Τεχνολογικο Εκπαιδευτικο Ιδρυμα Πελοποννησου
Σχολη Τεχνολογικων Εφαρμογων
Τμημα Μηχανικων Πληροφορικης Τ.Ε.

Αλγόριθμοι Ταξινόμησης Γραμμικού Χρόνου και Εκτέλεσης

Μελέτη και υλοποίηση

Πτυχιακη Εργασια

της

ΣΤΑΥΡΟΥΛΑΣ ΔΑΦΝΑ

Επιβλέπων: Γρηγόρης Καραγιώργος
Επίκουρος καθηγητής

Σπάρτη, Νοέμβριος 2015

Τεχνολογικο Εκπαιδευτικο Ιδρυμα Πελοποννησου
Σχολη Τεχνολογικων Εφαρμογων
Τμημα Μηχανικων Πληροφορικης Τ.Ε.

Αλγόριθμοι Ταξινόμησης Γραμμικού Χρόνου και Εκτέλεσης

Μελέτη και υλοποίηση

Πτυχιακη Εργασια

της

ΣΤΑΥΡΟΥΛΑΣ ΔΑΦΝΑ

Επιβλέπων: Γρηγόρης Καραγιώργος
Επίκουρος καθηγητής

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 11α Νοεμβρίου 2015.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Γρηγόρης Καραγιώργος	Ιωάννης Λιαπέρδος	Νικόλαος Κατσάκος-Μαυρομιχάλης
Επίκουρος καθηγητής	Καθηγητής Εφαρμογών	Καθηγητής

Σπάρτη, Νοέμβριος 2015

Τεχνολογικο Εκπαιδευτικο Ιδρυμα Πελοποννησου
Σχολη Τεχνολογικων Εφαρμογων
Τμημα Μηχανικων Πληροφορικης Τ.Ε.

Υπεύθυνη Δήλωση

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ενυπογράφως ότι είμαι αποκλειστικός συγγραφέας της παρούσας Πτυχιακής Εργασίας, για την ολοκλήρωση της οποίας κάθε βοήθεια είναι πλήρως αναγνωρισμένη και αναφέρεται λεπτομερώς στην εργασία αυτή. Έχω αναφέρει πλήρως και με σαφείς αναφορές, όλες τις πηγές χρήσης δεδομένων, απόψεων, θέσεων και προτάσεων, ιδεών και λεκτικών αναφορών, είτε κατά κυριολεξία είτε βάσει επιστημονικής παράφρασης. Αναλαμβάνω την προσωπική και ατομική ευθύνη ότι σε περίπτωση αποτυχίας στην υλοποίηση των ανωτέρω δηλωθέντων στοιχείων, είμαι υπόλογος έναντι λογοκλοπής, γεγονός που σημαίνει αποτυχία στην Πτυχιακή μου Εργασία και κατά συνέπεια αποτυχία απόκτησης του Τίτλου Σπουδών, πέραν των λοιπών συνεπειών του νόμου περί πνευματικών δικαιωμάτων. Δηλώνω, συνεπώς, ότι αυτή η Πτυχιακή Εργασία προετοιμάστηκε και ολοκληρώθηκε από εμένα προσωπικά και αποκλειστικά και ότι, αναλαμβάνω πλήρως όλες τις συνέπειες του νόμου στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής άλλης πνευματικής ιδιοκτησίας.

(Υπογραφή)

.....
Σταυρούλα Δαφνά

Περίληψη

Η παρούσα πτυχιακή εργασία έχει σκοπό να γνωρίσει τους αλγόριθμους ταξινόμησης γραμμικού χρόνου. Συγκεκριμένα οι αλγόριθμοι γραμμικού χρόνου που παρουσιάζονται είναι τρεις: η απαριθμιτική ταξινόμηση, η αριθμοτακτική και η ταξινόμηση με δοχεία οι οποίοι θα υλοποιηθούν με την γλώσσα προγραμματισμού C. Επίσης αναλύει τον χρόνο εκτέλεσης τους και γίνεται μια σύγκριση με τους αλγόριθμους ταξινόμησης των συγκριτικών αλγορίθμων οι οποίοι βασίζονται στην σύγκριση. Όσον αφορά για τους συγκριτικούς αλγορίθμους αποδεικνύεται ότι με την βοήθεια των δέντρων αποφάσεων και χρησιμοποιώντας το κάτω φράγμα απαιτούν $\Omega(n \log n)$ βήματα η οποία είναι η ιδανικη χρήση για αυτούς τους αλγόριθμους στην χειρότερη περίπτωση. Συμπερασματικά βλέπουμε ότι οι αλγόριθμοι γραμμικού χρόνου απαιτούν $O(n)$ βήματα και είναι η καλύτερη λύση μιας και είναι γρηγορότεροι σε χρόνο εκτέλεση από τους συγκριτικούς αλγορίθμους.

