιμου ύψους του πάνελ. Δηλαδή το 1ο item χρησιμοποιεί ένα μικρό ποσοστό του ύψους του πάνελ ενώ το τελευταίο item χρησιμοποιεί όλο το διαθέσιμο ύψος του πάνελ. Αυτό μας οδηγεί σε ένα ήδη ταξινομημένο πίνακα οπού όλα τα items έχουν διαφορετική τιμή. Στη συνέχεια αλλάζουμε θέση σε τυχαία items ώστε ο πίνακας να αποταξινομηθεί. Τέλος γίνεται η σκιαγράφηση του πίνακα στο εκάστοτε πάνελ.

```
private void Sort_Click(object sender, EventArgs e)
      {
         int speed = (int)(numericUpDownSpeed.Value);
         //string alg = "";

         array2 = new ArrayList(array1);
         ……..
         array6 = new ArrayList(array1);

         SortAlgorithm sa1 = new SortAlgorithm(array1, pnlSort1, true, "", speed, "", -1);
         ……..
         SortAlgorithm sa6 = new SortAlgorithm(array6, pnlSort6, true, "", speed, "", -1);

         sa1.SortSelect(comboBox1);
         ……..
         sa6.SortSelect(comboBox6);

         ThreadStart ts = delegate()
         {
            if (sa1.GetEidos() == 0)
               sa1.BubbleSort(array1);
            else if (sa1.GetEidos() == 1)
               sa1.BiDerectionalBubbleSort(array1);
            else if (sa1.GetEidos() == 2)
               sa1.CombSort(array1);
            else if (sa1.GetEidos() == 3)
               sa1.InsertionSort(array1);
            else if (sa1.GetEidos() == 4)
               sa1.SelectionSort(array1);
            else if (sa1.GetEidos() == 5)
               sa1.CountingSort(array1);
            else if (sa1.GetEidos() == 6)
               sa1.ShellSort(array1);
            else if (sa1.GetEidos() == 7)
               sa1.HeapSort(array1);
            else if (sa1.GetEidos() == 8)
               sa1.MergeSort(array1, 0, array1.Count - 1);
            else if (sa1.GetEidos() == 9)
               sa1.QuickSort(array1, 0, array1.Count - 1);
            else if (sa1.GetEidos() == 10)
               sa1.GnomeSort(array1);
            else if (sa1.GetEidos() == 11)
               sa1.BucketSort(array1);
            else if (sa1.GetEidos() == 12)
               sa1.CycleSort(array1);
            else if (sa1.GetEidos() == 13)
               sa1.OddEvenSort(array1);
            else if (sa1.GetEidos() == 14)
               sa1.PigeonHoleSort(array1);
         };
         ……..
         ThreadStart ts6 = delegate()
```

```
        {
            if (sa6.GetEidos() == 0)
                sa6.BubbleSort(array6);
            else if (sa6.GetEidos() == 1)
                sa6.BiDerectionalBubbleSort(array6);
            else if (sa6.GetEidos() == 2)
                sa6.CombSort(array6);
            else if (sa6.GetEidos() == 3)
                sa6.InsertionSort(array6);
            else if (sa6.GetEidos() == 4)
                sa6.SelectionSort(array6);
            else if (sa6.GetEidos() == 5)
                sa6.CountingSort(array6);
            else if (sa6.GetEidos() == 6)
                sa6.ShellSort(array6);
            else if (sa6.GetEidos() == 7)
                sa6.HeapSort(array6);
            else if (sa6.GetEidos() == 8)
                sa6.MergeSort(array6, 0, array6.Count - 1);
            else if (sa6.GetEidos() == 9)
                sa6.QuickSort(array6, 0, array6.Count - 1);
            else if (sa6.GetEidos() == 10)
                sa6.GnomeSort(array6);
            else if (sa6.GetEidos() == 11)
                sa6.BucketSort(array6);
            else if (sa6.GetEidos() == 12)
                sa6.CycleSort(array6);
            else if (sa6.GetEidos() == 13)
                sa6.OddEvenSort(array6);
            else if (sa6.GetEidos() == 14)
                sa6.PigeonHoleSort(array6);
        };

        Thread thr1 = new Thread(ts);
        thr1.Start();
        ……
        Thread thr6 = new Thread(ts6);
        thr6.Start();
    }
```

Παραπάνω βλέπουμε την μέθοδο που χρησιμοποιείται όταν πατήσουμε το Sort Button. Αρχικά, αντιγράφουμε τον τυχαίο πίνακα που δημιουργήθηκε 5 φορές για να μπορέσουμε να ταξινομήσουμε τον κάθε ένα ξεχωριστά, με διαφορετική μέθοδο. Στη συνέχεια δημιουργούμε νέα αντικείμενα για κάθε πανελ και αρχικοποιούμε τις τιμές τους. Επίσης, μέσα από ένα δέντρο if-else καθορίζουμε ποια μέθοδο έχει επιλέξει ο χρήστης για το κάθε πάνελ και καλούμε την κατάλληλη μέθοδο ταξινόμησης, δημιουργώντας πολλαπλά ασύγχρονα threads, ώστε να μπορέσουν να τρέξουν όλες οι μέθοδοι ταυτόχρονα μεταξύ τους.

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Sort_Animator
{
   static class Program
   {
      /// <summary>
      /// The main entry point for the application.
