ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΟΠΤΙΚΟΠΟΙΗΣΗ ΑΛΓΟΡΙΘΜΩΝ ΤΑΞΙΝΟΜΗΣΗΣ

ΓΕΩΡΓΙΟΣ ΒΑΝΑΣΑΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ

Δαδαλιάρης Αντώνιος

Επίκουρος Καθηγητής

Λαμία 2025

«Με ατομική μου ευθύνη και γνωρίζοντας τις κυρώσεις [1], που προβλέπονται από της διατάξεις της παρ. 6 του άρθρου 22 του Ν. 1599/1986, δηλώνω ότι:

1. Δεν παραθέτω κομμάτια βιβλίων ή άρθρων ή εργασιών άλλων αυτολεξεί **χωρίς να τα περικλείω σε εισαγωγικά** και χωρίς να αναφέρω το συγγραφέα, τη χρονολογία, τη σελίδα. Η αυτολεξεί παράθεση χωρίς εισαγωγικά χωρίς αναφορά στην πηγή, είναι λογοκλοπή. Πέραν της αυτολεξεί παράθεσης, λογοκλοπή θεωρείται και η παράφραση εδαφίων από έργα άλλων, συμπεριλαμβανομένων και έργων συμφοιτητών μου, καθώς και η παράθεση στοιχείων που άλλοι συνέλεξαν ή επεξεργάσθηκαν, χωρίς αναφορά στην πηγή. Αναφέρω πάντοτε με πληρότητα την πηγή κάτω από τον πίνακα ή σχέδιο, όπως στα παραθέματα.
2. Δέχομαι ότι η αυτολεξεί **παράθεση χωρίς εισαγωγικά**, ακόμα κι αν συνοδεύεται από αναφορά στην πηγή σε κάποιο άλλο σημείο του κειμένου ή στο τέλος του, είναι αντιγραφή. Η αναφορά στην πηγή στο τέλος π.χ. μιας παραγράφου ή μιας σελίδας, δεν δικαιολογεί συρραφή εδαφίων έργου άλλου συγγραφέα, έστω και παραφρασμένων, και παρουσίασή τους ως δική μου εργασία.
3. Δέχομαι ότι υπάρχει επίσης περιορισμός στο μέγεθος και στη συχνότητα των παραθεμάτων που μπορώ να εντάξω στην εργασία μου εντός εισαγωγικών. Κάθε μεγάλο παράθεμα (π.χ. σε πίνακα ή πλαίσιο, κλπ), προϋποθέτει ειδικές ρυθμίσεις, και όταν δημοσιεύεται προϋποθέτει την άδεια του συγγραφέα ή του εκδότη. Το ίδιο και οι πίνακες και τα σχέδια
4. Δέχομαι όλες τις συνέπειες σε περίπτωση λογοκλοπής ή αντιγραφής.

Ημερομηνία: 03/11/2025

Ο Δηλών

(1) «Όποιος εν γνώσει του δηλώνει ψευδή γεγονότα ή αρνείται ή αποκρύπτει τα αληθινά με έγγραφη υπεύθυνη δήλωση του άρθρου 8 παρ. 4 Ν. 1599/1986 τιμωρείται με φυλάκιση τουλάχιστον τριών μηνών. Εάν ο υπαίτιος αυτών των πράξεων σκόπευε να προσπορίσει στον εαυτόν του ή σε άλλον περιουσιακό όφελος βλάπτοντας τρίτον ή σκόπευε να βλάψει άλλον, τιμωρείται με κάθειρξη μέχρι 10 ετών.»

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΚΕΦΑΛΑΙΟ 1 ΕΙΣΑΓΩΓΗ

Η ταξινόμηση δεδομένων είναι ένα πολυσυζητημένο θέμα στον τομέα του προγραμματισμού και έχει απασχολήσει ένα ευρύ κοινό. Υπάρχει ένας πολύ μεγάλος αριθμός αλγορίθμων που καθιστούν εφικτή αυτή τη διαδικασία, ωστόσο στην παρούσα εργασία η μελέτη επικεντρώνεται σε δέκα αλγορίθμους από αυτούς. Συγκεκριμένα, αναφερόμαστε στους Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort, Heap Sort, Counting Sort, Radix Sort, Bucket Sort και Shell Sort. Ο κάθε αλγόριθμος είναι μοναδικός, τόσο στο πως είναι γραμμένος σε κώδικα αλλά και σε τι τύπο δεδομένων απευθύνεται, τι χώρο καταναλώνει, τι χρόνο απαιτεί και πόσο σταθερός είναι.

Για να γίνει κατανοητή η εργασία σε όσο το δυνατόν περισσότερο κόσμο γίνεται, ασχέτως προσωπικής γνώσης που μπορεί να έχει επί του θέματος, αναπτύχθηκε ένα πρόγραμμα σε Python που συμπεριλαμβάνει όλους τους αλγορίθμους ταξινόμησης που προαναφέρθηκαν, με ενσωμάτωση GUI ώστε να γίνει δυνατή η εισαγωγή του μεγέθους πίνακα από τον χρήστη, το ελάχιστο και μέγιστο στοιχείο του πίνακα, η επιλογή διπλότυπων αριθμών και τι περίπτωση θέλει να παρατηρήσει (ήδη ταξινομημένο πίνακα, τυχαίο πίνακα, αντιστρόφως ταξινομημένο πίνακα).

Επίσης, προστέθηκε γραφικό περιβάλλον μέσω του οποίου γίνεται εμφανής η πορεία κάθε ενός από τους δέκα αλγορίθμους, ώστε να γίνει κατανοητός ο τρόπος που λειτουργεί. Όπως θα παρουσιαστεί αναλυτικά στη συνέχεια, παρέχεται η δυνατότητα παρακολούθησης της πορείας του κάθε αλγορίθμου από την στιγμή που θα γίνει η εισαγωγή του πίνακα ακεραίων αριθμών από τον χρήστη, μέχρι την στιγμή που ο πίνακας θα είναι πλήρως ταξινομημένος. Θα είναι έτσι εφικτό να φανούν όλα τα βήματα που πραγματοποίησε κάθε αλγόριθμος ώστε να ταξινομήσει τον πίνακα, αλλά και τις αλλαγές στοιχείων που έγιναν αναλυτικά. Θα παρέχονται πειραματικά αποτελέσματα από εξαγωγή δεδομένων μέσω γραφικού περιβάλλοντος για τον συγκεκριμένο σκοπό.

Θα είναι εφικτή η παρατήρηση της απόδοσης κάθε αλγορίθμου βάσει διαφορετικών εισόδων του χρήστη, ώστε να παρουσιαστεί η λειτουργία του στην εκάστοτε περίπτωση. Στην πραγματικότητα, δεν υπάρχει ένας αλγόριθμος που είναι βέλτιστος για κάθε λειτουργία, η απόδοση του εξαρτάται από διάφορους παράγοντες, οι οποίοι θα αναλυθούν καλύτερα στην πορεία. Σκοπός της εργασίας είναι η παρουσίαση των πλεονεκτημάτων και των μειονεκτημάτων κάθε αλγορίθμου, καθώς και η εμφάνιση της ολοκληρωτικής του συμπεριφοράς. Έτσι θα υπάρχει μια εμπεριστατωμένη εικόνα της ταυτότητας κάθε αλγορίθμου και της αποδοτικότητας του σε συγκεκριμένο τύπο δεδομένων.

1.1 Σκοπός της Εργασίας

Ο κύριος σκοπός της εργασίας είναι η παρατήρηση των δέκα αλγορίθμων ταξινόμησης από κάθε όψη. Συγκεκριμένα, θα γίνει ανάλυση του θεωρητικού τους υπόβαθρου ώστε να γίνουν σαφή τα χαρακτηριστικά και οι πληροφορίες του καθενός, αλλά παράλληλα θα γίνει εξέταση και στην πράξη, δηλαδή με την εκτέλεση της εφαρμογής, ώστε να επιβεβαιωθεί η θεωρία τους με πραγματικά παραδείγματα.

Στην εργασία θα παρουσιαστεί η αποδοτικότητα των αλγορίθμων σε συγκεκριμένο τύπο δεδομένων, συνθήκες και κριτήρια. Θα γίνει εμφανής η οπτικοποίηση με ειδικό γράφημα και μέσω των πειραμάτων θα γίνει πιο κατανοητή η συμπεριφορά τους.

Θα μελετηθεί η συμπεριφορά κάθε αλγορίθμου, από τον πιο σύνθετο μέχρι τον πιο απλό, σε συγκεκριμένα δεδομένα. Η εργασία παρέχει ένα ενδιαφέρον και ολοκληρωμένο περιβάλλον αναπαράστασης της πορείας εκτέλεσης των αλγορίθμων.

1.2 Δομή της Εργασίας

Η παρούσα εργασία έχει οργανωθεί σε πέντε κεφάλαια, κάθε κεφάλαιο εξετάζει και δείχνει διαφορετικά κομμάτια του θέματος.

Στο Κεφάλαιο 1 γίνεται η εισαγωγή στο θέμα, δηλαδή η περιγραφή του σκοπού της εργασίας και η παρουσίαση της δομής που θα ακολουθήσει.

Στο Κεφάλαιο 2 υπάρχει αναφορά στο θεωρητικό υπόβαθρο των αλγορίθμων ταξινόμησης, με επεξήγηση των διάφορων χαρακτηριστικών - ιδιοτήτων που έχουν, αλλά και της κατηγορίας τους.

Στο Κεφάλαιο 3 γίνεται περιγραφή των τεχνολογιών και των εργαλείων που χρησιμοποιήθηκαν για την ανάπτυξη της εφαρμογής ώστε να ολοκληρωθεί με επιτυχία.

Στο Κεφάλαιο 4 είναι η παρουσίαση των αποτελεσμάτων από τα πειράματα που έγιναν και υπάρχει λεπτομερής σύγκριση της απόδοσης των αλγορίθμων σε διαφορετικά σενάρια.

Στο Κεφάλαιο 5 γίνεται αναφορά των συμπερασμάτων που εξήχθησαν συνολικά από την παρούσα εργασία και την προσωπική τοποθέτηση επί του θέματος.

ΚΕΦΑΛΑΙΟ 2 ΒΙΒΛΙΟΓΡΑΦΙΚΗ ΕΠΙΣΚΟΠΗΣΗ

2.1 Ανάλυση Αλγορίθμου

Στην ανάλυση των αλγορίθμων εξετάζεται ο τρόπος που εκτελείται ένας αλγόριθμος από την μεριά κατανάλωσης πόρων και υπολογιστικής ισχύος. Με τον όρο "πόροι" στην ουσία αναφερόμαστε στον χρόνο που εκτελείται το πρόγραμμα, την μνήμη δηλαδή που καταλαμβάνει στο σύστημα ώστε να εκτελεστεί και να ολοκληρωθεί επιτυχώς. Έτσι υπάρχει μια ξεκάθαρη εικόνα για την πορεία κάθε αλγορίθμου, καθώς γίνεται καλύτερη διάκριση των πιο αποδοτικών αλγορίθμων. Παράδειγμα, ένας αλγόριθμος που απαιτεί πολύ μνήμη από το σύστημα ή κάνει τον περισσότερο χρόνο σε σχέση με τους άλλους για να ταξινομήσει τον πίνακα, δεν θα αποτελούσε καλή επιλογή.

Είναι προφανές ότι δεν υπάρχει βέλτιστος αλγόριθμος που μπορεί άριστα να χειριστεί όλα τα σενάρια που μπορεί ο χρήστης να ζητήσει. Μπορούν πολλοί από τους δέκα αλγορίθμους να πραγματοποιήσουν την επιθυμητή δουλειά, αλλά ο καθένας με διαφορετικό τρόπο, διαφορετικές απαιτήσεις πόρων, διαφορετικό χρόνο για ολοκλήρωση, αλλά όμως το ίδιο αποτέλεσμα. Στην παρούσα εργασία η βαρύτητα θα επικεντρωθεί κυρίως στον χρόνο εκτέλεσης των αλγορίθμων.

Αυτό που επηρεάζει περισσότερο τον χρόνο εκτέλεσης κάθε αλγορίθμου, είναι το μέγεθος του πίνακα, δηλαδή από πόσα στοιχεία αποτελείται. Όσο μεγαλώνει ο πίνακας σε μέγεθος, τόσο περισσότερο χρόνο κάνουν οι αλγόριθμοι ταξινόμησης να ολοκληρώσουν την διαδικασία. Οι περισσότεροι αλγόριθμοι είναι γρήγοροι σε μικρό πίνακα, αλλά όσο μεγαλώνει το μέγεθος του πίνακα, τόσο μεγαλύτερες είναι και οι διαφορές τους.

Για να εξεταστεί αντικειμενικά κάθε αλγόριθμος, [5] θα γίνει χρήση ενός όρου που ονομάζεται "ασυμπτωτική ανάλυση". [1] Έτσι γίνεται πιο εύκολη και δυνατή η εξέταση των αλγορίθμων με μια πιο θεωρητική ματιά, ασχέτως του συστήματος που εκτελούνται ή της σύνδεσης στο διαδίκτυο.

2.2 Πολυπλοκότητα Χρόνου

[6] Η χρονική πολυπλοκότητα κάθε αλγορίθμου είναι ένα πολύ σημαντικό κριτήριο ως προς την αξιολόγηση της αποδοτικότητάς του. Μέσω αυτής, γίνεται φανερό πόσο χρόνο έκανε ο κάθε αλγόριθμος από την στιγμή που έλαβε τον πίνακα ακεραίων αριθμών από το χρήστη, μέχρι την στιγμή που ταξινόμησε τον πίνακα. Επίσης, παρουσιάζεται η σχέση μεταξύ μεγέθους πίνακα και χρόνου, και ότι αυτά τα δυο αυξάνονται αντίστοιχα. Είναι ένας πολύ ρεαλιστικός τρόπος προσέγγισης καθώς με τον συγκεκριμένο τρόπο δεν έχει σχέση το σύστημα που διαθέτει ο κάθε χρήστης που εκτελεί το πρόγραμμα. Δεν θα ήταν αντικειμενική η εκτέλεση του κώδικα σε ένα πολύ καλό σύστημα, σε σχέση με ένα υποδεέστερο. Τα αποτελέσματα θα ήταν στην πλειοψηφία των περιπτώσεων λανθασμένα. Στην εργασία θα γίνει μελέτη της χρονικής πολυπλοκότητας κάθε αλγορίθμου θεωρητικά, αλλά και πως είναι στην πράξη, δηλαδή κατά την εκτέλεση του προγράμματος.

Worst Case (Αντίστροφα Ταξινομημένος Πίνακας)

[4] Στην χειρότερη περίπτωση δίνεται ο πίνακας που εισάγει ο χρήστης, σε αντίστροφα ταξινομημένη μορφή, δηλαδή από το μεγαλύτερο προς το μικρότερο στοιχείο του πίνακα. Γίνεται επίδειξη της πορείας κάθε αλγορίθμου σε θέμα χρόνου, το οποίο οδηγεί σε επιβεβαίωση ότι οι χρόνοι θα είναι μεγαλύτεροι με διαφορά από κάθε άλλη περίπτωση. Οι αλγόριθμοι δεν μπορούν να είναι πιο αργοί από αυτή την περίπτωση, καθώς κάνουν το μέγιστο των μετακινήσεων στα στοιχεία και γενικά τις περισσότερες ενέργειες για να τον ταξινομήσουν.

Best Case (Ταξινομημένος Πίνακας)

Αντίθετα, στο Best Case δίνεται ο πίνακας σε ταξινομημένη μορφή. Παρατηρείται ότι ο χρόνος εκτέλεσης των αλγορίθμων είναι πολύ χαμηλός, καθώς πραγματοποιείται ο ελάχιστος αριθμός ενεργειών για την ταξινόμησή του. Είναι βέβαια λίγο δύσκολο να υπάρχει περίπτωση όπου ο πίνακας που απαιτείται να ταξινομηθεί είναι ήδη ταξινομημένος και δεν χρειάζεται κάποια ενέργεια.

Average Case (Τυχαία Ταξινομημένος Πίνακας)

Στο Average Case δίνεται ένας τυχαία ταξινομημένος πίνακας, οπότε παρουσιάζεται και το πιο ρεαλιστικό σενάριο. Όπως και στις άλλες δύο περιπτώσεις, εξετάζεται ο χρόνος που απαιτήθηκε από κάθε αλγόριθμο για την ταξινόμηση του πίνακα, ο οποίος βρίσκεται κάπου ανάμεσα στο Worst και το Best Case. Γενικά, όταν απαιτείται να εξεταστεί η πραγματική απόδοση κάποιου αλγορίθμου σε ένα σύνολο δεδομένων, είναι προτιμότερο να χρησιμοποιείται το Average Case, καθώς αποτελεί κατά κύριο λόγο την πιο τυπική περίπτωση ταξινόμησης και απαιτεί αρκετούς υπολογισμούς.

2.3 Ρυθμός Αύξησης των Συναρτήσεων

[7], [9] Η ταχύτητα με την οποία αναπτύσσεται ο χρόνος εκτέλεσης ενός αλγορίθμου αποτελεί το κύριο στοιχείο για τη σύγκρισή του με άλλους αλγορίθμους. Η ασυμπτωτική αποδοτικότητα των δεδομένων αυξάνεται, όσο αυξάνεται το μέγεθος του πίνακα, γι' αυτό είναι σημαντική όταν υπάρχουν μεγάλα μεγέθη εισόδου.

Οι αποδοτικοί αλγόριθμοι όσον αφορά την ασυμπτωτική ανάλυση, συνήθως, είναι καλύτεροι για όλα τα μεγέθη δεδομένων, εκτός από τα αρκετά μικρά μεγέθη. Για την αξιοποίηση της ανάλυσης, χρησιμοποιείται η ασυμπτωτική σημειογραφία, όπως το O Notation που βοηθά στην περιγραφή της συμπεριφοράς των αλγορίθμων με απλό τρόπο.

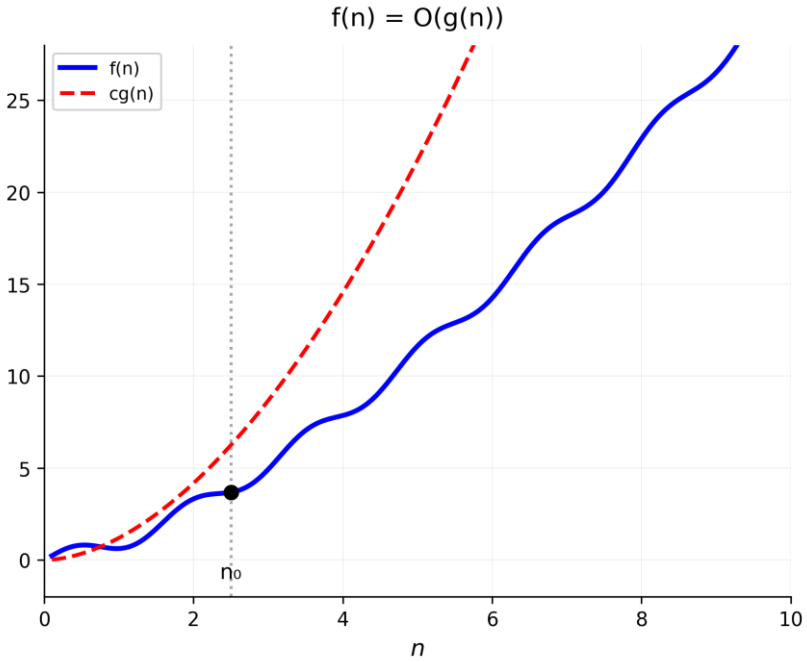
Asymptotic Notation

[14] Για την περιγραφή του πόσο γρήγορος είναι ένας αλγόριθμος, χρησιμοποιούνται ειδικοί συμβολισμοί. Αυτοί δείχνουν πως αυξάνεται ο χρόνος εκτέλεσης όταν ο αλγόριθμος δέχεται μεγαλύτερο όγκο δεδομένων.

Υπάρχουν πέντε τέτοιοι συμβολισμοί στο σύνολο, αλλά στην παρούσα εργασία εξετάζονται μόνο οι τρεις βασικοί: το Big-O, το Big-Ω και το Big-Θ.

Big-O Notation

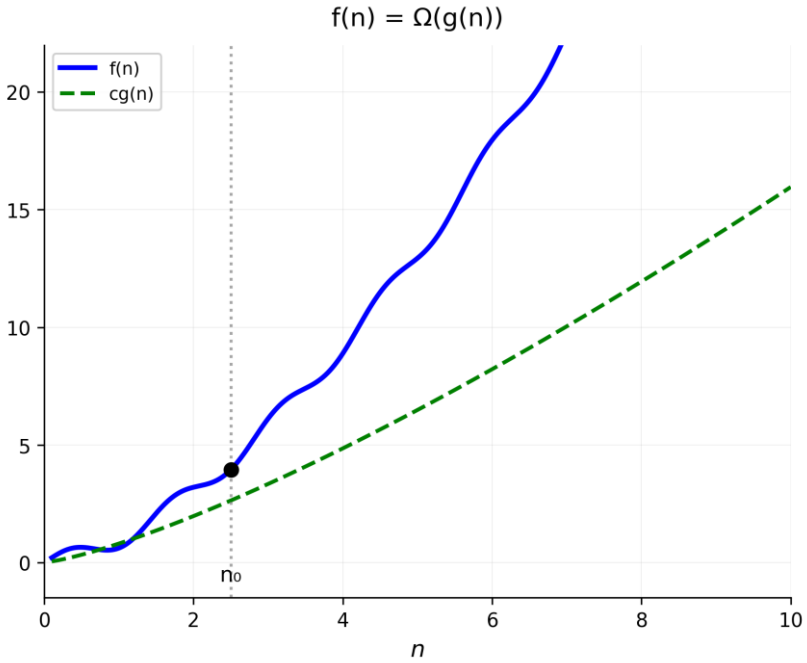
Το Big-O χρησιμοποιείται για την περιγραφή του ανώτερου ορίου μιας συνάρτησης, δηλαδή του μέγιστου αριθμού πόρων που απαιτούνται από έναν αλγόριθμο για την ολοκλήρωση της εκτέλεσής του. Σύμφωνα με τον ορισμό, έστω f(n) και g(n) συναρτήσεις που αντιστοιχούν θετικούς ακέραιους σε θετικούς πραγματικούς αριθμούς. Η f(n) είναι O(g(n)) αν υπάρχει μια πραγματική σταθερά $c > 0$ και μια ακέραια σταθερά $n_o \geq 1$ τέτοια ώστε $f(n) \leq c \cdot g(n)$ για κάθε ακέραιο $n \geq n_o$.