      /// </summary>
      [STAThread]
      static void Main()
      {
         Application.EnableVisualStyles();
         Application.SetCompatibleTextRenderingDefault(false);
         Application.Run(new Form1());
      }
   }
}
```

SortAlgorithm.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing;
using System.Windows.Forms;
using System.Drawing.Imaging;
using System.Threading;

using System.Diagnostics;

namespace SortComparison
{
   public class SortAlgorithm
   {

      ArrayList arrayToSort;
      Graphics g;
      Bitmap bmpsave;
      PictureBox pnlSamples;
      bool savePicture;
      string outputFolder;
      string outputFile;
      int imgCount;
      int speed;
      int eidos;
      Stopwatch Sw = new Stopwatch();

      public SortAlgorithm(ArrayList list, PictureBox pic, bool sp, string of, int s, string outFile, int
eidos)
      {
         imgCount = 0;
```

```
        arrayToSort = list;
        pnlSamples = pic;
        savePicture = sp;
        outputFolder = of;
        speed = s;
        outputFile = outFile;

        bmpsave = new Bitmap(pnlSamples.Width, pnlSamples.Height);
        g = Graphics.FromImage(bmpsave);

        pnlSamples.Image = bmpsave;
        DrawSamples();

    }

    public IList BubbleSort(IList arrayToSort)
    {
        Sw.Start();

        int n = arrayToSort.Count - 1;
        for (int i = 0; i < n; i++)
        {

            for (int j = n; j > i; j--)
            {
                if (((IComparable)arrayToSort[j - 1]).CompareTo(arrayToSort[j]) > 0)
                {
                    object temp = arrayToSort[j - 1];
                    arrayToSort[j - 1] = arrayToSort[j];
                    arrayToSort[j] = temp;
                    RedrawItem(j);
                    RedrawItem(j - 1);
                    RefreshPanel(pnlSamples);
                    if (savePicture)
                        SavePicture();

                }
                Thread.Sleep(speed);
            }
        }

        Sw.Stop();
        return arrayToSort;
    }

    public IList BiDerectionalBubbleSort(IList arrayToSort)
    {
        Sw.Start();

        int limit = arrayToSort.Count;
        int st = -1;
        bool swapped = false;
        do
        {
            swapped = false;
            st++;
            limit--;

            for (int j = st; j < limit; j++)
            {
                if (((IComparable)arrayToSort[j]).CompareTo(arrayToSort[j + 1]) > 0)
                {
                    object temp = arrayToSort[j];
```

```
                arrayToSort[j] = arrayToSort[j + 1];
                arrayToSort[j + 1] = temp;
                swapped = true;
                RedrawItem(j);
                RedrawItem(j + 1);
                RefreshPanel(pnlSamples);
                if (savePicture)
                    SavePicture();

            }
            Thread.Sleep(speed);

        }
        for (int j = limit - 1; j >= st; j--)
        {
            if (((IComparable)arrayToSort[j]).CompareTo(arrayToSort[j + 1]) > 0)
            {
                object temp = arrayToSort[j];
                arrayToSort[j] = arrayToSort[j + 1];
                arrayToSort[j + 1] = temp;
                swapped = true;
                RedrawItem(j);
                RedrawItem(j + 1);

                RefreshPanel(pnlSamples);
                if (savePicture)
                    SavePicture();

            }
            Thread.Sleep(speed);
        }

    } while (st < limit && swapped);

    Sw.Stop();
    return arrayToSort;
}

public IList CombSort(IList arrayToSort)
{
    Sw.Start();

    int gap = arrayToSort.Count;
    int swaps = 0;

    do
    {
        gap = (int)(gap / 1.247330950103979);
        if (gap < 1)
        {
            gap = 1;
        }
        int i = 0;
        swaps = 0;

        do
        {
            if (((IComparable)arrayToSort[i]).CompareTo(arrayToSort[i + gap]) > 0)
            {
                object temp = arrayToSort[i];
                arrayToSort[i] = arrayToSort[i + gap];
                arrayToSort[i + gap] = temp;
                RedrawItem(i);
```

```
                RedrawItem(i + gap);
                RefreshPanel(pnlSamples);
                if (savePicture)
                    SavePicture();
                swaps = 1;
            }
            i++;
            Thread.Sleep(speed);
        } while (!(i + gap >= arrayToSort.Count));

    } while (!(gap == 1 && swaps == 0));

    Sw.Stop();
    return arrayToSort;
}

public IList InsertionSort(IList arrayToSort)
{
    for (int i = 1; i < arrayToSort.Count; i++)
    {
        object val = arrayToSort[i];
        int j = i - 1;
        bool done = false;
        do
        {
            if (((IComparable)arrayToSort[j]).CompareTo(val) > 0)
            {
                arrayToSort[j + 1] = arrayToSort[j];
                RedrawItem(j + 1);
                RefreshPanel(pnlSamples);
                if (savePicture)
                    SavePicture();
                j--;
                if (j < 0)
                {
                    done = true;
                }
            }
            else
            {
                done = true;
            }
            Thread.Sleep(speed);
        } while (!done);
        arrayToSort[j + 1] = val;

        RedrawItem(j + 1);
        RefreshPanel(pnlSamples);
        if (savePicture)
            SavePicture();
        Thread.Sleep(speed);
    }
    return arrayToSort;
}

public IList SelectionSort(IList arrayToSort)
{
    int min;
    for (int i = 0; i < arrayToSort.Count; i++)
    {

        min = i;
        for (int j = i + 1; j < arrayToSort.Count; j++)
```

```
            {
                if (((IComparable)arrayToSort[j]).CompareTo(arrayToSort[min]) < 0)
                {
                    min = j;
                }
                Thread.Sleep(speed);
            }
            object temp = arrayToSort[i];
            arrayToSort[i] = arrayToSort[min];
            arrayToSort[min] = temp;

            RedrawItem(i);
            RedrawItem(min);
            RefreshPanel(pnlSamples);
            if (savePicture)
                SavePicture();

            Thread.Sleep(speed);
        }

        return arrayToSort;
    }

    public IList CountingSort(IList arrayToSort)
    {
        object min;
        object max;

        min = max = arrayToSort[0];

        for (int i = 0; i < arrayToSort.Count; i++)
        {
            if (((IComparable)arrayToSort[i]).CompareTo(min) < 0)
            {
                min = arrayToSort[i];
            }
            else if (((IComparable)arrayToSort[i]).CompareTo(max) > 0)
            {
                max = arrayToSort[i];
            }
        }

        int range = (int)max - (int)min + 1;

        int[] count = new int[range * sizeof(int)];

        for (int i = 0; i < range; i++)
        {
            count[i] = 0;
        }

        for (int i = 0; i < arrayToSort.Count; i++)
        {
            count[(int)arrayToSort[i] - (int)min]++;
        }
        int z = 0;
        for (int i = (int)min; i < arrayToSort.Count; i++)
        {
            for (int j = 0; j < count[i - (int)min]; j++)
            {
                arrayToSort[z++] = i;
                RedrawItem(z - 1);
                RefreshPanel(pnlSamples);
```

```
                if (savePicture)
                    SavePicture();
            }
        }

        return arrayToSort;
    }

    public IList ShellSort(IList arrayToSort)
    {
        int i, j, increment;
        object temp;

        increment = arrayToSort.Count / 2;

        while (increment > 0)
        {
            for (i = 0; i < arrayToSort.Count; i++)
            {
                j = i;
                temp = arrayToSort[i];
                while ((j >= increment) && (((IComparable)arrayToSort[j -
increment]).CompareTo(temp) > 0))
                {
                    arrayToSort[j] = arrayToSort[j - increment];
                    RedrawItem(j);
                    RefreshPanel(pnlSamples);
                    if (savePicture)
                        SavePicture();
                    j = j - increment;
                    Thread.Sleep(speed);
                }
                arrayToSort[j] = temp;
                RedrawItem(j);
                RefreshPanel(pnlSamples);
                if (savePicture)
                    SavePicture();
                Thread.Sleep(speed);
            }
            if (increment == 2)
                increment = 1;
            else
                increment = increment * 5 / 11;
        }

        return arrayToSort;
    }

    public IList HeapSort(IList list)
    {
        for (int i = (list.Count - 1) / 2; i >= 0; i--)
        {
            Adjust(list, i, list.Count - 1);
            Thread.Sleep(speed);
        }

        for (int i = list.Count - 1; i >= 1; i--)
        {
            object Temp = list[0];
            list[0] = list[i];
            list[i] = Temp;
            RedrawItem(0);
            RedrawItem(i);
```

```
            RefreshPanel(pnlSamples);
            if (savePicture)
                SavePicture();
            Adjust(list, 0, i - 1);
            Thread.Sleep(speed);
        }

        return list;
    }

    public void Adjust(IList list, int i, int m)
    {
        object Temp = list[i];
        int j = i * 2 + 1;
        while (j <= m)
        {
            if (j < m)
                if (((IComparable)list[j]).CompareTo(list[j + 1]) < 0)
                    j = j + 1;

            if (((IComparable)Temp).CompareTo(list[j]) < 0)
            {
                list[i] = list[j];
                RedrawItem(i);
                RefreshPanel(pnlSamples);
                if (savePicture)
                    SavePicture();
                i = j;
                j = 2 * i + 1;
            }
            else
            {
                j = m + 1;
            }
            Thread.Sleep(speed);
        }
        list[i] = Temp;
        RedrawItem(i);
        RefreshPanel(pnlSamples);
        if (savePicture)
            SavePicture();
        Thread.Sleep(speed);
    }

    public IList MergeSort(IList a, int low, int height)
    {
        int l = low;
        int h = height;

        if (l >= h)
        {
            return a;
        }

        int mid = (l + h) / 2;
        Thread.Sleep(speed);
        MergeSort(a, l, mid);
        MergeSort(a, mid + 1, h);

        int end_lo = mid;
        int start_hi = mid + 1;
        while ((l <= end_lo) && (start_hi <= h))
        {
```

```
            if (((IComparable)a[l]).CompareTo(a[start_hi]) < 0)
            {