Εικόνα 2.3.Α Γραφική αναπαράσταση Big-O

Big-Ω Notation

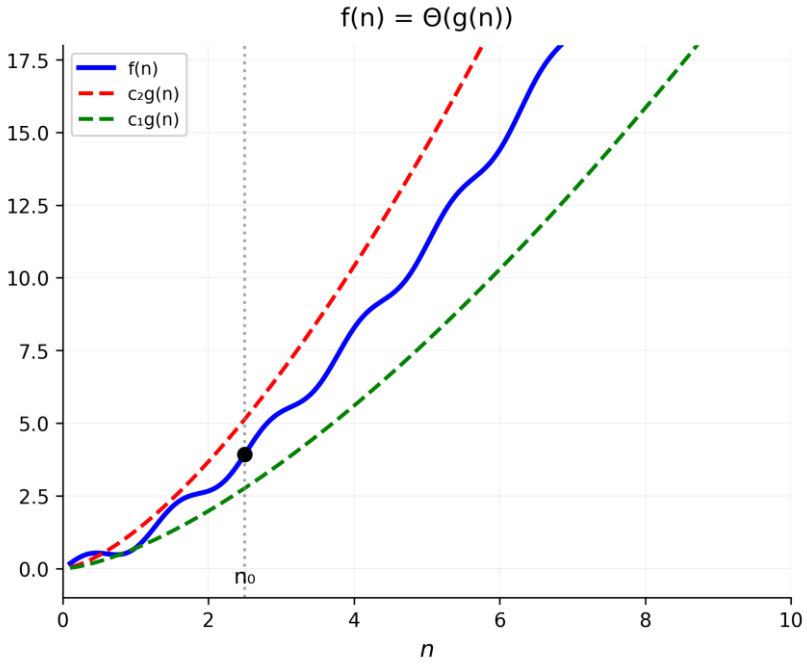
Το Big-Ω χρησιμοποιείται για την περιγραφή του κατώτερου ορίου μιας συνάρτησης, δηλαδή του ελάχιστου αριθμού πόρων που απαιτούνται από έναν αλγόριθμο για την ολοκλήρωση της εκτέλεσής του. Σύμφωνα με τον ορισμό, έστω f(n) και g(n) συναρτήσεις που αντιστοιχούν θετικούς ακέραιους σε θετικούς πραγματικούς αριθμούς. Η f(n) είναι Ω(g(n)) αν υπάρχει μια πραγματική σταθερά $c > 0$ και μια ακέραια σταθερά $n_0 \geq 1$ τέτοια ώστε $f(n) \geq c \cdot g(n)$ για κάθε ακέραιο $n \geq n_0$.



Εικόνα 2.3.Β Γραφική αναπαράσταση Big-Ω

Big-Θ Notation

Το Big-Θ χρησιμοποιείται για την περιγραφή μιας συνάρτησης που έχει τόσο ανώτερο όσο και κατώτερο όριο. Σύμφωνα με τον ορισμό, έστω f(n) και g(n) συναρτήσεις που αντιστοιχούν θετικούς ακέραιους σε θετικούς πραγματικούς αριθμούς. Η f(n) είναι Θ(g(n)) αν και μόνο αν η f(n) είναι O(g(n)) και η f(n) είναι Ω(g(n)).



Εικόνα 2.3.Γ Γραφική αναπαράσταση Big-Θ

Standard Notation

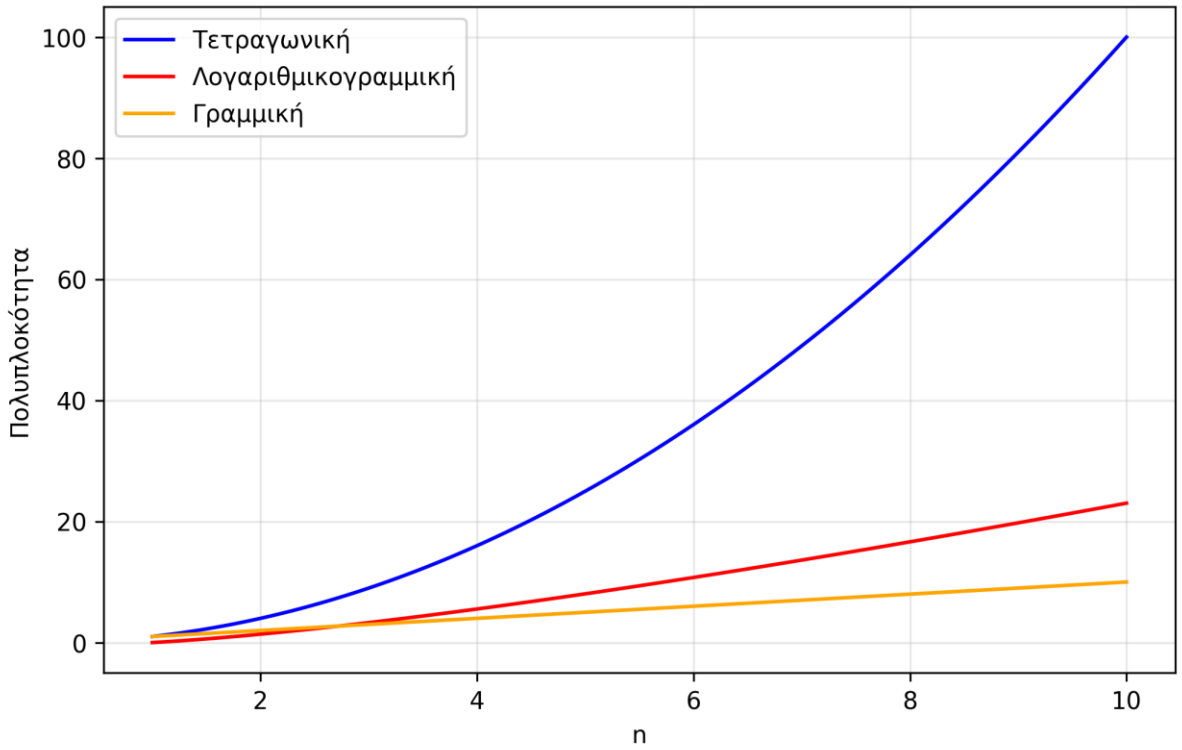
[15] Ο ρυθμός ανάπτυξης των αλγορίθμων ταξινόμησης είναι ο ρυθμός με τον οποίο αυξάνεται το κόστος ενός αλγορίθμου όσο μεγαλώνει το μέγεθος των δεδομένων που απαιτείται να ταξινομήσει. Συγκεκριμένα, στην παρούσα εργασία αναλύονται τρεις βασικοί ρυθμοί ανάπτυξης: ο γραμμικός, ο λογαριθμικογραμμικός και ο τετραγωνικός ρυθμός ανάπτυξης.

Στον γραμμικό ρυθμό ανάπτυξης O(n) ο χρόνος εκτέλεσης κάθε αλγορίθμου αυξάνεται σύμφωνα με το μέγεθος των δεδομένων του. Δηλαδή, αν διπλασιαστούν τα δεδομένα που πρέπει να ταξινομηθούν, τότε διπλασιάζεται και ο χρόνος εκτέλεσης. Σε αυτή την περίπτωση υπάρχει μια σταθερή και προβλέψιμη αύξηση της υπολογιστικής πολυπλοκότητας του αλγορίθμου.

Στον λογαριθμικογραμμικό ρυθμό ανάπτυξης O(nlogn) ο χρόνος εκτέλεσης κάθε αλγορίθμου αυξάνεται λίγο πιο γρήγορα από τον γραμμικό ρυθμό, αλλά εξακολουθεί να παραμένει αποδοτικός σε γενικές γραμμές. Αυτό σημαίνει ότι παρόλο που μεγαλώνουν τα δεδομένα του πίνακα, ο χρόνος εκτέλεσης του αλγορίθμου βρίσκεται σε ικανοποιητικά πλαίσια.

Στον τετραγωνικό ρυθμό ανάπτυξης $O(n^2)$ ο χρόνος εκτέλεσης κάθε αλγορίθμου αυξάνεται πολύ πιο γρήγορα από τους δύο προηγούμενους ρυθμούς ανάπτυξης. Αυτό σημαίνει ότι σε πίνακα με πολλά δεδομένα ο αλγόριθμος καθίσταται τόσο αργός που καταλήγει μη πρακτικός.

Οι ρυθμοί ανάπτυξης παρουσιάζονται καλύτερα στο γράφημα παρακάτω.



Εικόνα 2.3.Δ Γραφική αναπαράσταση των ρυθμών ανάπτυξης αλγορίθμων

2.4 Χαρακτηριστικά Αλγορίθμων Ταξινόμησης

2.4.Α Υπολογιστική Πολυπλοκότητα

[10], [18] Ένας τρόπος για την κατάταξη ενός αλγορίθμου ταξινόμησης, είναι η υπολογιστική του πολυπλοκότητα. Ειδικότερα, η υπολογιστική πολυπλοκότητα δείχνει πόσος χρόνος και πόση μνήμη απαιτήθηκε από έναν αλγόριθμο για την ολοκλήρωση μιας εργασίας. Σαφώς, όσο μεγαλώνουν τα δεδομένα του πίνακα, τόσο αυξάνεται ο χρόνος και η μνήμη. Δεν είναι το ίδιο να ταξινομείται ένας πίνακας με 30 ακεραίους αριθμούς με έναν πίνακα με 500 αριθμούς. Γι' αυτό τον λόγο είναι πολύ χρήσιμη η υπολογιστική πολυπλοκότητα, καθώς δίνεται η δυνατότητα να κατανοηθεί τι είναι καλύτερο για κάθε περίπτωση.

2.4.Β Πολυπλοκότητα Χώρου

[20] Άλλο ένα κριτήριο που μπορεί να βοηθήσει στην καλύτερη κατηγοριοποίηση ενός αλγορίθμου, είναι η πολυπλοκότητα χώρου. Με άλλα λόγια, η χωρική πολυπλοκότητα χωρίζεται σε δύο κατηγορίες: την ταξινόμηση επί τόπου και την ταξινόμηση εκτός τόπου. Στην ταξινόμηση επί τόπου ο αλγόριθμος πρέπει να έχει σταθερό μέγεθος μνήμης, δηλαδή O(1), εκτός από τα στοιχεία που προορίζονται για ταξινόμηση. Στην άλλη περίπτωση, η ταξινόμηση εκτός τόπου χρησιμοποιεί μνήμη ανάλογα με τα δεδομένα που έχει ο πίνακας, δηλαδή η διαδικασία χώρου γίνεται πιο δυναμικά. Αυτός είναι ένας σημαντικός λόγος που καθιστά τους αλγορίθμους που χρησιμοποιούν ταξινόμηση επί τόπου αισθητά πιο αργούς. Γενικά οι αλγόριθμοι που έχουν στη διάθεσή τους περισσότερη μνήμη παρουσιάζουν καλύτερη απόδοση και μπορούν να ταξινομήσουν τα δεδομένα σε μικρότερα χρονικά διαστήματα.

2.4.Γ Σταθερότητα

[19] Το επόμενο κριτήριο που εξετάζεται όσον αφορά τα κριτήρια μέσω των οποίων μπορούν να κατατάσσονται οι αλγόριθμοι σε κατηγορίες, είναι η σταθερότητα που μπορούν να έχουν. Αναλυτικότερα, η σταθερότητα είναι στην ουσία η δυνατότητα διατήρησης της σχετικής σειράς των στοιχείων του πίνακα όταν αυτά έχουν ίσες τιμές. Για να γίνει κατανοητό, όταν δύο στοιχεία έχουν την ίδια τιμή-κλειδί, η σειρά εμφάνισής τους στον αρχικό πίνακα πρέπει να διατηρείται και στον τελικό ταξινομημένο πίνακα. Δηλαδή, αν ένα στοιχείο x εμφανίζεται πριν από ένα άλλο στοιχείο y στην αρχική διάταξη, και έχουν ίδια τιμή, τότε το x θα πρέπει να εμφανίζεται πριν το y και στη ταξινομημένη λίστα. Το αποτέλεσμα είναι μεν ίδιο και στις δύο περιπτώσεις, αλλά στην περίπτωση που ο αλγόριθμος αλλάξει τις τιμές σημαίνει ότι πραγματοποιείται ένα βήμα παραπάνω και καταναλώνεται παραπάνω χρόνος και πόροι.

2.4.Δ Αναδρομικός και μη Αναδρομικός

[16] Ακόμη ένα χαρακτηριστικό των αλγορίθμων ταξινόμησης, είναι το εάν ανήκουν στους αναδρομικούς ή μη αναδρομικούς. Συγκεκριμένα, ένας αναδρομικός αλγόριθμος έχει την ιδιότητα να καλεί τον εαυτό του, αλλά αυτή τη φορά με μικρότερες τιμές εισόδου. Στη συνέχεια εκτελούνται οι πράξεις και εξάγεται το αποτέλεσμα για τις τρέχουσες τιμές εισόδου. Συνήθως το πρόβλημα αυτό μπορεί να βελτιωθεί με τη χρήση λύσεων σε μικρότερες παραλλαγές του ίδιου προβλήματος, οι οποίες στη συνέχεια οδηγούν σταδιακά στην επίλυση του αρχικού προβλήματος. Στην παρούσα εργασία, αναδρομικούς αλγορίθμους αποτελούν οι Quick Sort, Merge Sort, Heap Sort. Από την άλλη πλευρά, ένας μη αναδρομικός αλγόριθμος χρησιμοποιεί επαναληπτικές δομές για την ταξινόμηση των δεδομένων. Το θετικό που έχουν είναι ότι πραγματοποιούν καλύτερη διαχείριση μνήμης αλλά είναι πιο πολύπλοκοι σε υλοποίηση από τους αναδρομικούς αλγορίθμους. Μη αναδρομικοί αλγόριθμοι είναι οι Bubble Sort, Selection Sort, Insertion Sort, Counting Sort, Radix Sort, Bucket Sort και Shell Sort.

2.4.Ε Εσωτερική Ταξινόμηση και Εξωτερική Ταξινόμηση

[13] Εδώ παρουσιάζεται αναλυτικά πως χωρίζονται οι αλγόριθμοι βάσει εσωτερικής και εξωτερικής ταξινόμησης και ποιοι ανήκουν στην κάθε κατηγορία. Η εσωτερική ταξινόμηση είναι ουσιαστικά η διαδικασία κατά την οποία τα δεδομένα ταξινομούνται και είναι αποθηκευμένα στη μνήμη RAM του συστήματος. Αυτό σημαίνει ότι τα δεδομένα βρίσκονται όλα εξ αρχής στη μνήμη του υπολογιστή και μπορούν ανά πάσα ώρα και στιγμή να προσπελαστούν. Όλοι οι δέκα αλγόριθμοι που υπάρχουν στη συγκεκριμένη εργασία ανήκουν σε αυτή την κατηγορία. Από την άλλη, η εξωτερική ταξινόμηση χρησιμοποιείται όταν τα δεδομένα είναι τόσο πολλά που δεν μπορούν να χωρέσουν στη μνήμη του υπολογιστή, οπότε αποθηκεύονται κάπου εξωτερικά π.χ. σκληρός δίσκος, usb stick κλπ.

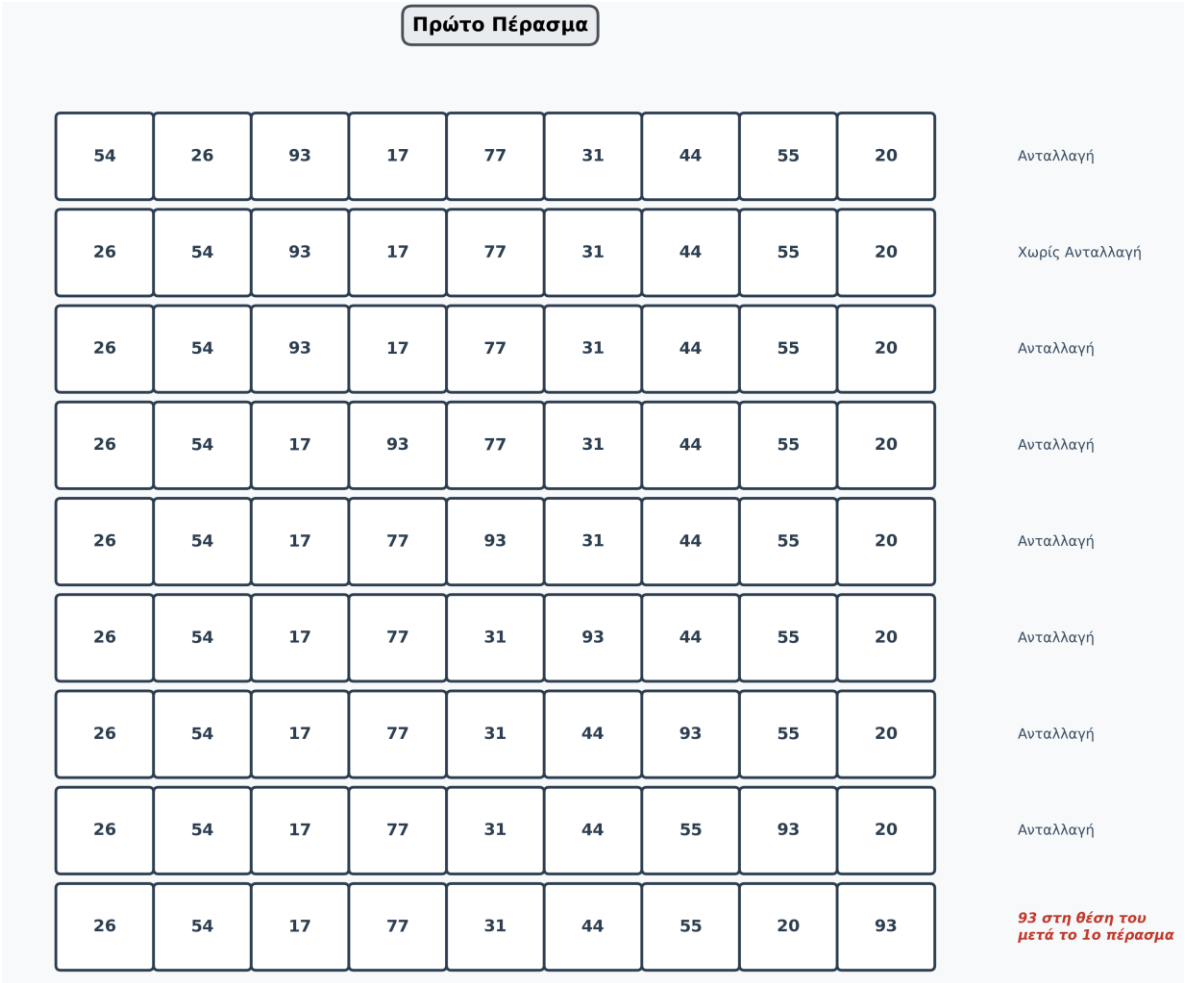
2.5 Θεωρία των Αλγορίθμων Ταξινόμησης

2.5.A Bubble Sort

Περιγραφή Λειτουργίας

[17], [36] Ο Bubble Sort είναι ο πιο αργός αλγόριθμος από όλους που υπάρχουν στην παρούσα εργασία. Η λειτουργία του είναι να συγκρίνει κάθε στοιχείο της λίστας με τα γειτονικά του στοιχεία και να πραγματοποιεί την εναλλαγή που απαιτείται αν δεν βρίσκονται στη σωστή σειρά. Η εκτέλεση του αλγορίθμου σταματάει όταν όλα τα στοιχεία του πίνακα είναι ταξινομημένα και δεν απαιτείται κάποια αλλαγή. Μπορεί λοιπόν να γίνει αντιληπτό ότι σε έναν πολύ μεγάλο πίνακα με μεγάλο όγκο δεδομένων, ο χρόνος που θα απαιτηθεί για την εκτέλεση του αλγορίθμου θα είναι πολύ μεγάλος.

Στην πρώτη επανάληψη του αλγορίθμου, που παρουσιάζεται στην Εικόνα 2.5.Α, παρατηρείται πως πραγματοποιούνται οι συγκρίσεις μεταξύ των γειτονικών στοιχείων. Για έναν πίνακα με n στοιχεία, απαιτείται να πραγματοποιηθούν n-1 συγκρίσεις στην πρώτη διέλευση. Κάτι που διαπιστώνεται είναι ότι το μεγαλύτερο στοιχείο του πίνακα, από τη στιγμή που θα συγκριθεί με το γειτονικό του, μετακινείται σταδιακά προς το τέλος μέχρι να ολοκληρωθεί αυτή η επανάληψη.



Εικόνα 2.5.Α Bubble Sort

Απόδοση

Ο Bubble Sort δεν θα χαρακτηριζόταν ιδιαίτερα καλός σε απόδοση καθώς έχει $O(n^2)$ πολυπλοκότητα σε χρόνο. Τέτοια πολυπλοκότητα ωστόσο έχουν και ο Selection και Insertion Sort, αλλά ο Bubble Sort είναι ακόμη πιο αργός και δεν αποτελεί καλή επιλογή για μεγάλο όγκο δεδομένων. [37] Υπάρχουν ακόμη δύο παραλλαγές του Bubble Sort, ο "Τροποποιημένος Bubble Sort" και ο "Διπλής Κατεύθυνσης Bubble Sort" γνωστός και ως Cocktail Sort. Ο πρώτος ελέγχει αν πραγματοποιήθηκε κάποια αλλαγή στα στοιχεία κατά την ταξινόμηση και αν δεν πραγματοποιήθηκε μπορεί να επιτύχει πολυπλοκότητα O(n). Ο δεύτερος ταξινομεί τον πίνακα προς και τις δύο κατευθύνσεις σε κάθε πέρασμα, μειώνοντας τις συνολικές συγκρίσεις στο σύνολο.

Πλεονεκτήματα και Μειονεκτήματα

Στα πλεονεκτήματα του Bubble Sort αναφέρεται σίγουρα η απλότητα του, το πόσο εύκολα υλοποιείται και η δυνατότητα να διαπιστώνεται αν ο πίνακας είναι ήδη ταξινομημένος. Από την άλλη, στα αρνητικά του αλγορίθμου μπορεί να αναφερθεί ότι δεν είναι κατάλληλος για μεγάλους πίνακες με πολλά δεδομένα για ταξινόμηση, δεν αποτελεί ιδιαίτερα αποτελεσματικό αλγόριθμο ταξινόμησης και επαναλαμβάνει πολλές φορές περιττές συγκρίσεις και ανταλλαγές.

Ψευδοκώδικας

```
1: BUBBLE SORT(A)
2:      for i ← 0 to A.length - 1
3:          for j ← 0 to A.length - 2
4:              if A[j] > A[j+1] then
5:                  temp ← A[j]
6:                  A[j] ← A[j+1]
7:                  A[j+1] ← temp
8: return A
```

2.5.B Selection Sort

Περιγραφή Λειτουργίας

Ο Selection Sort βρίσκει το μικρότερο και μεγαλύτερο στοιχείο του μη ταξινομημένου πίνακα και το ανταλλάσσει με το πρώτο στοιχείο του ταξινομημένου πίνακα. Στη συνέχεια βρίσκεται το επόμενο και ούτω καθεξής. Αποδοτικά βρίσκεται κάπου ανάμεσα από Bubble Sort και Insertion Sort. Τα στοιχεία που ταξινομούνται μεταφέρονται προς τα πάνω στον πίνακα και τα υπόλοιπα παραμένουν κάτω αταξινόμητα, αυτό πραγματοποιείται μέχρι να ταξινομηθεί όλος ο πίνακας. Ο Selection Sort είναι επίσης ένας αλγόριθμος επί τόπου, καθώς απαιτεί σταθερή ποσότητα μνήμης. Όπως και άλλες απλές μεθόδους ταξινόμησης, ο Selection Sort είναι αναποτελεσματικός για μεγάλους πίνακες.

Παρουσιάζεται η λειτουργία του Selection Sort στον παρακάτω πίνακα A = [26, 54, 93, 17, 77, 31, 44, 55, 20]. Συγκεκριμένα, βρίσκεται το μεγαλύτερο στοιχείο του πίνακα σε κάθε επανάληψη και ανταλλάσσεται στην κατάλληλη θέση. Στην πρώτη επανάληψη πραγματοποιείται εναλλαγή του 93 με το 20, στη δεύτερη επανάληψη πραγματοποιείται εναλλαγή 77 με 55 και ούτω καθεξής. Έτσι υπάρχει αύξηση της δεξιάς θέσης κατά ένα στοιχείο σε κάθε επανάληψη μέχρι να ταξινομηθεί πλήρως ο πίνακας.

Εικόνα 2.5.Β Selection Sort

Απόδοση

[21] Σε θέμα απόδοσης ο Selection Sort θα χαρακτηριζόταν καλύτερος από τον Bubble Sort αλλά λόγω των πολλών επαναλήψεων που πραγματοποιεί απαιτούνται n-1 επαναλήψεις για πίνακα με n στοιχεία, καθώς συγκρίνονται όλα τα στοιχεία στην πρώτη επανάληψη έχοντας στο σύνολο $O(n^2)$ συγκρίσεις και n-1 ανταλλαγές για την ταξινόμηση ενός πίνακα ακεραίων αριθμών. Αυτό το γεγονός δεν τον καθιστά κατάλληλο για μεγάλους πίνακες με πολλά δεδομένα. [24] Ωστόσο, υπάρχουν δύο παραλλαγές του Selection Sort, ο Quadratic Sort και ο Tree Sort. Στην πρώτη παραλλαγή ένας πίνακας με 20 στοιχεία χωρίζεται σε 4 υπο-πίνακες με 5 στοιχεία ο καθένας, ταξινομώντας τον κάθε υποπίνακα χωριστά επιτυγχάνεται η ταξινόμηση του αρχικού πίνακα. Στη δεύτερη παραλλαγή Tree Sort χρησιμοποιείται η δομή του Δυαδικού Δέντρου Αναζήτησης, εισάγοντας κάθε στοιχείο στο δέντρο και στη συνέχεια ανακτώντας τα ταξινομημένα μέσω διάσχισης κατά σειρά.

Πλεονεκτήματα και Μειονεκτήματα

Ο Selection Sort δεν είναι ιδιαίτερα καλός και αποδοτικός σε μεγάλα μεγέθη αλλά συγκριτικά με τον Bubble Sort είναι καλύτερος. Είναι εύκολος και απλός και θα χαρακτηριζόταν κατάλληλος για συγκεκριμένες περιπτώσεις.

Ψευδοκώδικας

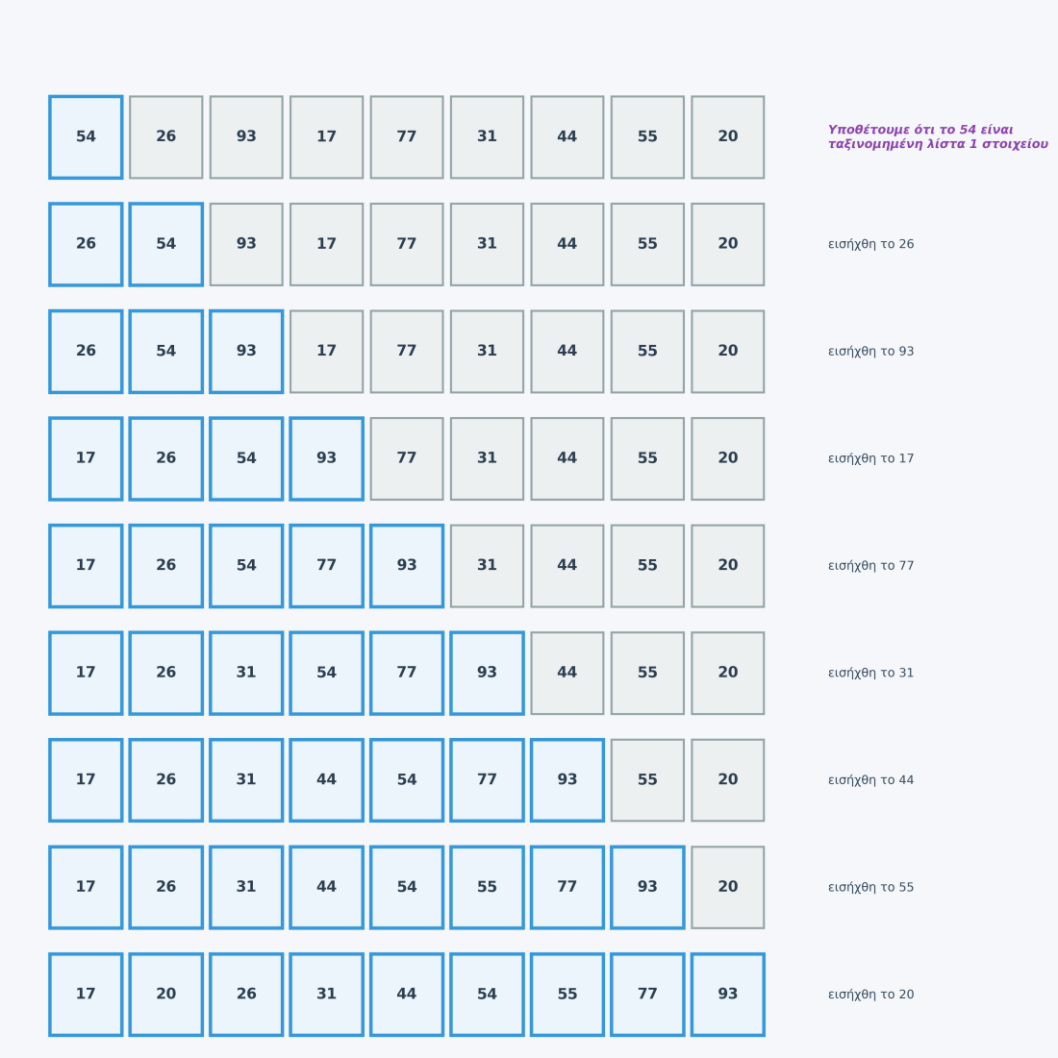
```
1: SELECTION SORT(list)
2:      n ← list.length
3:      for i ← 0 to n-2 do
4:          max_index ← 0
5:          for j ← 1 to n-i-1 do
6:              if list[j] > list[max_index] then
7:                  max_index ← j
8:          temp ← list[max_index]
9:          list[max_index] ← list[n-i-1]
10:         list[n-i-1] ← temp
```

2.5.Γ Insertion Sort

Περιγραφή Λειτουργίας

[35] Ο Insertion Sort λειτουργεί με εισαγωγή κάθε στοιχείου στην κατάλληλη θέση της τελικής ταξινομημένης λίστας, συγκρίνοντάς το με τα γειτονικά του στοιχεία. Αυτή η διαδικασία επαναλαμβάνεται μέχρι να ταξινομηθεί όλος ο πίνακας ακεραίων. Ο Insertion Sort είναι αλγόριθμος επί τόπου και απαιτεί συγκεκριμένο χώρο για να λειτουργήσει και συχνά χρησιμοποιείται ως μέρος πιο εξελιγμένων αλγορίθμων ταξινόμησης.

Παρουσιάζεται η λειτουργία του Insertion Sort στον παρακάτω πίνακα A = [54, 26, 93, 17, 77, 31, 44, 55, 20]. Συγκεκριμένα, λαμβάνεται κάθε στοιχείο ως κλειδί και συγκρίνεται με τα προηγούμενα στοιχεία, τοποθετούμενο στην κατάλληλη θέση. Στην πρώτη επανάληψη το 26 λαμβάνεται ως κλειδί που ανταλλάσσεται με το 54. Στη δεύτερη επανάληψη το 93 παραμένει στη θέση του καθώς είναι μεγαλύτερο από το προηγούμενο. Στην τρίτη επανάληψη το 17 συγκρίνεται με όλα τα προηγούμενα στοιχεία και τοποθετείται στην αρχή του πίνακα. Έτσι, σε κάθε επανάληψη το ταξινομημένο τμήμα αυξάνεται κατά ένα στοιχείο μέχρι να ταξινομηθεί πλήρως ο πίνακας.



Εικόνα 2.5.Γ Insertion Sort

Απόδοση

Ο Insertion Sort είναι επαναληπτικός αλγόριθμος ταξινόμησης και απαιτούνται n-1 επαναλήψεις για n στοιχεία στον πίνακα. Σε χειρότερη περίπτωση, αν ο πίνακας είναι αντίστροφα ταξινομημένος, μπορεί να υπάρχει $O(n^2)$ πολυπλοκότητα, ενώ από την άλλη στην καλύτερη περίπτωση μπορεί να υπάρχει $O(n)$ πολυπλοκότητα. Με μέση πολυπλοκότητα $O(n^2)$, ο Insertion Sort αποτελεί άλλη μια περίπτωση αναποτελεσματικού αλγορίθμου για μεγάλους πίνακες ακεραίων αριθμών. Παραλλαγή του Insertion Sort είναι ο Shell Sort που παρουσιάζεται παρακάτω στην εργασία.

Πλεονεκτήματα και Μειονεκτήματα

Στα πλεονεκτήματα του Insertion Sort υπάρχει η απλότητα του και η υψηλή αποδοτικότητα σε μικρούς πίνακες. Από την άλλη όμως, σε μεγάλα δεδομένα είναι αργός και γι' αυτό συνήθως χρησιμοποιείται μαζί με τον Quick Sort και τον Merge Sort για μεγαλύτερη αποδοτικότητα.

Ψευδοκώδικας

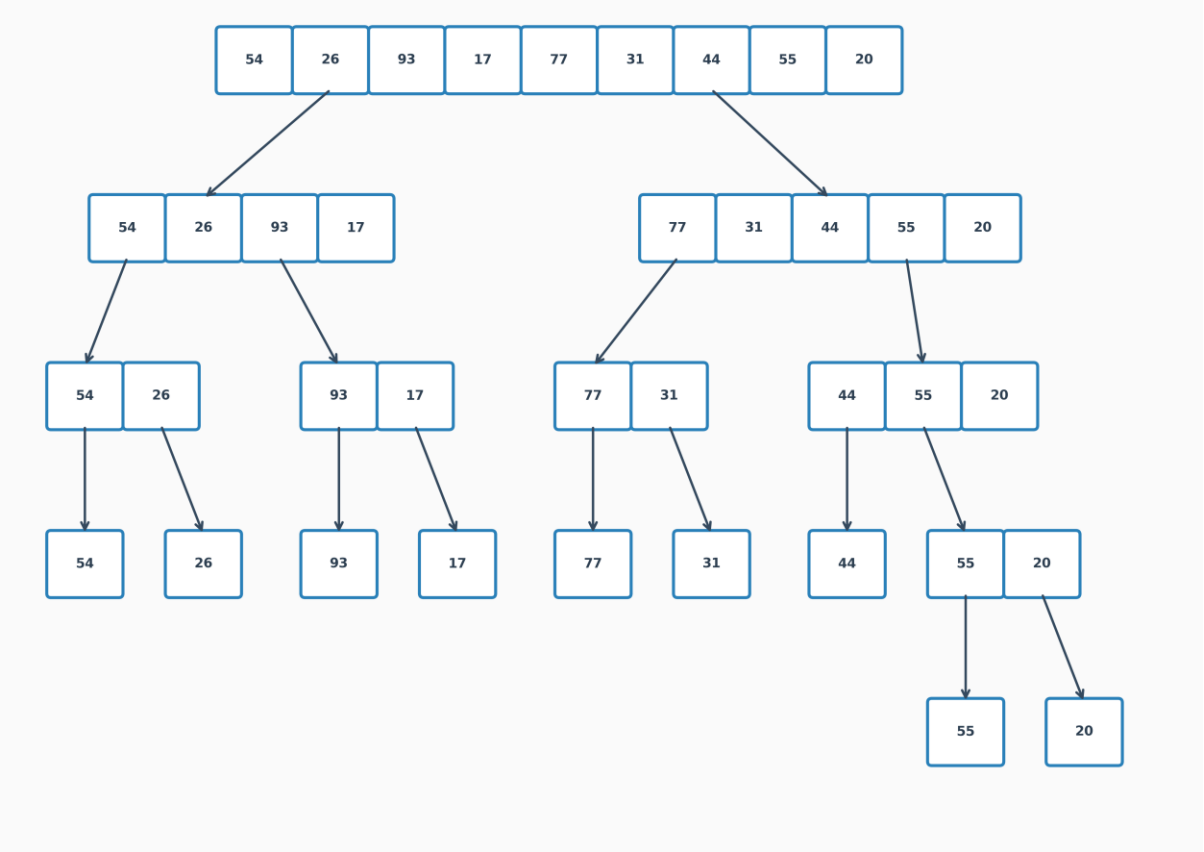
```
1: INSERTION SORT(A)
2:      for j ← 2 to length[A]
3:           key ← A[j]
4:           Insert A[j] into the sorted sequence A[1 . . j - 1]
5:           i ← j - 1
6:           while i > 0 and A[i] > key
7:                A[i + 1] ← A[i]
8:                i ← i - 1
9:           A[i + 1] ← key
```

2.5.Δ Merge Sort

Περιγραφή Λειτουργίας

Ο Merge Sort είναι ένας αλγόριθμος που χρησιμοποιεί τη μέθοδο διαίρει και βασίλευε. [8], [33] Ουσιαστικά χωρίζεται ο πίνακας σε δύο υποπίνακες και ταξινομείται ο κάθε υποπίνακας αναδρομικά και όταν είναι και οι δύο ταξινομημένοι συγχωνεύονται σε ένα ταξινομημένο πίνακα. Η διαδικασία αυτή πραγματοποιείται μέχρι ο κάθε υποπίνακας να έχει μόνο ένα στοιχείο οπότε δεν θα υπάρχει κάποια σύγκριση και εναλλαγή.

Παρουσιάζεται η λειτουργία του Merge Sort στον παρακάτω πίνακα A = [54, 26, 93, 17, 77, 31, 44, 55, 20]. Ο αλγόριθμος χωρίζει επαναληπτικά τον πίνακα στη μέση μέχρι κάθε τμήμα να περιέχει ένα στοιχείο. Στη συνέχεια, συγχωνεύονται τα τμήματα ανά ζεύγη σε ταξινομημένη σειρά μέχρι να προκύψει ο πλήρως ταξινομημένος πίνακας: [17, 20, 26, 31, 44, 54, 55, 77, 93].



Εικόνα 2.5.Δ Merge Sort

Απόδοση

Ο Merge Sort θεωρείται γενικά αποδοτικός αλγόριθμος ταξινόμησης όταν συγκρίνεται με άλλους πιο απλούς. Η πολυπλοκότητα χρόνου εκτέλεσης του αλγορίθμου σε χειρότερη και μέση περίπτωση είναι O(nlogn). Ωστόσο, στην πράξη είναι μη αποδοτικός γιατί απαιτεί διπλάσια έξτρα μνήμη για τον δεύτερο πίνακα σε σχέση με οποιονδήποτε άλλο πιο εξελιγμένο αλγόριθμο. Συχνά χρησιμοποιείται για εξωτερική ταξινόμηση λόγω της δομής του, ωστόσο στην παρούσα εργασία υλοποιείται ως αλγόριθμος εσωτερικής ταξινόμησης που απαιτεί O(n) πρόσθετο χώρο μνήμης για n στοιχεία. Σε χώρους με περιορισμένη μνήμη ο Merge Sort καθίσταται πολύ αναποτελεσματικός.

Πλεονεκτήματα και Μειονεκτήματα

[34] Σοβαρό μειονέκτημα του Merge Sort είναι το γεγονός ότι απαιτεί αρκετή μνήμη για ταξινόμηση πίνακα και δεν συνιστάται για μικρότερους πίνακες επειδή λειτουργεί αναδρομικά και απαιτεί O(n) βοηθητικό χώρο μνήμης. Ωστόσο, ο Merge Sort θεωρείται κατάλληλος για δεδομένα που είναι αποθηκευμένα σε συνδεδεμένη λίστα, επειδή η συγχώνευση δεν απαιτεί τυχαία πρόσβαση στα στοιχεία της λίστας.

Ψευδοκώδικας

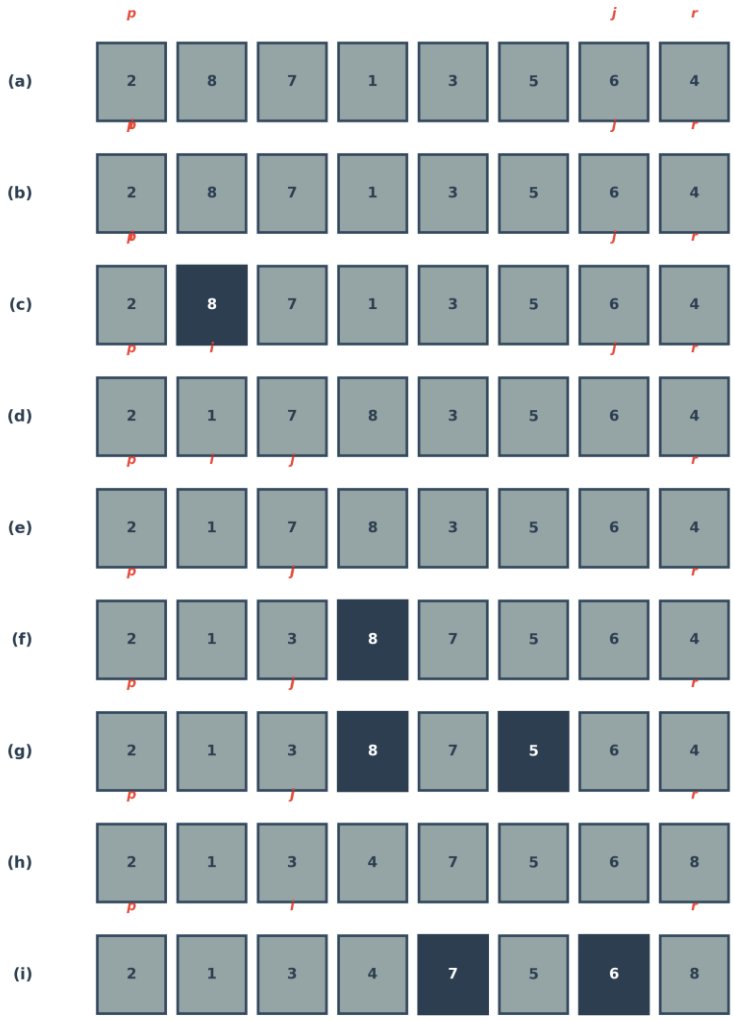
```
1: MERGE_SORT(arr, start, end)
2:     if start < end then
3:         middle ← (start + end) / 2
4:         MERGE_SORT(arr, start, middle)
5:         MERGE_SORT(arr, middle + 1, end)
6:         MERGE(arr, start, middle, middle + 1, end)
7.
8: MERGE(arr, start1, end1, start2, end2)
9:     temp ← array of same size as arr
10:    i ← start1
11:    j ← start2
12:    k ← start1
13:    while i ≤ end1 and j ≤ end2 do
14:        if arr[i] ≤ arr[j] then
15:            temp[k] ← arr[i]
16:            i ← i + 1
17:            k ← k + 1
18:        else
19:            temp[k] ← arr[j]
20:            j ← j + 1
21:            k ← k + 1
22:    while i ≤ end1 do
23:        temp[k] ← arr[i]
24:        i ← i + 1
25:        k ← k + 1
26:    while j ≤ end2 do
27:        temp[k] ← arr[j]
28:        j ← j + 1
29:        k ← k + 1
30:    for i ← start1 to end2 do
31:        arr[i] ← temp[i]
```

2.5.E Quick Sort

Περιγραφή Λειτουργίας

[22] Ο Quick Sort είναι ένας αλγόριθμος που βασίζεται στη μέθοδο "διαίρει και βασίλευε". Επιλέγεται ένα στοιχείο ως "pivot" (άξονας) και αναδιοργανώνεται η λίστα έτσι ώστε τα μικρότερα στοιχεία να βρίσκονται αριστερά του pivot και τα μεγαλύτερα δεξιά. Στη συνέχεια, εφαρμόζεται αναδρομικά η ίδια διαδικασία στις δύο υπολίστες. Η διαδικασία συνεχίζεται μέχρι κάθε υπολίστα να έχει μόνο ένα στοιχείο.

Όπως παρατηρείται στην εικόνα 2.5.E, υπάρχει ένας πίνακας με στοιχεία 2, 8, 7, 1, 3, 5, 6, 4. Απαιτείται να ταξινομηθεί ο πίνακας σε αύξουσα σειρά και βρίσκεται το μεγαλύτερο στοιχείο σε κάθε επανάληψη και ανταλλάσσεται με το στοιχείο της κατάλληλης θέσης. Στην πρώτη επανάληψη βρίσκεται το 7 ως μεγαλύτερο και τοποθετείται δεξιά του 4. Στη δεύτερη επανάληψη υπάρχει ως μεγαλύτερο το 3 και ανταλλάσσεται με τη δεύτερη θέση από τα δεξιά. Σε κάθε επανάληψη δηλαδή η θέση από δεξιά μετακινείται μία θέση αριστερά και συνεχίζεται αυτή η διαδικασία μέχρι να ταξινομηθεί όλος ο πίνακας.



Εικόνα 2.5.E Quick Sort

Απόδοση

Ο Quick Sort είναι από τους ταχύτερους αλγορίθμους ταξινόμησης, με μέση πολυπλοκότητα χρόνου O(n log n). Ωστόσο, όταν το pivot επιλέγεται σταθερά ως το πρώτο ή το τελευταίο στοιχείο, ένας ήδη ταξινομημένος πίνακας οδηγεί στη χειρότερη περίπτωση λειτουργίας. Σε αυτή την περίπτωση, ο αλγόριθμος εκτελείται σε τετραγωνικό χρόνο $O(n^2)$. Στον Quick Sort υπάρχει μια παραλλαγή με όνομα Qsort στην οποία μπορεί να επιλεγεί το pivot. Γενικά, ο Quick Sort είναι πολύ αποδοτικός σε μεγάλα δεδομένα αλλά σε μικρά απαιτεί τη βοήθεια του Insertion Sort συχνά.