                l++;
            }
            else
            {
                object temp = a[start_hi];
                for (int k = start_hi - 1; k >= l; k--)
                {
                    a[k + 1] = a[k];
                    RedrawItem(k + 1);
                    RefreshPanel(pnlSamples);
                    if (savePicture)
                        SavePicture();
                    Thread.Sleep(speed);
                }
                a[l] = temp;
                RedrawItem(l);
                RefreshPanel(pnlSamples);
                if (savePicture)
                    SavePicture();
                l++;
                end_lo++;
                start_hi++;
            }
        }
        return a;
    }

    public IList QuickSort(IList a, int left, int right)
    {
        int i = left;
        int j = right;
        double pivotValue = ((left + right) / 2);
        int x = (int)a[int.Parse(pivotValue.ToString())];
        Thread.Sleep(speed);

        while (i <= j)
        {
            while (((IComparable)a[i]).CompareTo(x) < 0)
            {
                i++;
            }
            while (((IComparable)x).CompareTo(a[j]) < 0)
            {
                j--;
            }
            if (i <= j)
            {
                object temp = a[i];
                a[i] = a[j];
                RedrawItem(i);
                i++;
                a[j] = temp;
                RedrawItem(j);
                j--;
                RefreshPanel(pnlSamples);
                if (savePicture)
                    SavePicture();
            }
            Thread.Sleep(speed);
        }
        if (left < j)
```

```
    {
        QuickSort(a, left, j);
    }
    if (i < right)
    {
        QuickSort(a, i, right);
    }
    return a;
}

public IList GnomeSort(IList arrayToSort)
{
    int pos = 1;
    while (pos < arrayToSort.Count)
    {
        if (((IComparable)arrayToSort[pos]).CompareTo(arrayToSort[pos - 1]) >= 0)
        {
            pos++;
        }
        else
        {
            object temp = arrayToSort[pos];
            arrayToSort[pos] = arrayToSort[pos - 1];
            RedrawItem(pos);

            arrayToSort[pos - 1] = temp;
            RedrawItem(pos - 1);
            RefreshPanel(pnlSamples);
            if (savePicture)
                SavePicture();
            if (pos > 1)
            {
                pos--;
            }
        }
        Thread.Sleep(speed);
    }
    return arrayToSort;
}

public IList BubbleSort(IList arrayToSort, int left, int right)
{

    for (int i = left; i < right; i++)
    {
        for (int j = right; j > i; j--)
        {
            if (((IComparable)arrayToSort[j - 1]).CompareTo(arrayToSort[j]) > 0)
            {
                object temp = arrayToSort[j - 1];
                arrayToSort[j - 1] = arrayToSort[j];
                RedrawItem(j - 1);
                arrayToSort[j] = temp;
                RedrawItem(j);
                RefreshPanel(pnlSamples);
                if (savePicture)
                    SavePicture();
                Thread.Sleep(speed);
            }

        }

    }
```

```
    return arrayToSort;
}

public IList QuickSortWithBubbleSort(IList a, int left, int right)
{
    int i = left;
    int j = right;

    Thread.Sleep(speed);
    if (right - left <= 6)
    {
        BubbleSort(a, left, right);
        return a;
    }

    double pivotValue = ((left + right) / 2);
    int x = (int)a[int.Parse(pivotValue.ToString())];

    a[(left + right) / 2] = a[right];
    a[right] = x;
    RedrawItem(right);
    RefreshPanel(pnlSamples);
    if (savePicture)
        SavePicture();

    while (i <= j)
    {
        while (((IComparable)a[i]).CompareTo(x) < 0)
        {
            i++;
        }
        while (((IComparable)x).CompareTo(a[j]) < 0)
        {
            j--;

        }

        if (i <= j)
        {
            object temp = a[i];
            a[i++] = a[j];
            RedrawItem(i - 1);
            a[j--] = temp;
            RedrawItem(j + 1);
            RefreshPanel(pnlSamples);
            if (savePicture)
                SavePicture();
        }
        Thread.Sleep(speed);
    }
    if (left < j)
    {
        QuickSortWithBubbleSort(a, left, j);
    }
    if (i < right)
    {
        QuickSortWithBubbleSort(a, i, right);
    }

    return a;
}
```

```
public IList BucketSort(IList arrayToSort)
{
    if (arrayToSort == null || arrayToSort.Count == 0) return arrayToSort;

    object max = arrayToSort[0];
    object min = arrayToSort[0];


    for (int i = 0; i < arrayToSort.Count; i++)
    {

        if (((IComparable)arrayToSort[i]).CompareTo(max) > 0)
        {
            max = arrayToSort[i];
        }

        if (((IComparable)arrayToSort[i]).CompareTo(min) < 0)
        {
            min = arrayToSort[i];
        }
        Thread.Sleep(speed);
    }
    ArrayList[] holder = new ArrayList[(int)max - (int)min + 1];

    for (int i = 0; i < holder.Length; i++)
    {
        holder[i] = new ArrayList();
    }

    for (int i = 0; i < arrayToSort.Count; i++)
    {
        holder[(int)arrayToSort[i] - (int)min].Add(arrayToSort[i]);
    }

    int k = 0;

    for (int i = 0; i < holder.Length; i++)
    {
        if (holder[i].Count > 0)
        {
            for (int j = 0; j < holder[i].Count; j++)