Πλεονεκτήματα και Μειονεκτήματα

Στα πλεονεκτήματα του Quick Sort υπάρχει πολύ μεγάλη ταχύτητα και αποδοτικότητα σε μεγάλους πίνακες που απαιτούν ταξινόμηση αλλά η αποδοτικότητα μειώνεται αν τα δεδομένα είναι ήδη ταξινομημένα ή ίσα. Στα αρνητικά του αλγορίθμου υπάρχει το γεγονός ότι καταλαμβάνει αρκετό χώρο σε μεγάλα δεδομένα με O(logn) παραπάνω μνήμη.

Ψευδοκώδικας

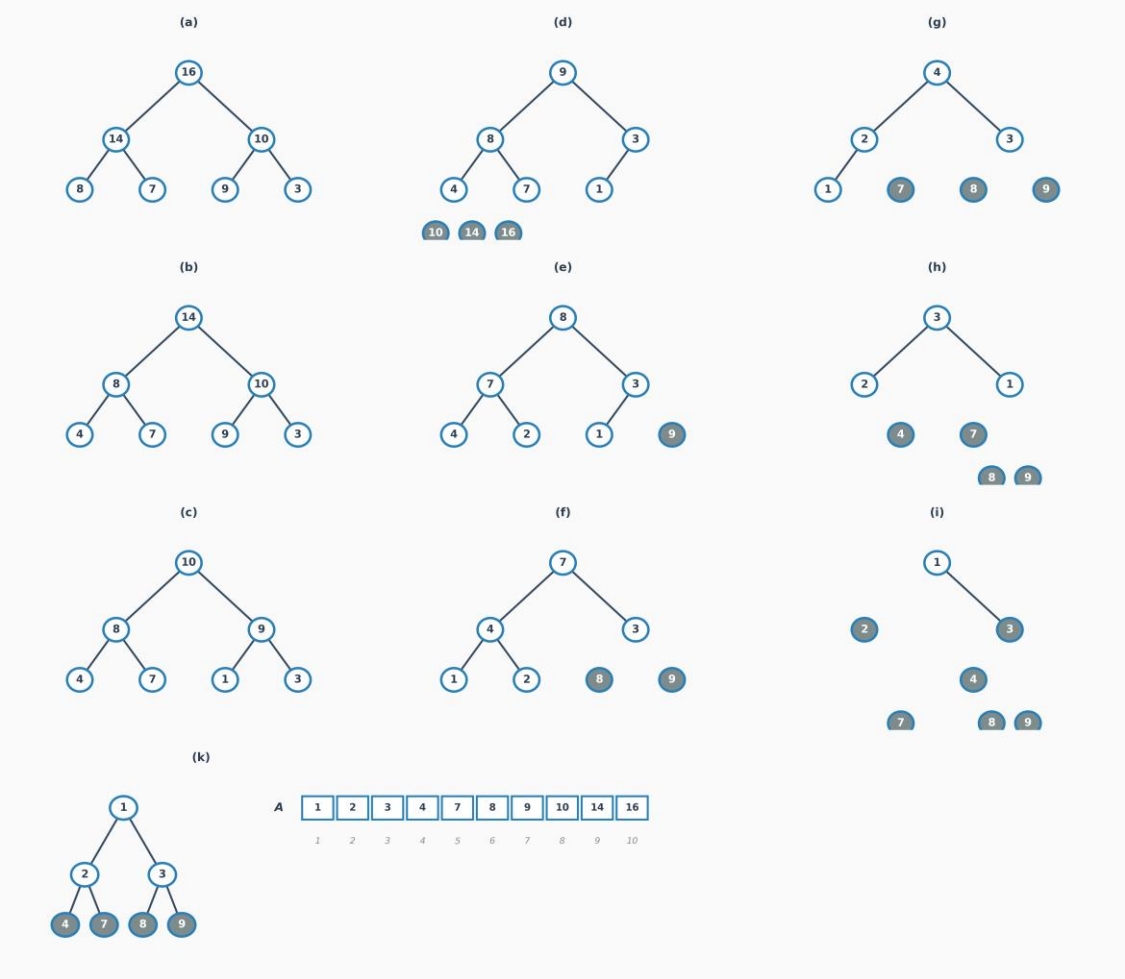
```
1: QUICKSORT(A, p, r)
2:      if p < r then
3:           q ← PARTITION(A, p, r)
4:           QUICKSORT(A, p, q-1)
5:           QUICKSORT(A, q+1, r)
6:
7: PARTITION(A, p, r)
8:      pivot ← A[r]
9:      i ← p-1
10:     for j ← p to r-1
11:          if A[j] < pivot then
12:               i ← i+1
13:               swap(A[i], A[j])
14:               swap(A[i+1], A[r])
15:     return q
```

2.5.Z Heap Sort

Περιγραφή Λειτουργίας

Ο Heap Sort για τη διαδικασία της ταξινόμησης χρησιμοποιεί μια δομή δεδομένων που ονομάζεται σωρός. [2] Πρώτα μετατρέπεται η λίστα σε μία μέγιστου σωρού, όπου το μεγαλύτερο στοιχείο βρίσκεται στη ρίζα και στη συνέχεια εξάγεται επαναληπτικά το μεγαλύτερο στοιχείο από τη ρίζα, τοποθετείται στο τέλος της λίστας και αλλάζει ο σωρός. Η διαδικασία συνεχίζεται μέχρι όλα τα στοιχεία του πίνακα να είναι ταξινομημένα.

Για την κατανόηση της λειτουργίας παρουσιάζεται η εικόνα παρακάτω όπου υπάρχει ένας πίνακας στοιχείων A = [16, 14, 10, 8, 7, 9, 3, 2, 4, 1] και απαιτείται να ταξινομηθεί σε αύξουσα σειρά. Αρχικά, ο αλγόριθμος δημιουργεί μια σωρό από τον πίνακα και αφαιρεί το μέγιστο στοιχείο από το δέντρο και το τοποθετεί στην πιο δεξιά θέση του τελικού ταξινομημένου πίνακα. Κάθε φορά που αφαιρείται το μέγιστο στοιχείο από τη σωρό, αυτή αποκαθίσταται. Παρατηρείται ότι ο πάνω κόμβος έχει τιμή 16 που είναι μεγαλύτερη από τους δύο κάτω που έχουν 14 και 10.



Εικόνα 2.5.Z Heap Sort

Απόδοση

Ο Heap Sort έχει πολυπλοκότητα μέσης και χειρότερης περίπτωσης O(nlogn). Συγκριτικά με άλλους αλγορίθμους ίδιας πολυπλοκότητας ο Heap Sort δεν λειτουργεί τόσο γρήγορα. Ωστόσο, θεωρείται αποδοτικός για ταξινόμηση πινάκων με πολλά δεδομένα, λόγω της αναδρομικότητάς του. Επίσης ο αλγόριθμος απαιτεί μια σταθερή ποσότητα πρόσθετου χώρου μνήμης για να λειτουργήσει. Υπάρχουν επίσης και παραλλαγές του Heap Sort όπως είναι ο Bottom-Up-Heap Sort που είναι πιο γρήγορος από τον Quick Sort.

Πλεονεκτήματα και Μειονεκτήματα

Στα πλεονεκτήματα του Heap Sort υπάρχει το γεγονός ότι είναι γρήγορος σε μεγάλους πίνακες με πολλά δεδομένα λόγω της αναδρομικότητάς του. Από την άλλη όμως δεν είναι ιδιαίτερα γρήγορος σε συνδεδεμένες λίστες γιατί δυσκολεύεται να μετατρέψει συνδεδεμένες λίστες σε δομή σωρού.

Ψευδοκώδικας

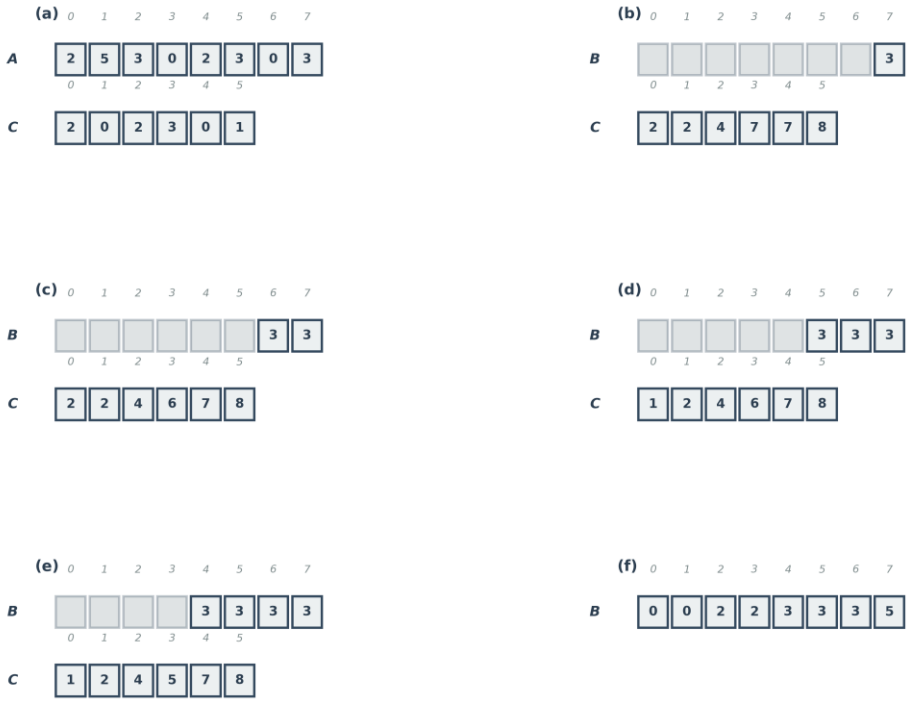
```
1: HEAPSORT(A)
2: BUILD_MAX_HEAP(A)
3: for i ← A.length downto 2 do
4:         exchange A[1] with A[i]
5:         A.heap_size ← A.heap_size - 1
6:         MAX_HEAPIFY(A, 1)
```

2.5.H Counting Sort

Περιγραφή Λειτουργίας

[25] Ο Counting Sort είναι ένας μη συγκριτικός αλγόριθμος που λειτουργεί μετρώντας πόσες φορές εμφανίζεται κάθε διαφορετική τιμή στη λίστα. Δημιουργείται ένας πίνακας καταμέτρησης όπου κάθε θέση αντιστοιχεί σε μια τιμή και αποθηκεύεται πόσες φορές εμφανίζεται αυτή η τιμή. Στη συνέχεια, χρησιμοποιείται αυτός ο πίνακας για να τοποθετηθεί κάθε στοιχείο στη σωστή θέση στην ταξινομημένη λίστα. Λειτουργεί καλά όταν ο αριθμός των τιμών δεν είναι μεγάλος.

Στην παρακάτω εικόνα παρουσιάζεται η λειτουργία του Counting Sort στον πίνακα A = [2, 5, 3, 0, 2, 3, 0, 3]. Ο αλγόριθμος δημιουργεί έναν βοηθητικό πίνακα C που μετρά πόσο συχνά εμφανίζεται κάθε στοιχείο του πίνακα. Στη συνέχεια, ο πίνακας C μετατρέπεται σε αθροιστική μορφή που δείχνει τη θέση κάθε στοιχείου στον τελικό πίνακα. Τέλος, τα στοιχεία τοποθετούνται στον πίνακα B σύμφωνα με τις θέσεις C και προκύπτει ο τελικός ταξινομημένος πίνακας.



Εικόνα 2.5.H Counting Sort

Απόδοση

Όσον αφορά την απόδοση του αλγορίθμου, στη χειρότερη και μέση περίπτωση έχει πολυπλοκότητα O(n+k). Για να επιτευχθεί η μέγιστη απόδοση από τον αλγόριθμο πρέπει το k να είναι μικρότερο από το n. Σε σχέση με άλλους αλγορίθμους ταξινόμησης, ο Counting Sort είναι εύκολα υλοποιήσιμος χωρίς ιδιαίτερες δυσκολίες. Επιπλέον, αποτελεί έναν σταθερό αλγόριθμο που διατηρεί τη σχετική σειρά των στοιχείων με ίσες τιμές, γεγονός που τον καθιστά χρήσιμο ως βοήθεια σε πιο σύνθετους αλγορίθμους όπως ο Radix Sort. Ωστόσο, δεν είναι κατάλληλος για μεγάλα σύνολα δεδομένων με μεγάλο εύρος τιμών ή για την ταξινόμηση συμβολοσειρών.

Πλεονεκτήματα και Μειονεκτήματα

Ένα από τα πλεονεκτήματα του Counting Sort είναι ότι χρησιμοποιεί τις τιμές κλειδιών ως δείκτες σε έναν πίνακα. Για αυτόν τον λόγο, μπορεί να χρησιμοποιηθεί σε άλλους αλγορίθμους ταξινόμησης, επομένως θα πρέπει να διατηρείται η σχετική σειρά των στοιχείων με ίσα κλειδιά. Από την άλλη πλευρά, δεν είναι κατάλληλος για μεγάλα σύνολα δεδομένων και συμβολοσειρές.

Ψευδοκώδικας

```
1: COUNTING SORT(A, B, k)
2:      let C[0...k] be a new array
3:      for i ← 0 to k
4:              C[i] ← 0
5:      for j ← 1 to A.length
6:              C[A[j]] ← C[A[j]] + 1
7:      for i ← 1 to k
8:              C[i] ← C[i] + C[i - 1]
9:      for j ← A.length downto 1
10:             B[C[A[j]]] ← A[j]
11:             C[A[j]] ← C[A[j]] - 1
```

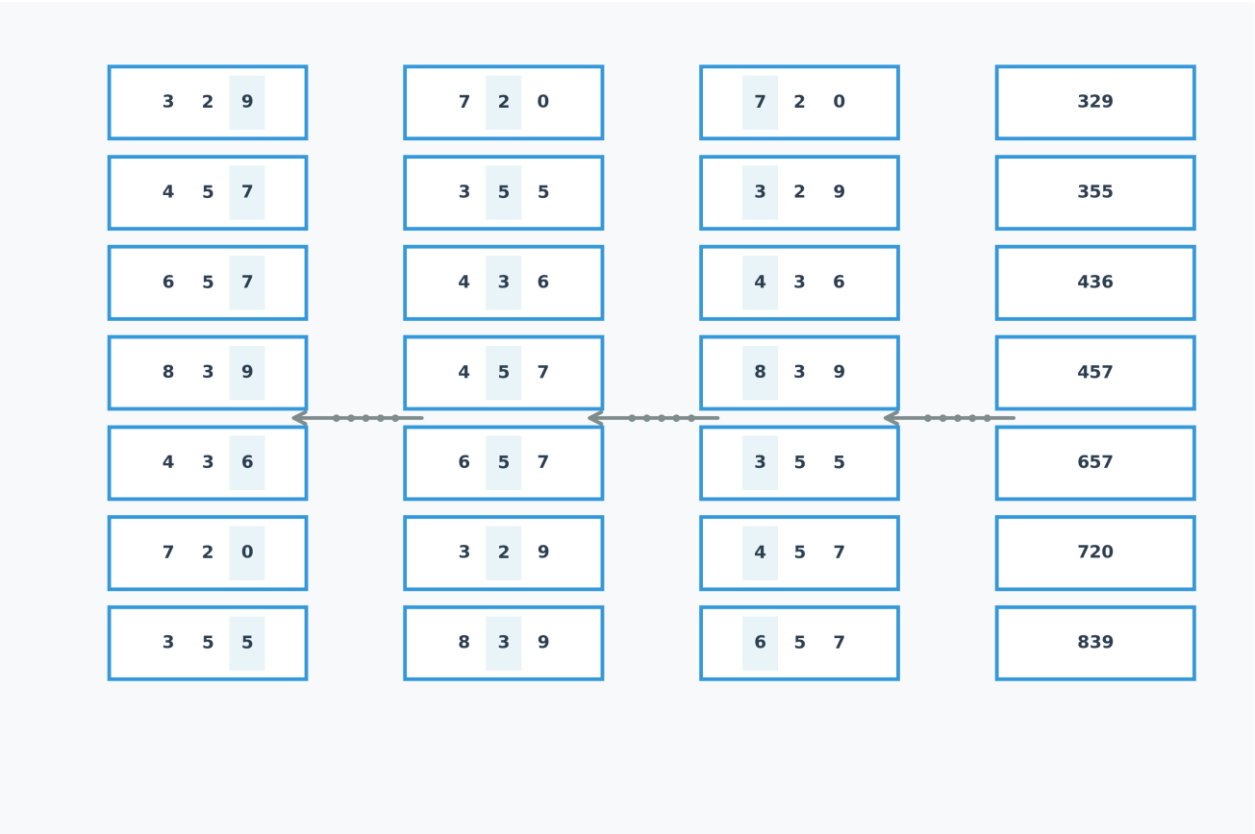
2.5.Θ Radix Sort

Περιγραφή Λειτουργίας

Ο Radix Sort δεν χρησιμοποιεί σύγκριση στοιχείων για την ταξινόμηση του πίνακα, σε αντίθεση με τον Insertion ή τον Quick Sort. Ο Radix Sort λειτουργεί ταξινομώντας τα στοιχεία δεδομένων με βάση τα κλειδιά τους. Ταξινομείται κάθε ψηφίο από τα στοιχεία εισόδου ξεκινώντας από το λιγότερο σημαντικό μέχρι το πιο σημαντικό. Αποτελεί έναν σταθερό αλγόριθμο καθώς διατηρείται η σχετική σειρά των στοιχείων με ίσα κλειδιά. Στον Radix Sort υπάρχουν δύο εκδοχές: ο Least Significant Digit (LSD) και ο Most Significant Digit (MSD). Η μέθοδος LSD λειτουργεί όπως προαναφέρθηκε ενώ η MSD με τον ακριβώς αντίστροφο τρόπο.

Παρουσιάζεται η λειτουργία του Radix Sort στον παρακάτω πίνακα A = [329, 457, 657, 839, 436, 720, 355]. Ο αλγόριθμος ταξινομεί τα στοιχεία με βάση κάθε ψηφίο τους, ξεκινώντας από το λιγότερο σημαντικό ψηφίο (LSD) και προχωρώντας προς το πιο σημαντικό.

Στην πρώτη επανάληψη, πραγματοποιείται ταξινόμηση με βάση το δεξιότερο ψηφίο, δίνοντας: [720, 355, 436, 457, 657, 329, 839]. Στη δεύτερη επανάληψη, πραγματοποιείται ταξινόμηση με βάση το μεσαίο ψηφίο, δίνοντας: [720, 329, 436, 839, 355, 457, 657]. Στην τελευταία επανάληψη, πραγματοποιείται ταξινόμηση με βάση το αριστερότερο ψηφίο, δίνοντας τον ταξινομημένο πίνακα: [329, 355, 436, 457, 657, 720, 839].



Εικόνα 2.5.Θ Radix Sort

Απόδοση

Στην απόδοση του Radix Sort υπάρχει μέση πολυπλοκότητα O(d*n), με d αριθμό ψηφίων. Όσο δεν αλλάζει το d τόσο πιο σταθερός είναι ο αλγόριθμος. Ωστόσο, με διαφορετικό αριθμό ψηφίων, η πολυπλοκότητα γίνεται O(logn), παρόμοια με τον Quick Sort και Merge Sort. [26] Υπάρχουν επίσης και παραλλαγές όπως ο Fast Radix Sort και ο Forward Radix Sort.

Πλεονεκτήματα και Μειονεκτήματα

Στα πλεονεκτήματα του Radix Sort υπάρχει η απόδοσή του η οποία δεν επηρεάζεται από τον τύπο και το μέγεθος των δεδομένων εισόδου που ταξινομούνται. Από την άλλη στα μειονεκτήματα μπορεί να αναφερθεί ότι είναι αρκετά δύσκολος στην υλοποίησή του και δεν είναι τόσο ευέλικτος. Στην κατανάλωση χώρου έχει αρκετά μεγάλες απαιτήσεις οπότε σε μεγάλους πίνακες απαιτείται ιδιαίτερη προσοχή.

Ψευδοκώδικας

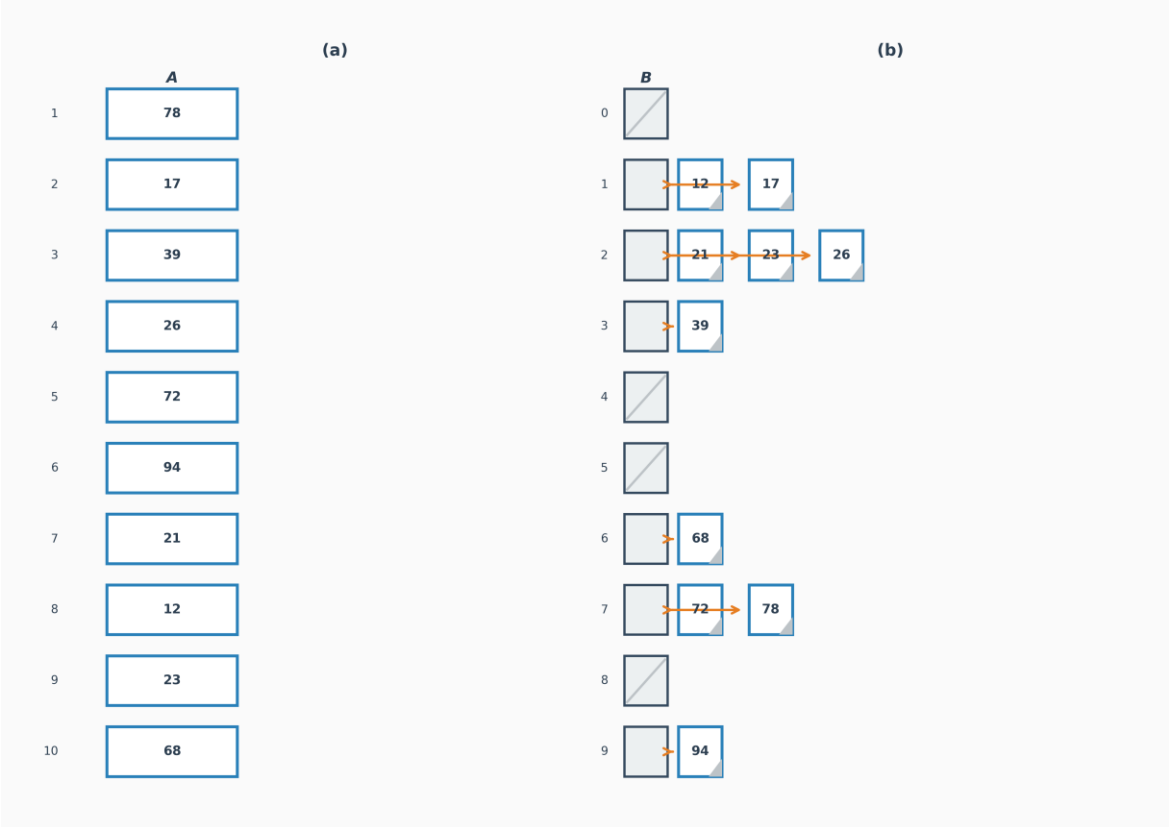
```
RADIX_SORT(list)
1:      for i ← 1 to d do
2:              Use a stable sort to sort array A on digit i
```

2.5.I Bucket Sort

Περιγραφή Λειτουργίας

Ο Bucket Sort λειτουργεί χωρίς τη χρήση συγκρίσεων, όπως συμβαίνει και στον Radix Sort. Η λειτουργία του βασίζεται στη διανομή των στοιχείων του πίνακα σε buckets με βάση το εύρος των τιμών τους, έτσι ώστε κάθε bucket να καλύπτει ένα συγκεκριμένο εύρος τιμών. Στη συνέχεια, κάθε bucket ταξινομείται ξεχωριστά, συνήθως με τη χρήση κάποιου βοηθητικού αλγορίθμου ταξινόμησης και τα ταξινομημένα buckets συγχωνεύονται για την παραγωγή του τελικού ταξινομημένου πίνακα. Ο Bucket Sort αποδίδει καλύτερα όταν τα δεδομένα είναι ομοιόμορφα κατανεμημένα, ενώ η σταθερότητά του εξαρτάται από τον βοηθητικό αλγόριθμο που χρησιμοποιείται.

Όπως φαίνεται και στην εικόνα 2.5.I, ο Bucket Sort λειτουργεί σε δύο φάσεις. Στην πρώτη φάση, τα στοιχεία του πίνακα κατανέμονται σε buckets με βάση την τιμή τους, όπως παρουσιάζεται στον πίνακα A, όπου τα δεδομένα διανέμονται σε 10 buckets από B[0] έως B[9]. Για παράδειγμα, οι τιμές 78 και 72 τοποθετούνται στο B[7], οι τιμές 17 και 12 στο B[1], ενώ ορισμένα buckets, όπως τα B[0] και B[4], παραμένουν κενά. Στη δεύτερη φάση πραγματοποιείται η εσωτερική ταξινόμηση κάθε bucket με τη χρήση βοηθητικού αλγορίθμου, συνήθως Insertion Sort, με αποτέλεσμα την παραγωγή του τελικού ταξινομημένου πίνακα.



Εικόνα 2.5.I Bucket Sort

Απόδοση

Η απόδοση του Bucket Sort έχει πολυπλοκότητα $O(n^2)$ στη χειρότερη περίπτωση και $O(n + k)$ στη μέση περίπτωση. Εξαιτίας του βοηθητικού αλγορίθμου ο χρόνος εκτέλεσης αυξάνεται, ενώ στη χειρότερη περίπτωση απαιτείται χώρος $O(n * k)$ για την αποθήκευση των πινάκων.

Πλεονεκτήματα και Μειονεκτήματα

Στα πλεονεκτήματα του Bucket Sort περιλαμβάνεται ο γραμμικός χρόνος εκτέλεσης στη μέση περίπτωση, με πολυπλοκότητα $O(n + k)$, κάτι που τον κάνει ιδιαίτερα γρήγορο όταν τα δεδομένα είναι ομοιόμορφα κατανεμημένα. Ωστόσο, δεν είναι αποδοτικός σε μεγάλους πίνακες, καθώς στη χειρότερη περίπτωση η πολυπλοκότητα φτάνει το $O(n^2)$ λόγω του βοηθητικού αλγορίθμου που χρησιμοποιείται για την ταξινόμηση των buckets. Επιπλέον, απαιτεί σημαντική ποσότητα μνήμης για την αποθήκευση των buckets.

Ψευδοκώδικας

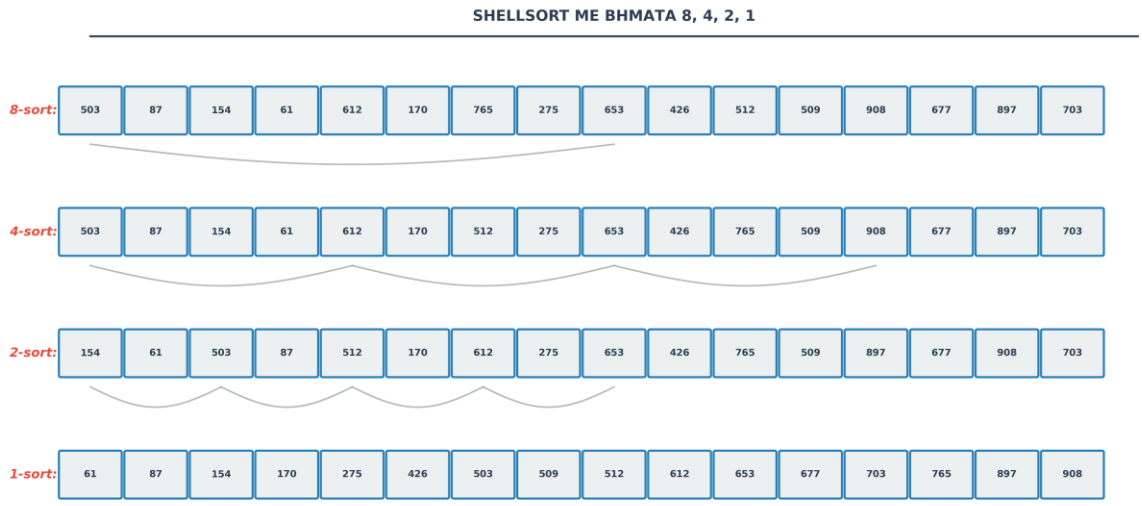
```
1: BUCKET-SORT(A)
2:      n ← A.length
3:      let B[0...n-1] be a new array of empty lists
4:      for i ← 0 to n-1
5:          insert A[i] into list B[⌊n · A[i]⌋]
6:      for i ← 0 to n-1
7:          sort list B[i] with INSERTION-SORT
8:      concatenate the lists B[0], B[1], ..., B[n-1] together in order
9:      return the concatenated list
```

2.5.K Shell Sort

Περιγραφή Λειτουργίας

[23] Ο Shell Sort μπορεί να θεωρηθεί βελτιωμένη εκδοχή του Insertion Sort. [3] Αντί για σύγκριση και εναλλαγή γειτονικών στοιχείων, συγκρίνονται στοιχεία που βρίσκονται σε συγκεκριμένη απόσταση μεταξύ τους, ενώ αυτή η απόσταση μειώνεται σταδιακά όσο προχωρά η διαδικασία ταξινόμησης, μέχρι να φτάσει στη μονάδα. Με αυτόν τον τρόπο πραγματοποιείται μετακίνηση απομακρυσμένων στοιχείων ώστε ο πίνακας να γίνει αρκετά ταξινομημένος και στη συνέχεια με τη βοήθεια του Insertion Sort να ολοκληρωθεί η ταξινόμηση πιο γρήγορα.

Στην εικόνα 2.5.K παρουσιάζεται η λειτουργία του Shell Sort με την ακολουθία βημάτων 8, 4, 2, 1. Στο βήμα 8 συγκρίνονται στοιχεία που απέχουν 8 θέσεις, στο βήμα 4 στοιχεία που απέχουν 4 θέσεις, και η διαδικασία συνεχίζεται έως το βήμα 1, όπου εφαρμόζεται ο Insertion Sort στον σχεδόν ταξινομημένο πίνακα, παράγοντας τον τελικό ταξινομημένο πίνακα.



Εικόνα 2.5.K Shell Sort

Απόδοση

Η απόδοση του Shell Sort εξαρτάται από τα βήματα που θα κάνει ώστε να ταξινομήσει τον πίνακα. Στην χειρότερη περίπτωση έχει πολυπλοκότητα από $O(n^2)$ έως $O(n\log^2 n)$, ενώ στην μέση περίπτωση έχει πολυπλοκότητα O(nlogn) ή O(n^(3/2)) ανάλογα την περίπτωση. Ο Shell Sort ταξινομεί επί τόπου και καταλαμβάνει χώρο ίσο με O(1).

Πλεονεκτήματα και Μειονεκτήματα

Στα πλεονεκτήματα του Shell Sort περιλαμβάνονται η ταχύτητα, η μικρή κατανάλωση μνήμης, η απλότητα στην υλοποίηση και η αποδοτικότητα. Στα μειονεκτήματά του υπάρχει η εξάρτηση της απόδοσης του από την επιλογή των βημάτων, γεγονός που σημαίνει ότι ο αλγόριθμος δεν είναι σταθερός, καθώς μπορεί να αλλάξει τη σειρά ίσων στοιχείων.

Ψευδοκώδικας

```
1: SHELL-SORT(A)
2:      n ← A.length
3:      gap ← n / 2
4:      while gap > 0
5:              for i ← gap to n-1
6:                      temp ← A[i]
7:                      j ← i
8:                      while j ≥ gap and A[j - gap] > temp
9:                              A[j] ← A[j - gap]
10:                             j ← j - gap
11:                     A[j] ← temp
12:             gap ← gap / 2
```

ΚΕΦΑΛΑΙΟ 3 ΣΧΕΔΙΑΣΜΟΣ ΚΑΙ ΥΛΟΠΟΙΗΣΗ

3.1 Εισαγωγή

[27], [30] Στην παρούσα εργασία παρουσιάζεται μία εφαρμογή οπτικοποίησης δέκα διαφορετικών αλγορίθμων ταξινόμησης για πίνακα ακεραίων, γραμμένη σε Python version 3.14.0. Η συγκεκριμένη έκδοση της Python έκανε την εμφάνισή της τον Οκτώβριο του 2025 και προσφέρει βελτιωμένη απόδοση, πολύ μεγάλη ασφάλεια και υποστήριξη μεγάλου όγκου βιβλιοθηκών. Η εργασία δημιουργήθηκε με σκοπό την κατανόηση σε βάθος των λειτουργιών των αλγορίθμων μέσω γραφικής αναπαράστασης και μέτρησης χρόνου για τον κάθε αλγόριθμο ξεχωριστά. [31] Το παρόν κεφάλαιο αναφέρεται στην αρχιτεκτονική της εφαρμογής, δηλαδή τις τεχνολογίες που χρησιμοποιήθηκαν και τις επιλογές στον σχεδιασμό, αλλά και τον τρόπο υλοποίησης κάθε κομματιού από τον κώδικα. [29] Η εφαρμογή προσφέρει ένα πλήρες περιβάλλον για την δοκιμή, σύγκριση και οπτική αναπαράσταση των δέκα αλγορίθμων ταξινόμησης. Επιλέχθηκε η Python ως γλώσσα υλοποίησης της εφαρμογής λόγω των πολλών πλεονεκτημάτων που διαθέτει και την καθιστά ίσως ιδανική για την παρούσα εργασία.

Πλεονεκτήματα της Γλώσσας

[12] Αρχικά, η Python αποτελεί μία πολύ απλή γλώσσα προγραμματισμού και η σύνταξή της είναι εύκολα κατανοητή, γεγονός που επιτρέπει την υλοποίηση πολλών αλγορίθμων με σαφή τρόπο, επιτρέποντας εύκολη προσθήκη άλλου αλγορίθμου στον κώδικα αλλά και συντήρηση των υπαρχόντων. Η σχεδίαση της εφαρμογής έχει φιλοσοφία και δομή που είναι προσβάσιμη και κατανοήσιμη ακόμη και από προγραμματιστές με ελάχιστη εμπειρία, αφού το να είναι ευανάγνωστος ένας κώδικας θεωρείται πολύ σημαντικό κριτήριο στον προγραμματισμό. Επίσης, η γλώσσα Python προσφέρει πλούσια συλλογή βιβλιοθηκών που μπορούν να καλύψουν σχεδόν κάθε ανάγκη που μπορεί να υπάρξει. Συγκεκριμένα, στην εργασία υπάρχει η δημιουργία γραφικής διεπαφής με την χρήση του Tkinter που είναι ήδη ενσωματωμένο στην Python και η οπτικοποίηση των δεδομένων με την βοήθεια του Matplotlib. Η ενσωμάτωση αυτών των βιβλιοθηκών στον κώδικα καθιστά την διαδικασία πολύ πιο εύκολη και απλή.

Ακόμη, η Python διαθέτει δυναμική τυποποίηση, η οποία επιτρέπει στον προγραμματιστή να ασχοληθεί περισσότερο με το κομμάτι των αλγορίθμων, χωρίς να αφιερώνεται ιδιαίτερος χρόνος στην δήλωση των τύπων δεδομένων. Επιπλέον, η αυτόματη διαχείριση μνήμης που διαθέτει η γλώσσα απαλλάσσει τον προγραμματιστή από χειροκίνητη δέσμευση και αποδέσμευση της μνήμης, μειώνοντας τις πιθανότητες σφάλματος. Ένα ακόμη θετικό της γλώσσας είναι ότι αποτελεί διερμηνευόμενη γλώσσα, που σημαίνει ότι επιτρέπει την γρήγορη εκτέλεση του κώδικα χωρίς την ανάγκη μεταγλωττιστή. Η Python γενικά έχει μεγάλο documentation και τεράστια κοινότητα προγραμματιστών παγκοσμίως. Αυτό διευκολύνει σημαντικά την αντιμετώπιση προβλημάτων και την πρόσβαση σε μεγάλη γκάμα υλικού και γνώσης. Είναι cross-platform και υποστηρίζεται σε όλα τα κύρια λειτουργικά συστήματα, εξασφαλίζοντας την φορητότητα της εφαρμογής. [39], [40] Ανεξαρτήτως του γεγονότος ότι η Python θεωρείται πιο αργή σε σχέση με γλώσσες προγραμματισμού που χρησιμοποιούν μεταγλώττιση όπως η C++ ή η Java, η απόδοσή της στην εργασία και ο τρόπος που χρησιμοποιήθηκε ήταν επαρκής.

3.2 Αρχιτεκτονική Συστήματος

[11] Για την υλοποίηση και τη δοκιμή της εφαρμογής χρησιμοποιήθηκε ένα σύστημα με επεξεργαστή AMD Ryzen 5 5600X, ο οποίος προσφέρει υψηλή απόδοση και σταθερότητα για εφαρμογές που απαιτούν συνεχείς υπολογισμούς, όπως οι αλγόριθμοι ταξινόμησης. Η κάρτα γραφικών ήταν η MSI RTX 3060 και βοήθησε στην ομαλή οπτικοποίηση των βημάτων ταξινόμησης, καθώς και στην καλή απόδοση του Matplotlib, ειδικά σε περιπτώσεις μεγάλων πινάκων με πολλά ενδιάμεσα βήματα. Η μνήμη RAM G.Skill Ripjaws 16GB (2×8GB) ήταν επαρκής για την εκτέλεση όλων των λειτουργιών της εφαρμογής, εξασφαλίζοντας ότι δεν υπήρξε επιβάρυνση στις διαδικασίες καταγραφής βημάτων, χρονομέτρησης και οπτικοποίησης. Ο συνδυασμός αυτών των χαρακτηριστικών παρείχε ένα σταθερό και αξιόπιστο περιβάλλον για την ανάπτυξη και τη σωστή αξιολόγηση όλων των αλγορίθμων που παρουσιάζονται στην εργασία.

3.2.Α Επίπεδο Αλγορίθμων

Στο επίπεδο αλγορίθμων είναι δυνατή η επισκόπηση της υλοποίησης των δέκα αλγορίθμων ταξινόμησης. Όλοι οι αλγόριθμοι είναι αποθηκευμένοι σε ένα dictionary με το όνομα “ALGORITHMS”, όπου κάθε κλειδί αντιστοιχεί στο όνομα του αλγορίθμου και η τιμή αποτελεί την αντίστοιχη συνάρτηση. Αυτή η δομή επιτρέπει την εύκολη επανάληψη όλων των αλγορίθμων και απλοποιεί την προσθήκη νέων αλγορίθμων στον κώδικα. Κάθε αλγόριθμος είναι γραμμένος ως ξεχωριστή συνάρτηση ώστε να είναι εύκολα διαχωρισμένος από τους υπόλοιπους και η δομή του είναι αυθεντική, όπως παρουσιάζεται στα βιβλία. Κάθε αλγόριθμος δέχεται δύο ορίσματα: το πρώτο όρισμα είναι ο πίνακας που εισάγεται από τον χρήστη και πρόκειται να ταξινομηθεί, και το δεύτερο είναι μία λίστα για την αποθήκευση των ενδιάμεσων καταστάσεων, δηλαδή των βημάτων, τα οποία χρησιμοποιούνται στην γραφική αναπαράσταση.

Η φιλοσοφία στην υλοποίηση κάθε αλγορίθμου έχει την ίδια δομή και είναι οπτικά όπως παρακάτω:
Όνομα Αλγορίθμου Ταξινόμησης (Πίνακας Χρήστη, Βήματα):

Καταγραφή (Πίνακας Χρήστη, Βήματα)

Πραγματοποίηση της ταξινόμησης με καταγραφή βημάτων

Επιστροφή του ταξινομημένου πίνακα

3.2.Β Επίπεδο Παρουσίασης

Για το επίπεδο παρουσίασης περιλαμβάνεται στον κώδικα η κλάση GUI, σκοπός της οποίας είναι η αλληλεπίδραση με τον χρήστη, ώστε να παρέχεται ένα όμορφο και κατανοητό περιβάλλον για την εισαγωγή των στοιχείων του πίνακα προς ταξινόμηση. Συγκεκριμένα, εισάγεται από τον χρήστη το μέγεθος του πίνακα, καθώς και η ελάχιστη και μέγιστη τιμή που πρόκειται να έχουν τα στοιχεία του πίνακα. Επίσης, παρέχεται επιλογή με checkbox εάν επιθυμείται ο πίνακας να περιλαμβάνει διπλότυπους αριθμούς, και ακολουθεί η επιλογή του τύπου ταξινόμησης του πίνακα που εισάγεται, δηλαδή εάν επιθυμείται ο πίνακας να είναι σε αύξουσα, τυχαία ή φθίνουσα σειρά. Τέλος, παρέχονται δύο buttons με τα οποία είναι δυνατή η εκτέλεση του κώδικα και η εμφάνιση του γραφήματος με την αναπαράσταση των αλγορίθμων. Η συγκεκριμένη κλάση εκτελεί ένα προς ένα τους αλγορίθμους διαδοχικά, καταγράφοντας τον χρόνο εκτέλεσης του καθενός.

3.2.Γ Επίπεδο Οπτικοποίησης

Στο επίπεδο οπτικοποίησης, περιλαμβάνεται η κλάση Graph, μέσω της οποίας παρέχεται η δυνατότητα εξαγωγής της γραφικής αναπαράστασης κάθε αλγορίθμου ταξινόμησης, σύμφωνα με τα βήματα που πραγματοποιήθηκαν από τον καθένα. Χρησιμοποιείται γράφημα για την παρακολούθηση της πορείας του αλγορίθμου κάθε χρονική στιγμή, από την στιγμή που λαμβάνεται ο αρχικός πίνακας από τον χρήστη μέχρι την στιγμή που ο πίνακας είναι πλήρως ταξινομημένος. Ουσιαστικά, η γραφική αναπαράσταση κάθε αλγορίθμου διατηρεί ίδια αρχή και ίδιο τέλος, αλλά διαφέρουν τα ενδιάμεσα βήματα, καθώς κάθε αλγόριθμος λειτουργεί διαφορετικά.

3.3 Τεχνολογίες και Βιβλιοθήκες

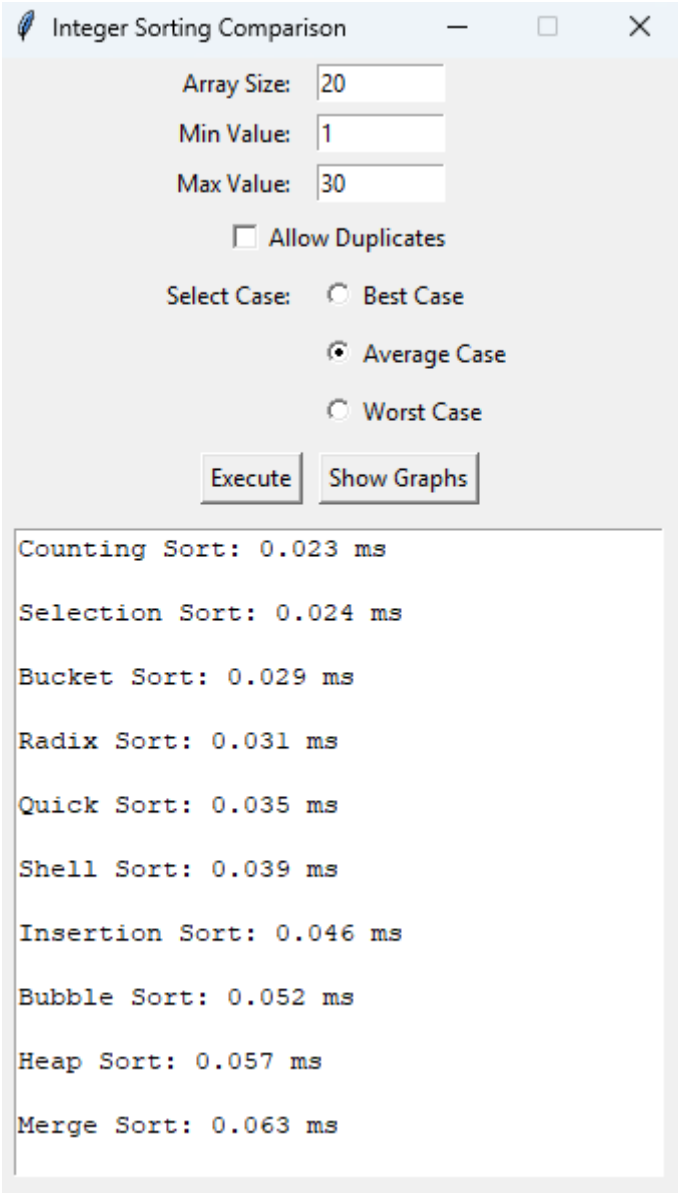
3.3.Α Βιβλιοθήκη για τη Γραφική Διεπαφή

[28] Για το επίπεδο παρουσίασης χρησιμοποιείται η βιβλιοθήκη Tkinter, η οποία είναι ενσωματωμένη στη γλώσσα Python. [41] Συγκεκριμένα, το Tkinter αποτελεί την προεπιλεγμένη βιβλιοθήκη της γλώσσας για την γραφική διεπαφή, δηλαδή το GUI. Το όνομά της προέρχεται από το "Tk interface" καθώς αποτελεί διεπαφή της εργαλειοθήκης Tk, η οποία αρχικά χρησιμοποιούνταν από τη γλώσσα Tcl. Η βιβλιοθήκη Tkinter δεν απαιτεί εξωτερική εγκατάσταση για τη χρήση της, γεγονός που την καθιστά πολύ βολική για την ανάπτυξη εφαρμογών που απαιτούν φορητότητα.