            {
                arrayToSort[k] = holder[i][j];
                RedrawItem(k);
                RefreshPanel(pnlSamples);
                if (savePicture)
                    SavePicture();
                k++;
                Thread.Sleep(speed);
            }
        }
    }

    return arrayToSort;
}

public IList CycleSort(IList arrayToSort)
{
    try
    {
        int writes = 0;
        for (int cycleStart = 0; cycleStart < arrayToSort.Count; cycleStart++)
        {
```

```
            object item = arrayToSort[cycleStart];
            int pos = cycleStart;

            do
            {
                int to = 0;
                for (int i = 0; i < arrayToSort.Count; i++)
                {
                    if (i != cycleStart && ((IComparable)arrayToSort[i]).CompareTo(item) < 0)
                    {
                        to++;
                    }

                }
                if (pos != to)
                {
                    while (pos != to && ((IComparable)item).CompareTo(arrayToSort[to]) == 0)
                    {
                        to++;
                        //Thread.Sleep(100);
                    }

                    object temp = arrayToSort[to];
                    lock (this)
                    {
                        arrayToSort[to] = item;
                    }
                    RedrawItem(to);
                    item = temp;
                    RedrawItem(cycleStart);
                    //Thread.Sleep(100);
                    RefreshPanel(pnlSamples);
                    if (savePicture)
                        SavePicture();

                    writes++;
                    pos = to;
                }
                Thread.Sleep(speed);

            } while (cycleStart != pos);
        }
    }
    catch (Exception err)
    {
        string errr = err.Message;
    }
    return arrayToSort;
}

public IList OddEvenSort(IList arrayToSort)
{
    bool sorted = false;
    while (!sorted)
    {
        sorted = true;
        for (var i = 1; i < arrayToSort.Count - 1; i += 2)
        {
            if (((IComparable)arrayToSort[i]).CompareTo(arrayToSort[i + 1]) > 0)
            {
                object temp = arrayToSort[i];
                arrayToSort[i] = arrayToSort[i + 1];
                RedrawItem(i);
```

```
                arrayToSort[i + 1] = temp;
                RedrawItem(i + 1);
                RefreshPanel(pnlSamples);
                if (savePicture)
                    SavePicture();
                sorted = false;
            }
            Thread.Sleep(speed);
        }

        for (var i = 0; i < arrayToSort.Count - 1; i += 2)
        {
            if (((IComparable)arrayToSort[i]).CompareTo(arrayToSort[i + 1]) > 0)
            {
                object temp = arrayToSort[i];
                arrayToSort[i] = arrayToSort[i + 1];
                arrayToSort[i + 1] = temp;
                RedrawItem(i);
                RedrawItem(i + 1);
                RefreshPanel(pnlSamples);
                if (savePicture)
                    SavePicture();
                sorted = false;
            }
            Thread.Sleep(speed);
        }


    }
    return arrayToSort;
}

public IList PigeonHoleSort(IList list)
{
    object min = list[0], max = list[0];
    foreach (object x in list)
    {
        if (((IComparable)min).CompareTo(x) > 0)
        {
            min = x;
        }
        if (((IComparable)max).CompareTo(x) < 0)
        {
            max = x;
        }
        Thread.Sleep(speed);
    }

    int size = (int)max - (int)min + 1;

    int[] holes = new int[size];

    foreach (int x in list)
        holes[x - (int)min]++;

    int i = 0;
    for (int count = 0; count < size; count++)
        while (holes[count]-- > 0)
        {

            list[i] = count + (int)min;
            RedrawItem(i);
            i++;
```

```
                RefreshPanel(pnlSamples);
                Thread.Sleep(speed);
            }
        return list;
    }

    private void RedrawItem(int index1)
    {
        lock (this)
        {
            int x1 = (int)(((double)pnlSamples.Width / arrayToSort.Count) * index1);

            g.DrawLine(new Pen(Color.White), new Point(x1, 0), new Point(x1, pnlSamples.Height));
            g.DrawLine(new Pen(Color.Red), new Point(x1, pnlSamples.Height), new Point(x1,
(int)(pnlSamples.Height - (int)arrayToSort[index1])));
            g.DrawLine(new Pen(Color.Black), new Point(x1, pnlSamples.Height), new Point(x1,
(int)(pnlSamples.Height - (int)arrayToSort[index1])));
        }
    }

    private void SavePicture()
    {
      /* ImageCodecInfo myImageCodecInfo = this.getEncoderInfo("image/gif");
        EncoderParameter myEncoderParameter = new
EncoderParameter(System.Drawing.Imaging.Encoder.Compression,
(long)EncoderValue.CompressionLZW);
        EncoderParameter qualityParam = new
EncoderParameter(System.Drawing.Imaging.Encoder.Quality, 0L);
        EncoderParameters myEncoderParameters = new EncoderParameters(1);

        EncoderParameters encoderParams = new EncoderParameters(2);
        encoderParams.Param[0] = qualityParam;
        encoderParams.Param[1] = myEncoderParameter;

        string destPath = System.IO.Path.Combine(outputFolder, outputFile + imgCount + ".gif");