Η βιβλιοθήκη Tkinter διαθέτει μια μεγάλη γκάμα από γραφικά στοιχεία που επιτρέπουν τη δημιουργία ενός πλήρως λειτουργικού περιβάλλοντος με διεπαφές. Τα βασικότερα γραφικά στοιχεία είναι το Label για εμφάνιση κειμένου, το Entry για εισαγωγή δεδομένων από τον χρήστη, το Button για εκτέλεση συγκεκριμένων ενεργειών, το Checkbutton για επιλογές τύπου boolean, το Radiobutton για αποκλειστικές επιλογές, το Text για εμφάνιση και επεξεργασία κειμένου, το Combobox για λίστες επιλογών και το Frame για οργάνωση των γραφικών στοιχείων. Η βιβλιοθήκη υποστηρίζει επίσης διαχειριστές διάταξης όπως το pack, grid και place, οι οποίοι επιτρέπουν την τοποθέτηση των γραφικών στοιχείων στο επιθυμητό σημείο του παραθύρου της εφαρμογής.

Στην εφαρμογή, υπάρχει ένα κύριο παράθυρο που θα εμφανιστεί στον χρήστη κατά την εκτέλεση του προγράμματος, το οποίο έχει διαστάσεις 345x570 και είναι πάντα κεντραρισμένο στην οθόνη του υπολογιστή με τη μέθοδο center_window(), η οποία υπολογίζει τις συντεταγμένες X και Y με βάση το μέγεθος της οθόνης και του παραθύρου. Το όνομα του παραθύρου είναι Integer Sorting Comparison και ακολουθούν τα πεδία εισόδου που αναφέρθηκαν παραπάνω. Κάτω από τα πεδία εισόδου υπάρχει ένα text πεδίο 40x20 για εμφάνιση του χρόνου εκτέλεσης κάθε αλγορίθμου.

Στην Εικόνα 3.3.Α είναι δυνατή η ακριβής επισκόπηση του παραθύρου στο οποίο γίνεται αναφορά.



Εικόνα 3.3.Α: Το κύριο παράθυρο της εφαρμογής με παραμέτρους και αποτελέσματα

3.3.Β Βιβλιοθήκη για την Οπτικοποίηση

[38] Για το επίπεδο παρουσίασης χρησιμοποιείται η βιβλιοθήκη Matplotlib, η οποία χρησιμοποιείται στην οπτικοποίηση δεδομένων. Η βιβλιοθήκη είναι ανοιχτού κώδικα και η συντήρησή της γίνεται από πολύ μεγάλη κοινότητα προγραμματιστών. [42], [43] Το Matplotlib αποτελεί τη βάση για πολλές άλλες βιβλιοθήκες οπτικοποίησης της Python, όπως το Seaborn και το Pandas plotting, γεγονός που υποδεικνύει τη δύναμη που διαθέτει ως βιβλιοθήκη.

Η αρχιτεκτονική της βιβλιοθήκης χαρακτηρίζεται ιεραρχική. Συγκεκριμένα, στο πάνω μέρος βρίσκεται το Figure, το οποίο είναι ουσιαστικά το παράθυρο του διαγράμματος. Κάθε Figure μπορεί να περιλαμβάνει ένα ή περισσότερα Axes, τα οποία αποτελούν τις περιοχές σχεδίασης που απεικονίζονται τα δεδομένα. Κάθε Axe περιλαμβάνει τους άξονες, τους τίτλους, τις ετικέτες και τα γραφικά στοιχεία. Το pyplot module της βιβλιοθήκης προσφέρει μία απλοποιημένη διεπαφή τύπου MATLAB που επιτρέπει την γρήγορη δημιουργία διαγραμμάτων, παρέχοντας πλήρη έλεγχο.

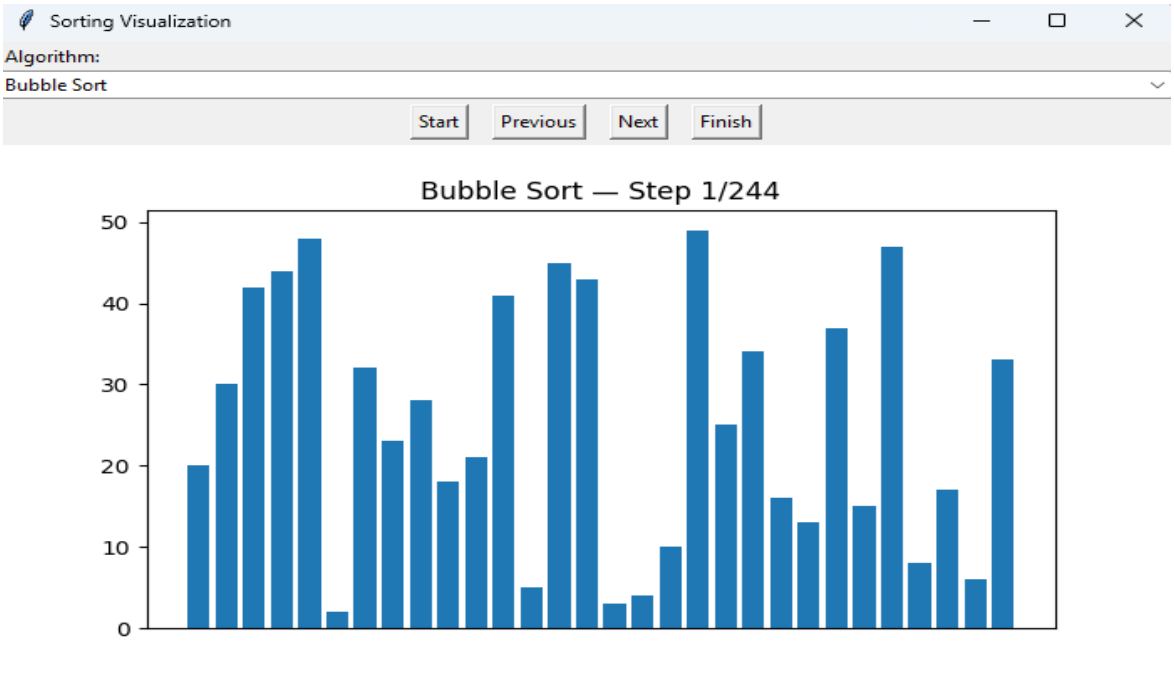
Η βιβλιοθήκη υποστηρίζει την αναπαράσταση πολλών τύπων διαγραμμάτων, όπως γραμμικά διαγράμματα, ραβδογράμματα, ιστογράμματα, διαγράμματα διασποράς, διαγράμματα πίτας, τρισδιάστατα διαγράμματα και άλλα. Επίσης, επιτρέπει τον πλήρη έλεγχο κάθε οπτικού στοιχείου ώστε να εμφανίζεται με τον επιθυμητό τρόπο. Τα διαγράμματα αυτά είναι δυνατόν να εξαχθούν στην επιθυμητή μορφή, όπως PNG, PDF, SVG και EPS, καθιστώντας τα κατάλληλα για κάθε σκοπό που μπορεί να χρησιμοποιηθεί το διάγραμμα.

Επιπλέον, με το module FigureCanvasTkAgg, τα διαγράμματα του Matplotlib μπορούν να ενσωματωθούν απευθείας σε παράθυρα Tkinter, επιτρέποντας τη δημιουργία διαδραστικών εφαρμογών που συνδυάζουν γραφικά στοιχεία εισαγωγής δεδομένων με οπτικοποιήσεις. Αυτή η δυνατότητα είναι σημαντική για την παρούσα εφαρμογή, καθώς επιτρέπει την εμφάνιση των βημάτων ταξινόμησης σε πραγματικό χρόνο εντός του περιβάλλοντος της εφαρμογής.

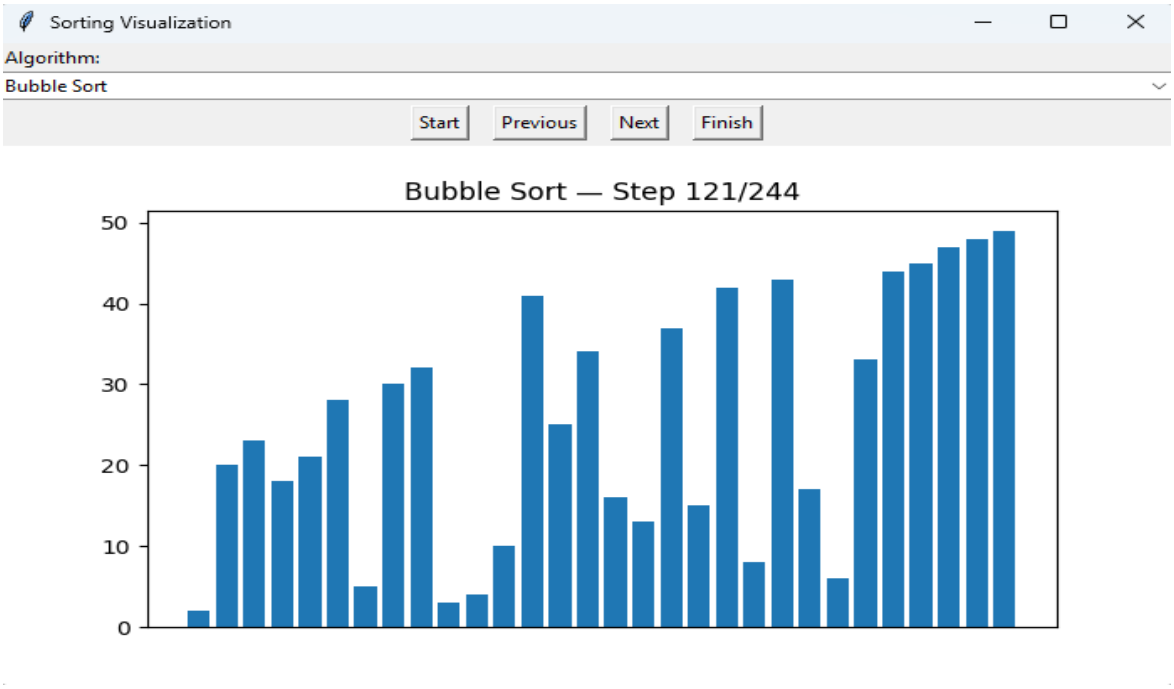
Στην εφαρμογή, το Matplotlib χρησιμοποιείται για τη δημιουργία ραβδογραμμάτων που δείχνουν την κατάσταση του πίνακα σε κάθε βήμα ταξινόμησης. Κάθε ράβδος του διαγράμματος αντιστοιχεί σε ένα στοιχείο του πίνακα, με το ύψος της ράβδου να αντιπροσωπεύει την τιμή του στοιχείου. Ο κάθετος άξονας εμφανίζει τις τιμές των στοιχείων, ενώ ο οριζόντιος άξονας εμφανίζει τις θέσεις τους στον πίνακα. Το παράθυρο οπτικοποίησης δημιουργείται με τίτλο "Sorting Visualization" και περιλαμβάνει ένα Combobox για την επιλογή του αλγορίθμου που θα εμφανιστεί.

Ο τίτλος κάθε διαγράμματος εμφανίζει το όνομα του αλγορίθμου και το τρέχον βήμα με τη μορφή "Algorithm Name - Current Step/Total Steps". Για τη διαχείριση μνήμης, η μέθοδος plt.close(fig) καλείται μετά από κάθε ενημέρωση του διαγράμματος, αποτρέποντας διαρροές μνήμης από τη συσσώρευση αντικειμένων Figure. Τα κουμπιά πλοήγησης (Start, Previous, Next, Last) επιτρέπουν στον χρήστη να μετακινηθεί ανάμεσα στα βήματα της ταξινόμησης, με τη δυναμική προσαρμογή του βήματος μετακίνησης μέσω της μεθόδου get_step_jump() για ταχύτερη πλοήγηση σε πίνακες με πολλά βήματα.

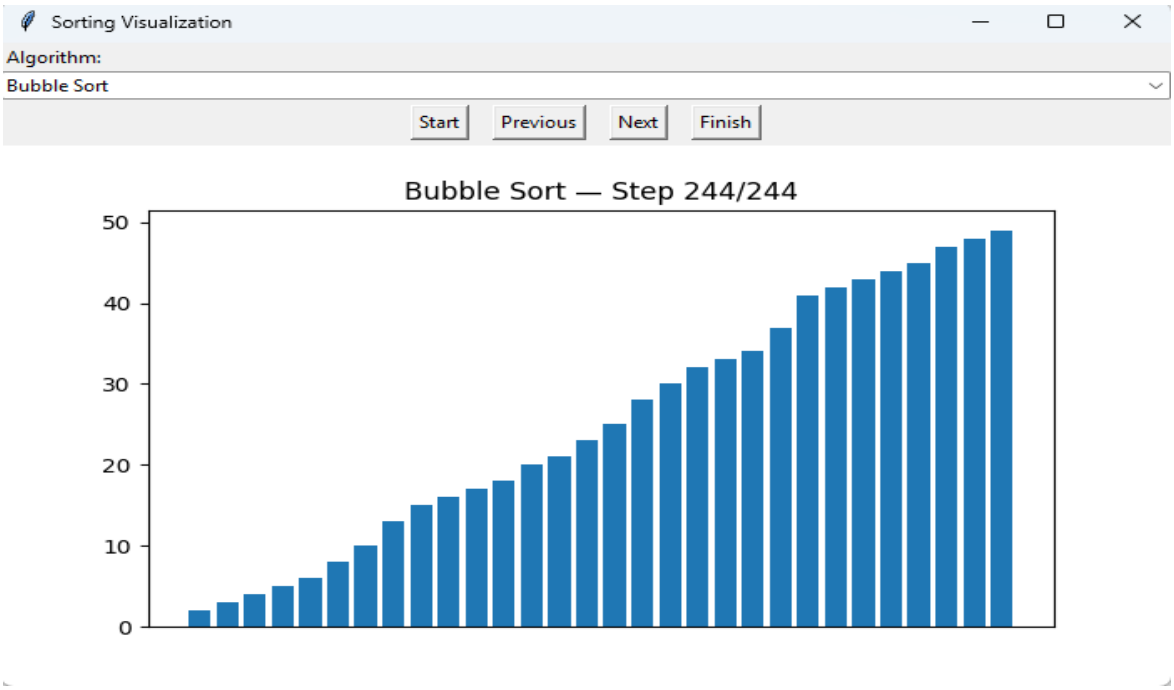
Το παράθυρο στο οποίο γίνεται αναφορά είναι ορατό στην Εικόνα 3.3.Β με τον Bubble Sort.



Εικόνα 3.3.Β(1): Αρχική φάση Bubble Sort (Step 1/244)



Εικόνα 3.3.Β(2): Ενδιάμεση φάση Bubble Sort (Step 121/244)



Εικόνα 3.3.Β(3): Τελική φάση Bubble Sort (Step 244/244)

3.3.Γ Πρόσθετες Βιβλιοθήκες

Εκτός από τις βιβλιοθήκες Tkinter και Matplotlib, οι οποίες αποτελούν τη βάση της γραφικής διεπαφής και της οπτικοποίησης αντίστοιχα, η εφαρμογή χρησιμοποιεί δύο επιπλέον βιβλιοθήκες της Python που είναι απαραίτητες για τη σωστή λειτουργία της: τη βιβλιοθήκη time για τη χρονομέτρηση της εκτέλεσης των αλγορίθμων και τη βιβλιοθήκη random για τη δημιουργία τυχαίων δεδομένων εισόδου. Και οι δύο βιβλιοθήκες ανήκουν στην πρότυπη βιβλιοθήκη της Python, γεγονός που σημαίνει ότι είναι ενσωματωμένες στη γλώσσα και δεν απαιτούν εξωτερική εγκατάσταση.

Βιβλιοθήκη Time

Η βιβλιοθήκη time παρέχει διάφορες συναρτήσεις για την εργασία με χρόνο και χρονομέτρηση. Αποτελεί μία από τις βασικές βιβλιοθήκες της Python και χρησιμοποιείται συχνά σε εφαρμογές που απαιτούν μέτρηση χρόνου εκτέλεσης, καθυστερήσεις ή διαχείριση χρονικών διαστημάτων. Οι πιο βασικές συναρτήσεις της είναι η time.time() η οποία επιστρέφει τον τρέχοντα χρόνο σε δευτερόλεπτα, η time.perf_counter() η οποία παρέχει υψηλή ακρίβεια για τη μέτρηση του χρόνου, και η time.sleep(), η οποία αναστέλλει την εκτέλεση του προγράμματος για συγκεκριμένο χρονικό διάστημα.

Στην εφαρμογή, χρησιμοποιείται η συνάρτηση time.perf_counter() για την καταγραφή του χρόνου εκτέλεσης κάθε αλγορίθμου με υψηλή ακρίβεια. Η μέτρηση ξεκινά με την εντολή start = time.perf_counter() πριν την εκτέλεση του αλγορίθμου και ολοκληρώνεται με την εντολή elapsed = (time.perf_counter() - start) * 1000, όπου το αποτέλεσμα μετατρέπεται σε χιλιοστά του δευτερολέπτου για πιο κατανοητή εμφάνιση. Τα αποτελέσματα αποθηκεύονται στη λίστα self.results[name] και εμφανίζονται ταξινομημένα με βάση τον χρόνο εκτέλεσης.

Βιβλιοθήκη Random

Η βιβλιοθήκη random παρέχει συναρτήσεις για τη δημιουργία τυχαίων αριθμών και την τυχαία επιλογή ή ανακατάταξη στοιχείων. [44] Χρησιμοποιείται ο αλγόριθμος Mersenne Twister για τον σκοπό αυτό, ο οποίος θεωρείται ένας από τους πιο αξιόπιστους αλγορίθμους για εφαρμογές. Οι κυριότερες συναρτήσεις της περιλαμβάνουν την random.randint(a, b), η οποία επιστρέφει έναν τυχαίο ακέραιο μεταξύ a και b, την random.shuffle(seq), η οποία ανακατατάσσει τα στοιχεία μιας λίστας επιτόπου, και την random.sample(population, k), η οποία επιστρέφει k μοναδικά στοιχεία από μία ακολουθία.

Στην εφαρμογή, η βιβλιοθήκη random χρησιμοποιείται στη συνάρτηση create_array() για τη δημιουργία των δεδομένων εισόδου. Όταν ο χρήστης επιτρέπει διπλότυπους αριθμούς, χρησιμοποιείται η random.randint(min_val, max_val) για κάθε στοιχείο. Όταν δεν επιτρέπονται διπλότυπα, χρησιμοποιείται η random.sample(range(min_val, max_val + 1), size) για τη δημιουργία μοναδικών τιμών. Επιπλέον, η random.shuffle(arr) χρησιμοποιείται για την τυχαία ανακατάταξη του πίνακα στο Average Case, επιτρέποντας την εξέταση της συμπεριφοράς των αλγορίθμων σε διαφορετικές αρχικές καταστάσεις.

3.4 Λειτουργίες και Υλοποίηση

3.4.Α Σύστημα Καταγραφής Βημάτων

[32] Για την καταγραφή βημάτων χρησιμοποιείται η συνάρτηση record(steps, arr) μέσω της οποίας γίνεται έλεγχος ότι υπάρχει τουλάχιστον ένα βήμα ώστε να πραγματοποιηθεί καταγραφή. Η συνάρτηση αυτή είναι υπεύθυνη για την παρακολούθηση κάθε αλγορίθμου ταξινόμησης και επιτρέπει την παρακολούθηση όλων των ενδιάμεσων καταστάσεων του αλγορίθμου. Η συνάρτηση καταγράφει μόνο τις απαραίτητες πληροφορίες χωρίς να επηρεάζει την απόδοση των αλγορίθμων.

Επίσης, γίνεται σύγκριση της κατάστασης του τρέχοντος βήματος με το προηγούμενο, αποθηκεύεται το στιγμιότυπο σε περίπτωση που υπάρχει κάποια αλλαγή, αποφεύγοντας τυχόν ίδια στιγμιότυπα. Η σύγκριση των λιστών γίνεται απευθείας και παράλληλα, ελέγχοντας αν steps[-1] != arr πριν την αποθήκευση. Η αποφυγή αποθήκευσης διπλότυπων στιγμιότυπων είναι σημαντική για δύο λόγους: πρώτον, εξοικονομεί μνήμη καθώς δεν αποθηκεύονται περιττά δεδομένα, και δεύτερον, παρέχει πιο καθαρή οπτικοποίηση στον χρήστη αφού κάθε βήμα αντιπροσωπεύει μία πραγματική αλλαγή στον πίνακα.

Τα δεδομένα κάθε βήματος αποθηκεύονται σε μια λίστα, όπου κάθε στοιχείο περιέχει ένα αντίγραφο της τρέχουσας κατάστασης του πίνακα μέσω της εντολής arr.copy(). Η χρήση αντιγράφου είναι πολύ σημαντική γιατί οι λίστες στην Python αλλάζουν κατά την εκτέλεση και χωρίς αντιγραφή θα υπήρχαν αναφορές στην ίδια λίστα που αλλάζει συνεχώς. Με αυτόν τον τρόπο, είναι δυνατή η αναδρομή σε οποιοδήποτε σημείο της εκτέλεσης του αλγορίθμου και είναι δυνατή η επισκόπηση της ακριβούς διάταξης των στοιχείων εκείνη τη στιγμή

Κάθε ένας από τους δέκα αλγορίθμους καλεί την συνάρτηση record στην αρχή για να καταγράψει την αρχική κατάσταση του πίνακα, μετά από κάθε αλλαγή στοιχείου για να καταγράψει την πρόοδο της ταξινόμησης, και στο τέλος εκτέλεσής του για να καταγράψει τον τελικό ταξινομημένο πίνακα. Αξίζει να σημειωθεί ότι ο αριθμός των καταγεγραμμένων βημάτων διαφέρει σημαντικά μεταξύ των αλγορίθμων, καθώς εξαρτάται από τη φύση του κάθε αλγορίθμου. Η διαδικασία καταγραφής επιτρέπει την πλήρη επισκόπηση της λειτουργίας κάθε αλγορίθμου και καθιστά δυνατή την οπτική σύγκριση της αποδοτικότητάς τους.

3.4.Β Διαχείριση και Επαλήθευση Εισόδου

Για τον έλεγχο εισόδου είναι υπεύθυνη η συνάρτηση validate_input() μέσω της οποίας γίνεται έλεγχος της εγκυρότητας των στοιχείων που εισάγει ο χρήστης. Η συνάρτηση αυτή αποτελεί το πιο σημαντικό επίπεδο προστασίας της εφαρμογής από λανθασμένα δεδομένα, επιτρέποντας την ομαλή λειτουργία των αλγορίθμων.