        //bmpsave.Save(destPath, myImageCodecInfo, encoderParams);
        imgCount++;*/
    }

    private ImageCodecInfo getEncoderInfo(string mimeType)
    {
        ImageCodecInfo[] codecs = ImageCodecInfo.GetImageEncoders();
        for (int i = 0; i < codecs.Length; i++)
            if (codecs[i].MimeType == mimeType)
                return codecs[i];
        return null;
    }

    delegate void SetControlValueCallback(Control pnlSort);

    private void RefreshPanel(Control pnlSort)

        if (pnlSort.InvokeRequired)
        {
            SetControlValueCallback d = new SetControlValueCallback(RefreshPanel);
            pnlSort.Invoke(d, new object[] { pnlSort });
        }
        else
        {
            pnlSort.Refresh();
        }
    }
```

```
        private void DrawSamples()
        {
            g.Clear(Color.White);

            if (this.arrayToSort != null)
            {
                for (int i = 0; i < this.arrayToSort.Count; i++)
                {
                    int x = (int)(((double)pnlSamples.Width / arrayToSort.Count) * i);
                    Pen pen = new Pen(Color.Black);
                    g.DrawLine(pen, new Point(x, pnlSamples.Height), new Point(x,
(int)(pnlSamples.Height - (int)arrayToSort[i])));
                }

            }
        }

        public void SortSelect (ComboBox cmbbx)
        {
            if (cmbbx.SelectedIndex == 0)
                this.eidos = 0;
            else if (cmbbx.SelectedIndex == 1)
                this.eidos = 1;
            else if (cmbbx.SelectedIndex == 2)
                this.eidos = 2;
            else if (cmbbx.SelectedIndex == 3)
                this.eidos = 3;
            else if (cmbbx.SelectedIndex == 4)
                this.eidos = 4;
            else if (cmbbx.SelectedIndex == 5)
                this.eidos = 5;
            else if (cmbbx.SelectedIndex == 6)
                this.eidos = 6;
            else if (cmbbx.SelectedIndex == 7)
                this.eidos = 7;
            else if (cmbbx.SelectedIndex == 8)
                this.eidos = 8;
            else if (cmbbx.SelectedIndex == 9)
                this.eidos = 9;
            else if (cmbbx.SelectedIndex == 10)
                this.eidos = 10;
            else if (cmbbx.SelectedIndex == 11)
                this.eidos = 11;
            else if (cmbbx.SelectedIndex == 12)
                this.eidos = 12;
            else if (cmbbx.SelectedIndex == 13)
                this.eidos = 13;
            else if (cmbbx.SelectedIndex == 14)
                this.eidos = 14;
        }

        public int GetEidos()
        {
            return this.eidos;
        }

        public string GetTime()
        {
            return Sw.Elapsed.ToString();
        }
    }
}
```

Form1.cs

```
using SortComparison;
using System;
using System.Collections;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Sort_Animator
{
    public partial class Form1 : Form
    {
        Graphics g1;
        ArrayList array1,array2,array3,array4,array5,array6;
        Bitmap bmpsave;
        ArrayList sorter = new ArrayList();

        static Random rng = new Random();

        public Form1()
        {
            InitializeComponent();

            sorter.Add("Bubble Sort");
            sorter.Add("Cocktail Sort");
            sorter.Add("Comb Sort");
            sorter.Add("Insertion Sort");
            sorter.Add("Selection Sort");
            sorter.Add("Counting Sort");
            sorter.Add("Shell Sort");
            sorter.Add("Heap Sort");
            sorter.Add("Merge Sort");
            sorter.Add("Quick Sort");
            sorter.Add("Gnome Sort");
            sorter.Add("Bucket Sort");
            sorter.Add("Cycle Sort");
            sorter.Add("Odd-Even Sort");
            sorter.Add("Pidgeon Hole Sort");

            comboBox1.DataSource = new ArrayList(sorter);
            comboBox2.DataSource = new ArrayList(sorter);
            comboBox3.DataSource = new ArrayList(sorter);
            comboBox4.DataSource = new ArrayList(sorter);
            comboBox5.DataSource = new ArrayList(sorter);
            comboBox6.DataSource = new ArrayList(sorter);

            comboBox2.SelectedIndex = 1;
            comboBox3.SelectedIndex = 2;
            comboBox4.SelectedIndex = 3;
            comboBox5.SelectedIndex = 4;
            comboBox6.SelectedIndex = 6;

        }
        public void Randomize(IList list)
        {
```

```
            for (int i = list.Count - 1; i > 0; i--)
            {
                int swapIndex = rng.Next(i + 1);
                if (swapIndex != i)
                {
                    object tmp = list[swapIndex];
                    list[swapIndex] = list[i];
                    list[i] = tmp;
                }
            }
        }
        private void DrawSamples(Graphics g)
        {
            g.Clear(Color.White);
            //g.Clear(Color.White);

            for (int i = 0; i < array1.Count; i++)
            {
                int x = (int)(((double)pnlSort1.Width / array1.Count) * i);

                Pen pen = new Pen(Color.Black);
                g.DrawLine(pen, new Point(x, pnlSort1.Height), new Point(x, (int)(pnlSort1.Height -
(int)array1[i])));
            }

        }

        private void rng_Click(object sender, EventArgs e)