Συγκεκριμένα, πραγματοποιείται έλεγχος ότι ο χρήστης εισάγει ακέραιους αριθμούς γιατί οι αλγόριθμοι είναι σχεδιασμένοι γι' αυτό τον τύπο δεδομένων. Σε περίπτωση που ο χρήστης εισάγει δεκαδικούς αριθμούς ή χαρακτήρες, εμφανίζεται το μήνυμα "Παρακαλώ εισάγετε μόνο ακέραιους αριθμούς". Επίσης ελέγχει ότι το μέγεθος του πίνακα είναι από 1 μέχρι 500 για λόγους απόδοσης και καλύτερης αναπαράστασης της εργασίας. Το όριο των 500 στοιχείων επιλέγθηκε διότι μεγαλύτεροι πίνακες θα καθυστερούσαν σημαντικά την εκτέλεση, ειδικά για αλγορίθμους με μεγάλη πολυπλοκότητα.

Ακόμη, ελέγχει αν το μεγαλύτερο στοιχείο του πίνακα που εισάγει ο χρήστης είναι μεγαλύτερο από το μικρότερο στοιχείο που εισάγει, ώστε να υπάρχει εγκυρότητα στο εύρος αριθμών. Αν η μέγιστη τιμή είναι μικρότερη ή ίση της ελάχιστης, εμφανίζεται το μήνυμα "Η μέγιστη τιμή πρέπει να είναι μεγαλύτερη από την ελάχιστη". Τέλος, αν ο χρήστης δεν επιτρέπει διπλότυπους αριθμούς, γίνεται έλεγχος ότι οι τιμές που εισάγει για ελάχιστη και μέγιστη τιμή επαρκούν για την δημιουργία του

πίνακα. Για παράδειγμα, αν ο χρήστης εισάγει πίνακα 100 στοιχείων χωρίς διπλότυπα με εύρος 1-50, εμφανίζεται μήνυμα ότι το εύρος δεν επαρκεί.

Στην εργασία πρώτα γίνεται έλεγχος για τον τύπο δεδομένων, μετά για το μέγεθος, έπειτα για το εύρος τιμών και τέλος για την επάρκεια σε μοναδικές τιμές. Αυτή η σειρά εξασφαλίζει ότι ο χρήστης λαμβάνει το πιο σχετικό μήνυμα σφάλματος πρώτα. Αν ο έλεγχος αντιληφθεί κάποιο λάθος στην είσοδο του χρήστη, εμφανίζεται κατάλληλο μήνυμα και ο χρήστης μπορεί να διορθώσει την είσοδό του και να ξαναπροσπαθήσει χωρίς να χρειάζεται επανεκκίνηση της εφαρμογής.

3.4.Γ Δημιουργία Δεδομένων Εισόδου

Με την συνάρτηση create_array() πραγματοποιείται η δημιουργία των τριών τύπων δεδομένων που επιτρέπουν την αξιολόγηση της απόδοσης των αλγορίθμων σε διαφορετικές συνθήκες. Κάθε τύπος δεδομένων απεικονίζει ένα διαφορετικό σενάριο για κάθε έναν από τους δέκα αλγορίθμους.

Ο πρώτος τύπος είναι το Best Case όπου ο πίνακας είναι ήδη ταξινομημένος. Αυτό γίνεται με την εντολή arr.sort(). Αποτελεί την καλύτερη περίπτωση των περισσότερων αλγορίθμων, καθώς δεν χρειάζεται να γίνει κάποια εναλλαγή στοιχείου. Ωστόσο, κάποιοι από τους αλγορίθμους εξακολουθούν να έχουν την ίδια μεγάλη πολυπλοκότητα λόγω της φύσης τους.

Ο δεύτερος τύπος είναι το Average Case όπου υπάρχει τυχαία σειρά των στοιχείων του πίνακα. Αυτό γίνεται με την εντολή random.shuffle() και φαίνεται μία πιο ρεαλιστική και τυπική προσομοίωση των αλγορίθμων σε πραγματικά δεδομένα. Η συνάρτηση random.shuffle() επιβεβαιώνει ότι κάθε πιθανή διάταξη του πίνακα έχει ίση πιθανότητα να επιλεγεί, παρέχοντας έτσι ίση τύχη για κάθε στοιχείο του πίνακα. Αυτό το σενάριο είναι το πιο ρεαλιστικό για εφαρμογές στην πραγματικότητα.

Ο τρίτος τύπος είναι το Worst Case όπου υπάρχει αντίστροφα ταξινομημένος πίνακας. Αυτό γίνεται με την εντολή arr.sort(reverse=True). Σε αυτή την περίπτωση, οι περισσότεροι αλγόριθμοι έχουν μέγιστη πολυπλοκότητα και είναι πολύ αργοί. Η δοκιμή αυτού του σεναρίου είναι σημαντική για την κατανόηση των ορίων κάθε αλγορίθμου.

3.4.Δ Διαδοχική Εκτέλεση Αλγορίθμων

Η διαδοχική εκτέλεση των αλγορίθμων γίνεται με αναδρομική μέθοδο και χρήση της εντολής root.after() του tkinter για να είναι το GUI ανταποκρίσιμο. Η επιλογή της root.after() έγινε για λόγους απλότητας και αποφυγής προβλημάτων που θα μπορούσαν ίσως να προκύψουν. Χωρίς αυτή την προσέγγιση, το GUI θα σταματούσε να λειτουργεί κατά την εκτέλεση των αλγορίθμων, κάνοντας αδύνατη την αλληλεπίδραση του χρήστη με την εφαρμογή.

Όταν ο χρήστης πατήσει "Execute" τρέχει η συνάρτηση run_sorting() και αρχικοποιεί την λίστα με όλους του αλγορίθμους. Η λίστα αυτή περιέχει αναφορές στις συναρτήσεις των δέκα αλγορίθμων ταξινόμησης με τη σειρά που θα εκτελεστούν. Με την συνάρτηση run_next_algorithm() εκτελείται κάθε αλγόριθμος ξεχωριστά και διαδοχικά, ξέροντας ότι κάθε αλγόριθμος παίρνει ακριβώς τα ίδια δεδομένα εισόδου για να υπάρχει δίκαιη σύγκριση.

Η ροή εκτέλεσης ακολουθεί τα εξής βήματα: αρχικά δημιουργείται αντίγραφο του αρχικού πίνακα, στη συνέχεια εκτελείται ο τρέχων αλγόριθμος, μετά καταγράφονται τα αποτελέσματα και τέλος καλείται η root.after() για να προγραμματιστεί η εκτέλεση του επόμενου αλγορίθμου. Έτσι, ενημερώνεται το GUI για τον χρόνο και το βήμα κάθε αλγορίθμου και τα αποτελέσματα αποθηκεύονται στην συνάρτηση results(). Το πρόγραμμα σταματά όταν το current_algorithm_index είναι ίσο με το μέγεθος της λίστας, δείχνοντας ότι όλοι οι αλγόριθμοι έχουν ολοκληρωθεί. Μετά, με

την συνάρτηση show_results() εμφανίζονται τα αποτελέσματα της εκτέλεσης σε μορφή πίνακα και γραφήματος.

3.4.Ε Σύστημα Χρονομέτρησης

Για την χρονομέτρηση του κάθε αλγορίθμου χρησιμοπιείται η συνάρτηση time. Η επιλογή της time.perf_counter() αντί της time.time() έγινε διότι η πρώτη παρέχει μεγαλύτερη ακρίβεια και δεν επηρεάζεται από αλλαγές στο ρολόι του συστήματος, κάνοντας την ιδανική για μετρήσεις απόδοσης.

Συγκεκριμένα, η εντολή start = time.perf_counter() πραγματοποιεί την καταγραφή του χρόνου κάθε αλγορίθμου αμέσως πριν την εκτέλεσή του, ακολουθεί η εκτέλεση του αλγορίθμου κατά την οποία γίνονται όλες οι συγκρίσεις, εναλλαγές και καταγραφές βημάτων και τέλος με το elapsed = (time.perf_counter() - start) * 1000 καταγράφεται ο χρόνος που έκανε ο κάθε αλγόριθμος και αποθηκεύεται σε milliseconds.

Τα αποτελέσματα αποθηκεύονται στην λίστα self.results[name] στην οποία υπάρχει ο χρόνος κάθε αλγορίθμου και τα βήματα που έκανε. Η δομή αυτή επιτρέπει την εύκολη πρόσβαση στα αποτελέσματα κάθε αλγορίθμου με βάση το όνομά του. Η εμφάνιση γίνεται με ταξινόμηση βάσει του χρόνου κάθε αλγορίθμου ταξινόμησης και με την βοήθεια της συνάρτησης sorted_results, η οποία χρησιμοποιεί τον χρόνο εκτέλεσης σε αύξουσα σειρά, ώστε ο ταχύτερος αλγόριθμος να εμφανίζεται πρώτος.

3.4.Ζ Σύστημα Οπτικοποίησης

Το σύστημα οπτικοποίησης ανοίγει με το κουμπί “Show Graph” του GUI και είναι επίσης κεντραρισμένο στην οθόνη του χρήστη. Με το Combobox γίνεται η επιλογή του αλγορίθμου που ταξινομεί τον πίνακα και η εναλλαγή γίνεται με την εντολή change_algo(). Επειδή τα βήματα που μπορεί να κάνει κάθε αλγόριθμος μπορεί να είναι πάρα πολλά και με τα κουμπιά “Previous” και “Next” να μην είναι εύκολο να εμφανιστούν καθαρά όλα τα βήματα, έγινε χρήση της μεθόδου get_step_jump() με την οποία τα βήματα εναλλάσουν δυναμικά, ανάλογα με το μέγεθος του πίνακα που έχει εισάγει ο χρήστης.

Αυτό σημαίνει ότι εάν υπάρχουν 1000 βήματα και πατηθεί το κουμπί “Next”, προχωρά κατά 100 βήματα επιτρέποντας πιο γρήγορη πλοήγηση ώστε να διασφαλιστεί μια καλύτερη εικόνα γενικά. Η πλοήγηση μεταξύ βημάτων γίνεται με τις μεθόδους first_step(), prev_step(), next_step() και last_step().

Επίσης, με την μέθοδο draw_plot() γίνεται εκκαθάριση του υπάρχον widget με την εντολή w.destroy(), ανάκτηση του παρόντος στιγμιότυπου με την εντολή arr = steps[current_step], δημιουργία νέου figure με την εντολή figsize=(7, 4), σχεδίαση bar chart με την εντολή ax.bar(range(len(arr)), arr), πρόσθεση τίτλου με την εντολή "Algorithm Name Current Step / Total Steps", ενσωμάτωση στο tkinter με την εντολή FigureCanvasTkAgg και τέλος εκκαθάριση του figure με την εντολή plt.close(fig) για να μην υπάρχει θέμα στην μνήμη.

ΚΕΦΑΛΑΙΟ 4 ΠΕΙΡΑΜΑΤΙΚΑ ΑΠΟΤΕΛΕΣΜΑΤΑ

4.1 Εισαγωγή

Σε αυτό το κεφάλαιο παρουσιάζονται τα πειραματικά αποτελέσματα από τους δέκα αλγορίθμους ταξινόμησης, εκτελώντας τον κώδικα. Σκοπός ουσιαστικά των συγκεκριμένων πειραμάτων είναι να επαληθευτεί το θεωρητικό υπόβαθρο που παρουσιάστηκε στο Κεφάλαιο 2 και να εξεταστεί με πραγματικά δεδομένα πως συμπεριφέρεται κάθε αλγόριθμος στην πράξη, σύμφωνα με τα δεδομένα που του δίνονται και την αρχική μορφή που έχει ο πίνακας. Με αυτό τον τρόπο θα κατανοηθεί σε ποιες περιπτώσεις είναι κατάλληλος ο κάθε ένας από αυτούς.

Τα πειράματα χωρίζονται σε τρεις κατηγορίες με βάση το μέγεθος του πίνακα: σε μικρό πίνακα με 20 στοιχεία, σε μεσαίο πίνακα με 100 στοιχεία και σε μεγάλο πίνακα με 500 στοιχεία που αποτελεί και το μέγιστο όριο πίνακα που επιτρέπεται στον κώδικα. Για κάθε μέγεθος πίνακα εκτελείται δοκιμή και στις τρεις περιπτώσεις που μπορεί να είναι ο πίνακας, δηλαδή σε Best Case, Average Case και Worst Case. Καταγράφεται ο χρόνος εκτέλεσης κάθε αλγορίθμου σε milliseconds και παρατηρούνται τα βήματα που πραγματοποιήθηκαν για την ταξινόμηση του πίνακα.

Παρακάτω παρουσιάζονται αναλυτικά τα αποτελέσματα για κάθε κατηγορία πίνακα, με ανάλογους χρόνους, γραφήματα οπτικοποίησης και διαγράμματα σύγκρισης. Στο τέλος του κεφαλαίου συγκρίνονται όλα τα αποτελέσματα σε ένα μέρος και εξάγονται κάποια γενικά συμπεράσματα για την απόδοση κάθε αλγορίθμου, σύμφωνα με το θεωρητικό του υπόβαθρο.

4.2 Δοκιμή σε Μικρό Πίνακα

Στην πρώτη σειρά πειραμάτων εξετάστηκε η απόδοση των αλγορίθμων σε μικρό πίνακα με 20 στοιχεία, με ελάχιστη τιμή 1 και μέγιστη τιμή 50, χωρίς διπλότυπους αριθμούς. Εκτελέστηκαν οι δέκα αλγόριθμοι στις τρεις περιπτώσεις Best Case, Average Case και Worst Case και καταγράφηκαν οι χρόνοι εκτέλεσης. Σε μικρό πίνακα αναμένεται οι διαφορές μεταξύ των αλγορίθμων να είναι μικρές, καθώς ο όγκος των δεδομένων δεν είναι αρκετός για να φανούν οι αδυναμίες των πιο αργών αλγορίθμων.

Πίνακας 4.2.Α Μικρός Πίνακας

Αλγόριθμος	Best Case	Average Case	Worst Case
Insertion Sort	0.007	0.034	0.977
Selection Sort	0.012	0.019	0.020
Bubble Sort	0.012	0.033	0.074
Bucket Sort	0.013	0.018	0.026
Counting Sort	0.014	0.021	0.023
Shell Sort	0.015	0.031	0.039
Quick Sort	0.021	0.026	0.032
Radix Sort	0.024	0.025	0.030
Merge Sort	0.034	0.049	0.060
Heap Sort	0.045	0.039	0.076

Από τα πειραματικά αποτελέσματα στον μικρό πίνακα με 20 στοιχεία παρατηρείται πως οι χρόνοι εκτέλεσης όλων των αλγορίθμων είναι πολύ μικροί και οι χρονικές τους πολυπλοκότητες δεν επηρεάζουν τόσο τη συνολική εικόνα. Όπως παρουσιάστηκε στο Κεφάλαιο 2, η πολυπλοκότητα των αλγορίθμων γίνεται αισθητή πιο έντονα σε μεγαλύτερα μεγέθη δεδομένων.

Οι αλγόριθμοι με $O(n^2)$ πολυπλοκότητα όπως ο Bubble Sort, Selection Sort και Insertion Sort έχουν πολύ καλή απόδοση σε τόσο μικρό πίνακα γιατί ο συνολικός αριθμός συγκρίσεων και μετακινήσεων είναι αρκετά μικρός. Ο Bubble Sort και ο Selection Sort έχουν σταθερή λειτουργία και είναι πολύ γρήγοροι, ενώ ο Insertion Sort είναι ο πιο αποδοτικός από τους τρεις στο Best Case επειδή σε ήδη ταξινομημένο πίνακα πραγματοποιούνται οι λιγότερες μετακινήσεις στοιχείων.

Οι αλγόριθμοι με O(nlogn) πολυπλοκότητα όπως ο Merge Sort, Quick Sort, Heap Sort και Shell Sort έχουν καλύτερη θεωρητική απόδοση, αλλά σε τόσο μικρό πίνακα το πλεονέκτημά τους δεν είναι εμφανές. Μάλιστα, σε κάποιες περιπτώσεις είναι ελαφρώς πιο αργοί από τους απλούστερους αλγορίθμους λόγω της αναδρομικότητας που έχουν και των πρόσθετων δομών δεδομένων που χρησιμοποιούν.

Τέλος, οι αλγόριθμοι με O(n) πολυπλοκότητα όπως ο Counting Sort, Radix Sort και Bucket Sort δεν εμφανίζουν κάποια τεράστια διαφορά στον μικρό πίνακα. Αυτό συμβαίνει εξαιτίας του τρόπου που λειτουργούν και της προετοιμασίας που πραγματοποιείται για την ταξινόμηση του πίνακα. Σε τόσο μικρό πίνακα η διάκριση από τους αλγορίθμους με $O(n^2)$ πολυπλοκότητα είναι δύσκολη, αφού το πλεονέκτημά τους φαίνεται κυρίως σε μεγάλα σύνολα δεδομένων.

Συμπερασματικά, σε μικρό πίνακα όλοι οι αλγόριθμοι φαίνονται γρήγοροι και ασχέτως της θεωρητικής τους πολυπλοκότητας, όλοι καταφέρνουν να ταξινομήσουν τον πίνακα σε πολύ σύντομο χρονικό διάστημα. Η μόνη εξαίρεση είναι ο Insertion Sort στο Worst Case που ξεχωρίζει αρνητικά, δείχνοντας ότι ακόμα και σε μικρά δεδομένα η αρχική διάταξη μπορεί να επηρεάσει σημαντικά την απόδοση ορισμένων αλγορίθμων.

4.3 Δοκιμή σε Μεσαίο Πίνακα

Στη δεύτερη σειρά πειραμάτων εξετάστηκε η απόδοση των αλγορίθμων σε μεσαίο πίνακα με 100 στοιχεία, με ελάχιστη τιμή 1 και μέγιστη τιμή 200, χωρίς διπλότυπους αριθμούς. Εκτελέστηκαν οι δέκα αλγόριθμοι στις τρεις περιπτώσεις Best Case, Average Case και Worst Case και καταγράφηκαν οι χρόνοι εκτέλεσης. Σε μεσαίο πίνακα υπάρχουν ελαφρώς μεγαλύτερες διαφορές σε σχέση με τον μικρό πίνακα και υπάρχει πιο ξεκάθαρη διάκριση μεταξύ της απόδοσης των αλγορίθμων.

Πίνακας 4.2.Β Μεσαίος Πίνακας

Αλγόριθμος	Best Case	Average Case	Worst Case
Insertion Sort	0.032	1.755	2.852
Bucket Sort	0.052	0.069	0.066
Counting Sort	0.053	0.072	0.072
Radix Sort	0.088	0.082	0.078
Shell Sort	0.134	0.443	0.383
Selection Sort	0.135	0.187	0.193
Bubble Sort	0.165	6.765	2.625
Merge Sort	0.247	0.728	0.873
Quick Sort	0.310	0.237	0.258
Heap Sort	1.461	0.434	0.366

Από τα πειραματικά αποτελέσματα στον πίνακα μεσαίου μεγέθους αρχίζουν να παρατηρούνται πιο έντονες διαφορές μεταξύ των αλγορίθμων. Σε αυτό το μέγεθος μπορεί να κατανοηθεί καλύτερα η πολυπλοκότητα που αναφέρθηκε στο Κεφάλαιο 2, αφού ο αριθμός των συγκρίσεων που πραγματοποιούνται αλλάζει αρκετά τους χρόνους των αλγορίθμων και μπορούν να διακριθούν πιο εύκολα οι αργοί από τους γρήγορους.

Οι αλγόριθμοι με τετραγωνική πολυπλοκότητα δυσκολεύονται περισσότερο με 100 στοιχεία. Ο χρόνος του Bubble Sort αυξάνεται πολύ στο Average και στο Worst Case γιατί ο αριθμός των συγκρίσεων που πραγματοποιούνται είναι πολύ μεγάλος, ενώ στο Best Case αποδίδει κάπως καλύτερα. Ο Selection Sort παραμένει σταθερός και στις τρεις περιπτώσεις αφού έχει σταθερό αριθμό συγκρίσεων ασχέτως της ταξινόμησης του πίνακα. Ο Insertion Sort είναι αρκετά αποδοτικός στο Best Case, ενώ στο Average και Worst Case έχει αυξημένο χρόνο.

Οι αλγόριθμοι με O(nlogn) πολυπλοκότητα εμφανίζουν μεγάλο πλεονέκτημα σε σχέση με τους $O(n^2)$ αλγορίθμους. Ο Quick Sort έχει πολύ καλή απόδοση στο Average Case, ενώ στο Worst Case μπορεί να επηρεαστεί από την επιλογή του pivot. Ο Merge Sort είναι σταθερός και γρήγορος σε όλες τις περιπτώσεις. Ο Heap Sort έχει χαμηλούς χρόνους με τη βοήθεια της δομής σωρού που χρησιμοποιεί και ο Shell Sort κινείται σε παρόμοια χρονικά πλαίσια με τους υπόλοιπους αλγορίθμους αυτής της κατηγορίας.

Οι γραμμικοί αλγόριθμοι ξεχωρίζουν πλέον καθαρά από τους τετραγωνικής πολυπλοκότητας αλγορίθμους. Ο Counting Sort είναι πολύ γρήγορος σε αυτό το μέγεθος πίνακα αφού έχει θεωρητική πολυπλοκότητα O(n+k). Ο Radix Sort είναι εξίσου γρήγορος αφού η πολυπλοκότητα O(d*n) με λίγα ψηφία d παραμένει πολύ αποδοτική. Ο Bucket Sort είναι επίσης εξαιρετικά γρήγορος σε ομοιόμορφα κατανεμημένα δεδομένα.

4.4 Δοκιμή σε Μεγάλο Πίνακα

Στην τρίτη σειρά πειραμάτων εξετάστηκε η απόδοση των αλγορίθμων σε μεγάλο πίνακα με 500 στοιχεία, με ελάχιστη τιμή 1 και μέγιστη τιμή 1000, χωρίς διπλότυπους αριθμούς. Εκτελέστηκαν οι δέκα αλγόριθμοι στις τρεις περιπτώσεις Best Case, Average Case και Worst Case και καταγράφηκαν οι χρόνοι εκτέλεσης. Σε αυτή την περίπτωση οι διαφορές ανάμεσα στους αλγορίθμους είναι μεγάλες και προφανείς.

Πίνακας 4.2.Γ Μεγάλος Πίνακας

Αλγόριθμος	Best Case	Average Case	Worst Case
Insertion Sort	0.282	246.630	475.456
Radix Sort	0.373	0.297	0.451
Bucket Sort	0.418	1.191	1.254
Counting Sort	0.679	1.150	1.343
Shell Sort	2.221	17.392	12.463
Merge Sort	2.743	12.365	15.324
Selection Sort	3.282	7.019	3.948
Bubble Sort	4.081	169.782	370.722
Quick Sort	5.278	5.783	4.716
Heap Sort	9.255	10.917	10.726

Από τα πειραματικά αποτελέσματα στον πίνακα μεγάλου μεγέθους μπορεί πλέον να υπάρχει μία απόλυτα ξεκάθαρη και σαφής εικόνα για την πολυπλοκότητα των αλγορίθμων. Σε αυτό το μέγεθος πίνακα, ο αριθμός των συγκρίσεων και των μετακινήσεων αυξάνεται κατά πολύ μεγάλο βαθμό και οι αργοί αλγόριθμοι καθυστερούν αισθητά, ενώ οι γρήγοροι διατηρούν χαμηλούς χρόνους εκτέλεσης.