        {
            bmpsave = new Bitmap(pnlSort1.Width, pnlSort1.Height);

            g1 = Graphics.FromImage(bmpsave);

            pnlSort1.Image = bmpsave;
            pnlSort2.Image = bmpsave;
            pnlSort3.Image = bmpsave;
            pnlSort4.Image = bmpsave;
            pnlSort5.Image = bmpsave;
            pnlSort6.Image = bmpsave;

            array1 = new ArrayList((int)numericUpDownObjects.Value);

            for (int i = 0; i < array1.Capacity; i++)
            {
                //int y = rng.Next(pnlSort1.Height);
                int y = (int)((double)i / array1.Capacity * pnlSort1.Height);
                array1.Add(y);
            }
            Randomize(array1);

            DrawSamples(g1);
        }
/*
        private void RedrawItem(int index1, Graphics g, IList a)
        {
            int x1 = (int)(((double)pnlSort1.Width / array1.Count) * index1);
            g.DrawLine(new Pen(Color.Red), new Point(x1, 0), new Point(x1, pnlSort1.Height));
            g.DrawLine(new Pen(Color.Red), new Point(x1, pnlSort1.Height), new Point(x1,
(int)(pnlSort1.Height - (int)a[index1])));
        }
*/
        private void Sort_Click(object sender, EventArgs e)
        {
```

```
int speed = (int)(numericUpDownSpeed.Value);
//string alg = "";

array2 = new ArrayList(array1);
array3 = new ArrayList(array1);
array4 = new ArrayList(array1);
array5 = new ArrayList(array1);
array6 = new ArrayList(array1);

var sw = new System.Diagnostics.Stopwatch();

SortAlgorithm sa1 = new SortAlgorithm(array1, pnlSort1, true, "", speed, "", -1);
SortAlgorithm sa2 = new SortAlgorithm(array2, pnlSort2, true, "", speed, "", -1);
SortAlgorithm sa3 = new SortAlgorithm(array3, pnlSort3, true, "", speed, "", -1);
SortAlgorithm sa4 = new SortAlgorithm(array4, pnlSort4, true, "", speed, "", -1);
SortAlgorithm sa5 = new SortAlgorithm(array5, pnlSort5, true, "", speed, "", -1);
SortAlgorithm sa6 = new SortAlgorithm(array6, pnlSort6, true, "", speed, "", -1);

sa1.SortSelect(comboBox1);
sa2.SortSelect(comboBox2);
sa3.SortSelect(comboBox3);
sa4.SortSelect(comboBox4);
sa5.SortSelect(comboBox5);
sa6.SortSelect(comboBox6);

ThreadStart ts = delegate()
{
   if (sa1.GetEidos() == 0)
      sa1.BubbleSort(array1);
   else if (sa1.GetEidos() == 1)
      sa1.BiDerectionalBubbleSort(array1);
   else if (sa1.GetEidos() == 2)
      sa1.CombSort(array1);
   else if (sa1.GetEidos() == 3)
      sa1.InsertionSort(array1);
   else if (sa1.GetEidos() == 4)
      sa1.SelectionSort(array1);
   else if (sa1.GetEidos() == 5)
      sa1.CountingSort(array1);
   else if (sa1.GetEidos() == 6)
      sa1.ShellSort(array1);
   else if (sa1.GetEidos() == 7)
      sa1.HeapSort(array1);
   else if (sa1.GetEidos() == 8)
      sa1.MergeSort(array1, 0, array1.Count - 1);
   else if (sa1.GetEidos() == 9)
      sa1.QuickSort(array1, 0, array1.Count - 1);
   else if (sa1.GetEidos() == 10)
      sa1.GnomeSort(array1);
   else if (sa1.GetEidos() == 11)
      sa1.BucketSort(array1);
   else if (sa1.GetEidos() == 12)
      sa1.CycleSort(array1);
   else if (sa1.GetEidos() == 13)
      sa1.OddEvenSort(array1);
   else if (sa1.GetEidos() == 14)
      sa1.PigeonHoleSort(array1);
};
ThreadStart ts2 = delegate()
{
   if (sa2.GetEidos() == 0)
      sa2.BubbleSort(array2);
   else if (sa2.GetEidos() == 1)
```

```
            sa2.BiDerectionalBubbleSort(array2);
        else if (sa2.GetEidos() == 2)
            sa2.CombSort(array2);
        else if (sa2.GetEidos() == 3)
            sa2.InsertionSort(array2);
        else if (sa2.GetEidos() == 4)
            sa2.SelectionSort(array2);
        else if (sa2.GetEidos() == 5)
            sa2.CountingSort(array2);
        else if (sa2.GetEidos() == 6)
            sa2.ShellSort(array2);
        else if (sa2.GetEidos() == 7)
            sa2.HeapSort(array2);
        else if (sa2.GetEidos() == 8)
            sa2.MergeSort(array2, 0, array2.Count - 1);
        else if (sa2.GetEidos() == 9)
            sa2.QuickSort(array2, 0, array2.Count - 1);
        else if (sa2.GetEidos() == 10)
            sa2.GnomeSort(array2);
        else if (sa2.GetEidos() == 11)
            sa2.BucketSort(array2);
        else if (sa2.GetEidos() == 12)
            sa2.CycleSort(array2);
        else if (sa2.GetEidos() == 13)
            sa2.OddEvenSort(array2);
        else if (sa2.GetEidos() == 14)
            sa2.PigeonHoleSort(array2);
    };
    ThreadStart ts3 = delegate()
    {
        if (sa3.GetEidos() == 0)
            sa3.BubbleSort(array3);
        else if (sa3.GetEidos() == 1)
            sa3.BiDerectionalBubbleSort(array3);
        else if (sa3.GetEidos() == 2)
            sa3.CombSort(array3);
        else if (sa3.GetEidos() == 3)
            sa3.InsertionSort(array3);
        else if (sa3.GetEidos() == 4)
            sa3.SelectionSort(array3);
        else if (sa3.GetEidos() == 5)
            sa3.CountingSort(array3);
        else if (sa3.GetEidos() == 6)
            sa3.ShellSort(array3);
        else if (sa3.GetEidos() == 7)
            sa3.HeapSort(array3);
        else if (sa3.GetEidos() == 8)
            sa3.MergeSort(array3, 0, array3.Count - 1);
        else if (sa3.GetEidos() == 9)
            sa3.QuickSort(array3, 0, array3.Count - 1);
        else if (sa3.GetEidos() == 10)
            sa3.GnomeSort(array3);
        else if (sa3.GetEidos() == 11)
            sa3.BucketSort(array3);
        else if (sa3.GetEidos() == 12)
            sa3.CycleSort(array3);
        else if (sa3.GetEidos() == 13)
            sa3.OddEvenSort(array3);
        else if (sa3.GetEidos() == 14)
            sa3.PigeonHoleSort(array3);
    };
    ThreadStart ts4 = delegate()
    {
```

```
    if (sa4.GetEidos() == 0)
        sa4.BubbleSort(array4);
    else if (sa4.GetEidos() == 1)
        sa4.BiDerectionalBubbleSort(array4);
    else if (sa4.GetEidos() == 2)
        sa4.CombSort(array4);
    else if (sa4.GetEidos() == 3)
        sa4.InsertionSort(array4);
    else if (sa4.GetEidos() == 4)
        sa4.SelectionSort(array4);
    else if (sa4.GetEidos() == 5)
        sa4.CountingSort(array4);
    else if (sa4.GetEidos() == 6)
        sa4.ShellSort(array4);
    else if (sa4.GetEidos() == 7)
        sa4.HeapSort(array4);
    else if (sa4.GetEidos() == 8)
        sa4.MergeSort(array4, 0, array4.Count - 1);
    else if (sa4.GetEidos() == 9)
        sa4.QuickSort(array4, 0, array4.Count - 1);
    else if (sa4.GetEidos() == 10)
        sa4.GnomeSort(array4);
    else if (sa4.GetEidos() == 11)
        sa4.BucketSort(array4);
    else if (sa4.GetEidos() == 12)
        sa4.CycleSort(array4);
    else if (sa4.GetEidos() == 13)
        sa4.OddEvenSort(array4);
    else if (sa4.GetEidos() == 14)
        sa4.PigeonHoleSort(array4);
};
ThreadStart ts5 = delegate()
{
    if (sa5.GetEidos() == 0)
        sa5.BubbleSort(array5);
    else if (sa5.GetEidos() == 1)
        sa5.BiDerectionalBubbleSort(array5);
    else if (sa5.GetEidos() == 2)
        sa5.CombSort(array5);
    else if (sa5.GetEidos() == 3)
        sa5.InsertionSort(array5);
    else if (sa5.GetEidos() == 4)
        sa5.SelectionSort(array5);
    else if (sa5.GetEidos() == 5)
        sa5.CountingSort(array5);
    else if (sa5.GetEidos() == 6)
        sa5.ShellSort(array5);
    else if (sa5.GetEidos() == 7)
        sa5.HeapSort(array5);
    else if (sa5.GetEidos() == 8)
        sa5.MergeSort(array5, 0, array5.Count - 1);
    else if (sa5.GetEidos() == 9)
        sa5.QuickSort(array5, 0, array5.Count - 1);
    else if (sa5.GetEidos() == 10)
        sa5.GnomeSort(array5);
    else if (sa5.GetEidos() == 11)
        sa5.BucketSort(array5);
    else if (sa5.GetEidos() == 12)
        sa5.CycleSort(array5);
    else if (sa5.GetEidos() == 13)
        sa5.OddEvenSort(array5);
    else if (sa5.GetEidos() == 14)
        sa5.PigeonHoleSort(array5);
```

```
};
ThreadStart ts6 = delegate()
{
    if (sa6.GetEidos() == 0)
        sa6.BubbleSort(array6);
    else if (sa6.GetEidos() == 1)
        sa6.BiDerectionalBubbleSort(array6);
    else if (sa6.GetEidos() == 2)
        sa6.CombSort(array6);
    else if (sa6.GetEidos() == 3)
        sa6.InsertionSort(array6);
    else if (sa6.GetEidos() == 4)
        sa6.SelectionSort(array6);
    else if (sa6.GetEidos() == 5)
        sa6.CountingSort(array6);
    else if (sa6.GetEidos() == 6)
        sa6.ShellSort(array6);
    else if (sa6.GetEidos() == 7)
        sa6.HeapSort(array6);
    else if (sa6.GetEidos() == 8)
        sa6.MergeSort(array6, 0, array6.Count - 1);
    else if (sa6.GetEidos() == 9)
        sa6.QuickSort(array6, 0, array6.Count - 1);
    else if (sa6.GetEidos() == 10)
        sa6.GnomeSort(array6);
    else if (sa6.GetEidos() == 11)
        sa6.BucketSort(array6);
    else if (sa6.GetEidos() == 12)
        sa6.CycleSort(array6);
    else if (sa6.GetEidos() == 13)
        sa6.OddEvenSort(array6);
    else if (sa6.GetEidos() == 14)
        sa6.PigeonHoleSort(array6);
};
/*
Parallel.Invoke(() =>
{
    ControlThreadingHelper.InvokeControlAction(comboBox1, () =>
    {
        if (comboBox1.SelectedIndex == 0)
            sa1.BubbleSort(array1);
        else if (comboBox1.SelectedIndex == 1)
            sa1.BiDerectionalBubbleSort(array1);
    });
},
() =>
{
    ControlThreadingHelper.InvokeControlAction(comboBox2, () =>
    {
        if (comboBox2.SelectedIndex == 0)
            sa2.BubbleSort(array2);
        else if (comboBox2.SelectedIndex == 1)
            sa2.BiDerectionalBubbleSort(array2);
    });
}
);
Task tsk1 = Task.Factory.StartNew(() =>
{
    ControlThreadingHelper.InvokeControlAction(comboBox1, () =>
    {
        if (comboBox1.SelectedIndex == 0)
            sa1.BubbleSort(array1);
        else if (comboBox1.SelectedIndex == 1)
```

```
                sa1.BiDerectionalBubbleSort(array1);
            });
        });

        Task tsk2 = Task.Factory.StartNew(() =>
        {
            ControlThreadingHelper.InvokeControlAction(comboBox2, () =>
            {
                if (comboBox2.SelectedIndex == 0)
                    sa2.BubbleSort(array2);
                else if (comboBox2.SelectedIndex == 1)
                    sa2.BiDerectionalBubbleSort(array2);
            });
        });
         */

        Thread thr1 = new Thread(ts);
        thr1.Start();
        Thread thr2 = new Thread(ts2);
        thr2.Start();
        Thread thr3 = new Thread(ts3);
        thr3.Start();
        Thread thr4 = new Thread(ts4);
        thr4.Start();
        Thread thr5 = new Thread(ts5);
        thr5.Start();
        Thread thr6 = new Thread(ts6);
        thr6.Start();

        label1.Text = sa1.GetTime();
    }
  }
}
```