Οι αλγόριθμοι με τετραγωνική πολυπλοκότητα έχουν πλέον πολύ μεγάλο χρόνο εκτέλεσης σε σχέση με τους υπόλοιπους. Ο χρόνος του Bubble Sort αυξάνεται δραματικά γιατί πραγματοποιούνται υπερβολικές συγκρίσεις και ανταλλαγές στοιχείων, καθιστώντας τον έναν από τους πιο αργούς αλγορίθμους σε μεγάλο πίνακα. Ο Selection Sort παραμένει σταθερός στις τρεις περιπτώσεις, όμως οι συγκρίσεις που πραγματοποιούνται είναι πλέον πάρα πολλές και καταλήγει να είναι αργός. Ο Insertion Sort έχει τη χειρότερη απόδοση, ειδικά στο Worst Case που παρουσιάζει τον πιο αργό χρόνο από όλους τους αλγορίθμους στο πείραμα.

Οι αλγόριθμοι με πολυπλοκότητα O(nlogn) έχουν προφανές πλεονέκτημα σε σχέση με τους αλγορίθμους τετραγωνικής πολυπλοκότητας. Ο Quick Sort αποδίδει πολύ καλά στο Average Case και αποτελεί γενικά καλή επιλογή εφόσον υπάρχει καλό pivot. Ο Merge Sort παραμένει σταθερός και αποδοτικός λόγω της αναδρομικότητάς του και της ισορροπημένης διαίρεσης που πραγματοποιεί. Ο Heap Sort είναι επίσης αρκετά γρήγορος, αν και πραγματοποιούνται παραπάνω κινήσεις λόγω της δομής σωρού. Ο Shell Sort αποδίδει σχετικά καλά, αλλά είναι πιο αργός από τους προηγούμενους.

Οι γραμμικοί αλγόριθμοι ξεχωρίζουν καθαρά ως οι καλύτεροι στον μεγάλο πίνακα, αφού οι χρόνοι τους παραμένουν πολύ χαμηλοί ασχέτως του μεγέθους των δεδομένων. Ο Counting Sort είναι εξαιρετικά γρήγορος με χρόνο σχεδόν αμετάβλητο και απόδοση εντυπωσιακή. Ο Radix Sort συνεχίζει άψογα με τους μικρότερους χρόνους, καθιστώντας τον καλύτερο αλγόριθμο του πειράματος συνολικά. Ο Bucket Sort έχει εξαίρετη απόδοση με χρόνους πολύ κοντά στον Counting Sort, δείχνοντας την αποτελεσματικότητα των μη συγκριτικών αλγορίθμων ταξινόμησης σε ακέραιους αριθμούς με περιορισμένο εύρος.

4.5 Συγκριτική Ανάλυση

Στην παρούσα ενότητα συγκεντρώνονται και αναλύονται συγκριτικά όλα τα πειραματικά αποτελέσματα από τις τρεις κατηγορίες πινάκων που εξετάστηκαν. Σκοπός είναι να παρατηρηθεί πως κλιμακώνεται η απόδοση κάθε αλγορίθμου καθώς αυξάνεται το μέγεθος των δεδομένων και να επαληθευτεί η θεωρητική πολυπλοκότητα που αναλύθηκε στο Κεφάλαιο 2. Μέσω αυτής της σύγκρισης μπορούν να εξαχθούν γενικά συμπεράσματα για το πότε είναι κατάλληλος ο κάθε αλγόριθμος και ποιες είναι οι ιδανικές συνθήκες χρήσης του.

Επικεντρώνεται η ανάλυση στο Average Case καθώς αποτελεί την πιο ρεαλιστική προσέγγιση για την αξιολόγηση των αλγορίθμων. Στην πράξη, τα δεδομένα που δέχονται οι αλγόριθμοι σπάνια είναι ήδη ταξινομημένα (Best Case) ή πλήρως αντίστροφα ταξινομημένα (Worst Case), αλλά συνήθως έχουν τυχαία σειρά. Στον παρακάτω πίνακα συγκεντρώνονται οι χρόνοι εκτέλεσης όλων των αλγορίθμων για τα τρία μεγέθη πινάκων στο Average Case:

Πίνακας 4.5.Α: Συγκριτικός πίνακας χρόνων εκτέλεσης στο Average Case

Αλγόριθμος	20 στοιχεία (ms)	100 στοιχεία (ms)	500 στοιχεία (ms)
Bucket Sort	0.018	0.069	1.191
Selection Sort	0.019	0.187	7.019
Counting Sort	0.021	0.072	1.150
Radix Sort	0.025	0.082	0.297
Quick Sort	0.026	0.237	5.783

Shell Sort	0.031	0.443	17.392
Bubble Sort	0.033	6.765	169.782
Insertion Sort	0.034	1.755	246.630
Heap Sort	0.039	0.434	10.917
Merge Sort	0.049	0.728	12.365

Από το γράφημα μπορεί να παρατηρηθεί καθαρά ο τρόπος που αυξάνεται ο χρόνος των αλγορίθμων τετραγωνικής, λογαριθμικογραμμικής και γραμμικής πολυπλοκότητας.

Αλγόριθμοι με $O(n^2)$ Πολυπλοκότητα

Οι αλγόριθμοι Bubble Sort, Selection Sort και Insertion Sort παρουσιάζουν σημαντική αύξηση του χρόνου εκτέλεσης καθώς μεγαλώνει το μέγεθος του πίνακα στο Average Case. Ο Insertion Sort ξεκινάει με πολύ χαμηλό χρόνο για μικρούς πίνακες αλλά καταλήγει να απαιτεί πολύ περισσότερο χρόνο σε μεγαλύτερους πίνακες. Ο Bubble Sort ακολουθεί παρόμοια πορεία με έντονη αύξηση χρόνου. Ο Selection Sort είναι ο πιο σταθερός από τους τρεις, παρουσιάζοντας μικρότερες διακυμάνσεις στην αύξηση του χρόνου εκτέλεσης.

Αυτή η συμπεριφορά επιβεβαιώνει τη θεωρητική πολυπλοκότητα $O(n^2)$. Όταν το μέγεθος του πίνακα αυξάνεται σημαντικά, ο χρόνος εκτέλεσης αυξάνεται ακόμα πιο έντονα. Και οι τρεις αλγόριθμοι ακολουθούν αυτό το μοτίβο τετραγωνικής πολυπλοκότητας, με κάποιες διαφορές ανάλογα με τον τρόπο που ο κάθε αλγόριθμος χειρίζεται τα δεδομένα.

Αλγόριθμοι με O(nlogn) Πολυπλοκότητα

Οι αλγόριθμοι Quick Sort, Merge Sort, Heap Sort και Shell Sort παρουσιάζουν πολύ καλύτερη κλιμάκωση στο Average Case. Ο Quick Sort ξεκινάει με πολύ χαμηλό χρόνο για μικρούς πίνακες και παραμένει σχετικά γρήγορος ακόμα και σε μεγάλους πίνακες. Ο Merge Sort και ο Heap Sort ακολουθούν παρόμοια πορεία με σταδιακή αύξηση χρόνου. Ο Shell Sort παρουσιάζει μεγαλύτερη αύξηση χρόνου σε σχέση με τους υπόλοιπους αλγορίθμους αυτής της κατηγορίας.

Η θεωρητική πολυπλοκότητα O(nlogn) επιβεβαιώνεται από τη συμπεριφορά των αλγορίθμων. Όταν το μέγεθος του πίνακα αυξάνεται σημαντικά, ο χρόνος εκτέλεσης αυξάνεται με πολύ πιο ήπιο ρυθμό σε σχέση με τους $O(n^2)$ αλγορίθμους. Οι διαφορές στην αύξηση οφείλονται στις αναδρομικές κλήσεις και τις πρόσθετες δομές δεδομένων που χρησιμοποιούν οι αλγόριθμοι.

Επίσης, ο Quick Sort είναι ο ταχύτερος από την κατηγορία του στο Average Case, αποτελώντας τον πιο αποδοτικό αλγόριθμο ταξινόμησης στον τομέα του. Ο Merge Sort και ο Heap Sort έχουν παρόμοια απόδοση, ενώ ο Shell Sort υστερεί λίγο.

Αλγόριθμοι με O(n) Πολυπλοκότητα

Οι αλγόριθμοι Counting Sort, Radix Sort και Bucket Sort παρουσιάζουν απίστευτη κλιμάκωση στο Average Case. Ο Radix Sort ξεκινάει με πολύ χαμηλό χρόνο για μικρούς πίνακες και παραμένει εξαιρετικά γρήγορος ακόμα και σε μεγάλους πίνακες. Ο Counting Sort και ο Bucket Sort επίσης διατηρούν καλή απόδοση, αν και παρουσιάζουν μεγαλύτερη αύξηση χρόνου σε σχέση με τον Radix Sort.

Η θεωρητική πολυπλοκότητα O(n) ή O(n+k) έχει γραμμική αύξηση. Όταν το μέγεθος του πίνακα αυξάνεται σημαντικά, ο χρόνος εκτέλεσης αυξάνεται με πολύ αργό ρυθμό. Ο Radix Sort παρουσιάζει τη πιο σταθερή συμπεριφορά, ενώ ο Counting Sort και ο Bucket Sort επηρεάζονται περισσότερο από την αύξηση του μεγέθους στοιχείων του πίνακα.

Ο Radix Sort παρουσιάζει την καλύτερη κλιμάκωση από όλους τους αλγορίθμους, καθώς η πολυπλοκότητά του O(d*n) επηρεάζεται μόνο από τον αριθμό ψηφίων d που παραμένει σταθερός. Αντίθετα, ο Counting Sort και ο Bucket Sort επηρεάζονται περισσότερο από το εύρος τιμών k που αυξάνεται σε μεγαλύτερους πίνακες, γι' αυτό αυξάνεται τόσο ο χρόνος τους.

Σύνοψη

Συγκρίνοντας τα πειραματικά αποτελέσματα του Average Case με τη θεωρητική πολυπλοκότητα, μπορεί να παρατηρηθεί ότι συνδέονται. Οι αλγόριθμοι με τετραγωνική πολυπλοκότητα $O(n^2)$ παρουσιάζουν πράγματι μεγάλη αύξηση του χρόνου εκτέλεσης, με τον Insertion Sort και Bubble Sort να είναι εξαιρετικά αργοί σε μεγάλα μεγέθη. Οι αλγόριθμοι με πολυπλοκότητα O(nlogn) είναι όπως παρουσιάστηκαν στη θεωρία, με τον Quick Sort να ξεχωρίζει ως ο ταχύτερος από την κατηγορία του.

Οι γραμμικοί αλγόριθμοι είναι με μεγάλη διαφορά οι πιο γρήγοροι σε ακέραιους αριθμούς με περιορισμένο εύρος, με τον Radix Sort να έχει την καλύτερη συνολική απόδοση σε μεγάλα μεγέθη. Ωστόσο, πρέπει να αναφερθεί ότι οι χρόνοι τους επηρεάζονται από το εύρος τιμών και τον αριθμό ψηφίων, όπως παρουσιάστηκε στη θεωρία.

Κάτι που παρατηρήθηκε είναι ότι σε μικρό πίνακα η θεωρητική πολυπλοκότητα δεν είναι αρκετή για να δείξει την πραγματική απόδοση των αλγορίθμων, καθώς το κόστος σε υπολογιστικούς πόρους της κάθε υλοποίησης παίζει σημαντικό ρόλο. Ωστόσο, καθώς το μέγεθος του πίνακα αυξάνεται, η θεωρητική πολυπλοκότητα αποτελεί τον κύριο παράγοντα και οι προβλέψεις της θεωρίας επαληθεύονται πλήρως.

4.6 Παρατηρήσεις

Μέσα από την εκτέλεση των παραπάνω πειραμάτων και την ανάλυση των αποτελεσμάτων που προέκυψαν, υπάρχουν κάποιες σημαντικές παρατηρήσεις που πρέπει να αναφερθούν. Μέσω των παρατηρήσεων αυτών θα κατανοηθεί καλύτερα η συμπεριφορά των αλγορίθμων ταξινόμησης.

Αρχικά, παρατηρείται πως δεν αρκεί η θεωρητική πολυπλοκότητα για να προβλεφθεί η πραγματική απόδοση των αλγορίθμων ταξινόμησης, ειδικά στην περίπτωση που ο πίνακας είναι πολύ μικρός. Διαπιστώθηκε πως σε μικρό πίνακα αλγόριθμοι ταξινόμησης με τετραγωνική πολυπλοκότητα κατέληξαν να είναι πιο γρήγοροι από αλγορίθμους με λογαριθμικογραμμική πολυπλοκότητα. Αυτό συμβαίνει γιατί οι πιο σύνθετοι αλγόριθμοι απαιτούν περισσότερο χρόνο για την ταξινόμηση του πίνακα, λόγω του τρόπου που λειτουργούν. Όμως όσο το μέγεθος του πίνακα αυξάνεται, τόσο πιο κοντά πλησιάζεται η θεωρητική πολυπλοκότητα.

Επίσης, η αρχική διάταξη των δεδομένων του πίνακα παίζει πολύ σημαντικό ρόλο στην απόδοση ορισμένων αλγορίθμων ταξινόμησης. Αυτό σημαίνει ότι εάν είναι γνωστή η δομή των δεδομένων, μπορεί να επιλεγεί με μεγαλύτερη ευκολία ο πιο αποδοτικός αλγόριθμος για τη συγκεκριμένη περίπτωση. Διαπιστώθηκε πως οι αλγόριθμοι γραμμικής πολυπλοκότητας είναι οι γρηγορότεροι σε μεγάλους πίνακες ακεραίων, απλά η απόδοσή τους εξαρτάται αρκετά από το εύρος των αριθμών του πίνακα.

Τέλος, κατανοήθηκε πως δεν υπάρχει καλύτερος αλγόριθμος για κάθε περίπτωση. Η επιλογή του αλγορίθμου που θα χρησιμοποιηθεί εξαρτάται κυρίως από το μέγεθος του πίνακα, τον τύπο δεδομένων του, το εύρος τιμών που έχει και την αρχική του διάταξη.

ΚΕΦΑΛΑΙΟ 5 ΣΥΜΠΕΡΑΣΜΑΤΑ

Η παρούσα εργασία είχε ως βασικό στόχο την υλοποίηση και γραφική αναπαράσταση των δέκα αλγορίθμων ταξινόμησης που εξετάστηκαν και μελετήθηκαν. Μέσα από την ανάπτυξη και τη μελέτη που πραγματοποιήθηκε στην εφαρμογή, διαπιστώθηκε ότι η θεωρία κάθε αλγορίθμου επαληθεύεται στην πραγματικότητα. Ωστόσο, η απόδοση εξαρτάται από πολλούς παράγοντες όπως η διάταξη των δεδομένων του πίνακα, ο τρόπος χρήσης της μνήμης και ο τρόπος υλοποίησης του κάθε αλγορίθμου.

Από τα πειραματικά αποτελέσματα διαπιστώθηκε πως οι απλοί αλγόριθμοι όπως ο Bubble Sort και Insertion Sort μπορούν να είναι πολύ αποτελεσματικοί σε μικρούς ή σχεδόν ταξινομημένους πίνακες ακεραίων, παρόλο που στη θεωρία έχουν πολυπλοκότητα $O(n^2)$. Από την άλλη όμως παρατηρήθηκε πως σε μεγάλα δεδομένα υπάρχουν αποδοτικοί αλγόριθμοι με μικρότερη πολυπλοκότητα όπως ο Quick Sort και ο Merge Sort που έχουν πολυπλοκότητα O(nlogn). Επίσης, μη συγκριτικοί αλγόριθμοι όπως ο Counting Sort και ο Radix Sort είχαν πολύ καλή απόδοση όταν το εύρος τιμών στον πίνακα είναι περιορισμένο. Σαφώς, παρατηρείται ότι δεν υπάρχει ένας και μοναδικός κατάλληλος αλγόριθμος για όλες τις περιπτώσεις που μπορεί να προκύψουν.

Με τη βοήθεια της οπτικοποίησης ήταν δυνατή η κατανόηση σε μεγάλο βαθμό των παραπάνω αποτελεσμάτων. Επιτεύχθηκε η παρατήρηση της εξέλιξης κάθε αλγορίθμου στη διαδικασία ταξινόμησης του πίνακα, του τρόπου δηλαδή που ταξινομεί τα δεδομένα και του χρόνου που απαιτείται για την ταξινόμησή τους. Παρατηρήθηκε ότι το γράφημα κάθε αλγορίθμου από την αρχή της ταξινόμησης του πίνακα μέχρι την ολοκλήρωση της ταξινόμησης, ήταν μοναδικό και διαφορετικό από τα υπόλοιπα.

Η εργασία θεωρείται ότι πέτυχε τον στόχο της και μπορεί να θεωρηθεί ένα σημαντικό εργαλείο μάθησης και κατανόησης των αλγορίθμων ταξινόμησης. Δίνεται η δυνατότητα στον χρήστη να πειραματιστεί και να εξάγει μόνος του τα συμπεράσματα, συγκρίνοντας τους αλγορίθμους και κατανοώντας τη λειτουργία του καθενός.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, 3rd ed. Cambridge, MA: MIT Press, 2009.

[2] D. E. Knuth, The Art of Computer Programming, Vol. 3: Sorting and Searching, 2nd ed. Reading, MA: Addison-Wesley, 1998.

[3] R. Sedgewick, Algorithms, 2nd ed. Reading, MA: Addison-Wesley, 1988.

[4] M. R. Koenke, "Asymptotic Analysis and Comparison of Sorting Algorithms," Medium, 2021.

[5] K. Tsichlas, Design and Analysis of Algorithms. Athens: Kallipos, Open Academic Editions, 2015.

[6] R. Sedgewick and P. Flajolet, An Introduction to the Analysis of Algorithms, 2nd ed. Reading, MA: Addison-Wesley, 2013.

[7] P. Bachmann, Die Analytische Zahlentheorie. Leipzig: B. G. Teubner, 1894.

[8] J. Kleinberg and E. Tardos, Algorithm Design. Boston, MA: Pearson Education, 2006.

[9] H. S. Wilf, Algorithms and Complexity, 2nd ed. Wellesley, MA: A K Peters, 2002.

[10] S. Arora and B. Barak, Computational Complexity: A Modern Approach. Cambridge, UK: Cambridge University Press, 2009.

[11] S. S. Skiena, The Algorithm Design Manual, 2nd ed. London: Springer-Verlag, 2008.

[12] B. N. Miller and D. L. Ranum, Problem Solving with Algorithms and Data Structures using Python, 2nd ed. Franklin, Beedle & Associates, 2011.

[13] C. A. Shaffer, A Practical Introduction to Data Structures and Algorithm Analysis, 3rd ed. Blacksburg, VA: Virginia Tech, 2011.

[14] D. E. Knuth, "Big Omicron and Big Omega and Big Theta," ACM SIGACT News, vol. 8, no. 2, 1976.

[15] J. J. McConnell, Analysis of Algorithms: An Active Learning Approach, 2nd ed. Sudbury, MA: Jones and Bartlett Publishers, 2008.

[16] N. Wirth, Algorithms + Data Structures = Programs. Englewood Cliffs, NJ: Prentice-Hall, 1976.

[17] V. Estivill-Castro and D. Wood, "A Survey of Adaptive Sorting Algorithms," ACM Computing Surveys, vol. 24, no. 4, 1992.

[18] S. Sundaramoorthy and G. Karunanidhi, "A Systematic Analysis on Performance and Computational Complexity of Sorting Algorithms," Discover Computing, vol. 28, no. 250, 2025.

[19] R. Ali, "Sorting and Classification of Sorting Algorithms," International Journal of Advanced Research in Computer Science, vol. 8, no. 5, 2017.

[20] D. Samanta, Classic Data Structures, 2nd ed. New Delhi: PHI Learning, 2009.

[21] M. A. Weiss, Data Structures and Algorithm Analysis in C++, 4th ed. Boston, MA: Pearson, 2014.

[22] C. A. R. Hoare, "Quicksort," The Computer Journal, vol. 5, no. 1, 1962.

[23] D. L. Shell, "A High-Speed Sorting Procedure," Communications of the ACM, vol. 2, no. 7, 1959.

[24] J. Boothroyd, "Algorithm 230: Matrix Permutation," Communications of the ACM, vol. 7, no. 11, 1964.

[25] K. E. Batcher, "Sorting Networks and their Applications," in Proceedings of the AFIPS Spring Joint Computer Conference, 1968.

[26] A. Andersson and S. Nilsson, "A New Efficient Radix Sort," in Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science, 1994.

[27] Python Software Foundation, Python 3.14.0 Documentation, 2025.

[28] Python Software Foundation, "tkinter — Python Interface to Tcl/Tk."

[29] M. T. Goodrich, R. Tamassia, and D. M. Mount, Data Structures and Algorithms in C++, 2nd ed. Hoboken, NJ: Wiley, 2011.

[30] M. Sipser, Introduction to the Theory of Computation, 3rd ed. Boston, MA: Cengage Learning, 2013.

[31] C. H. Papadimitriou, Computational Complexity. Reading, MA: Addison-Wesley, 1994.

[32] C. M. Bishop, Pattern Recognition and Machine Learning. New York: Springer, 2006.

[33] A. Aslam, M. S. Ansari, and S. Varshney, "Non-Partitioning Merge-Sort: Performance Enhancement by Elimination of Division in Divide-and-Conquer Algorithm," in Proceedings of the 2nd International Conference on Information and Communication Technology, 2016.

[34] J. Wyss-Gallifent, "CMSC 351: MergeSort," University of Maryland, Department of Computer Science, Lecture Notes.

[35] Ryerson University, "Insertion Sort," Department of Computer Science, Lecture Notes.

[36] PMT, "OCR Computer Science AS Level: Sorting Algorithms," Physics & Maths Tutor.

[37] C. Komalasari and W. Istiono, "A Comparative Study of Cocktail Sort and Insertion Sort," in International Conference on Information Management and Technology (ICIMTech), 2020.

[38] Matplotlib Development Team, Matplotlib 3.10.7 Documentation, 2025.

[39] ISO/IEC, ISO/IEC 14882:2020 Programming Languages – C++, 2020.

[40] Oracle Corporation, Java SE Documentation, 2025.

[41] Tcl Developer Xchange, Tcl/Tk Documentation, 2025.

[42] M. Waskom, Seaborn: Statistical Data Visualization.

[43] The Pandas Development Team, pandas documentation.

[44] M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator," ACM Transactions on Modeling and Computer Simulation, vol. 8, no. 1, 1998.