Λέξεις Κλειδιά

Αλγόριθμοι γραμμικού χρόνου, απαριθμιτική ταξινόμηση, αριθμοτακτική ταξινόμηση, ταξινόμηση με δοχεία, χρόνος εκτέλεσης

Abstract

This present thesis study aims to show the algorithms sort linear time. Specifically, the linear time algorithms presented here are these three: the counting sort, the radix sort and bucket sort with containers to be implemented with the programming language C. It also analyzes the execution time and compares with the sorting algorithms comparative algorithms which are based on comparison. With regard to the comparative algorithms it is proved that by means of decision trees and the use of the lower bound requires $\Omega(n \log n)$ steps which is ideal for using these algorithm in worst case scenarios. In conclusion we see that linear time algorithms require O(n) steps and they are the best solution as they are faster in execution time than by comparative algorithms.

Keywords

Algorithms linear time, counting sort, radix sort, bucket sort, execution time

στους γονείς μου Παναγιώτης και Παναγιώτα

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον καθηγητή κ. Γρηγόρη Καραγιώργο για την επίβλεψη αυτής της πτυχιακής εργασίας και για την ευκαιρία που μου έδωσε να την εκπονήσω. Επίσης ευχαριστώ ιδιαίτερα τον καθηγητή κ. Ιωάννη Λιαπέρδο για την πολύτιμη βοήθεια. Επίσης θα ήθελα να ευχαριστήσω τους συνφοιτητές μου στο διάστημα των φοιτιτικών μου χρόνων καθώς και την συνφοιτητριά μου Παρασκευή Ραφτοπούλου για την συνεργασία που είχαμε. Τέλος θα ήθελα να ευχαριστήσω τους γονείς μου και τον αδελφό μου για την καθοδήγηση και την ηθική συμπαράσταση που μου πρόσφεραν όλα αυτά τα χρόνια.

Σπάρτη, Νοέμβριος 2015

Σταυρούλα Δαφνά

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Εικόνων

Κατάλογος Πινάκων

Πρόλογος

Η πτυχιακή εργασία εκπονήθηκε στα πλαίσια του προγράμματος σπουδών στο τελευταίο εξάμηνο για την απόκτηση πτυχίου από το τμήμα Μηχανικών Πληροφορικής Τ.Ε. του ΤΕΙ ΠΕΛΟΠΟΝΝΗΣΟΥ (Έδρα: Σπάρτη), Τεχνολογικών Εφαρμογών. Στόχος της πτυχιακής εργασίας είναι να αναλυθούν και να υλοποιηθούν με την γλώσσα προγραμματισμού C οι αλγόριθμοι ταξινόμησης γραμμικού χρόνου καθώς και να συγκριθούν με τους αλγόριθμους ταξινόμησης συγκριτικών αλγορίθμων.

Κεφάλαιο 1

Εισαγωγή

1.1 Αντικείμενο της εργασίας

Ο όρος αλγόριθμος είναι ο θεμελιώδης στον τομέα της πληροφορικής. Λέγοντας αλγόριθμο εννοούμε την διαδικασία για την επίλυση κάποιων υπολογιστικών προβλημάτων. Χωρίς τους αλγόριθμους δεν θα μπορούσε να γίνει η υλοποίηση των προγραμμάτων, δεν θα υπήρχαν υπολογιστικά συστήματα και δεν θα μπορούσαν να αντιμετωπισθούν τα προβλήματα που υπάρχουν στην πληροφορική και σε άλλους τομείς όπως της ιατρικής, της τεχνικής νοημοσύνης κ.ά. Η σχεδίαση των αλγόριθμων βοήθησε πολλούς προγραμματιστές να τους υλοποιήσουν με την χρήση διάφορων γλωσσών προγραμματισμού όπως C, C++, Java κ.ά.

Η ανάλυση των αλγόριθμων παίζει σημαντικό ρόλο στην επιστήμη της πληροφορικής. Με την ανάλυση μπορεί ένας αλγόριθμος να υπολογίσει την πολυπλοκότητά του, δηλαδή να υπολογίσει τον χρόνο εκτέλεσης του αλγόριθμου καθώς επίσης και το χώρο μνήμης. Ένας αλγόριθμος μπορεί να αναλυθεί σε τρεις περιπτώσεις: την καλύτερη περίπτωση, την μέση περίπτωση και την χειρότερη περίπτωση. Για παράδειγμα ένας αλγόριθμος που έχει πολυπλόκοτηα $O(n \log n)$ θα βρίσκεται στην καλύτερη περίπτωση, ένας άλλος αλγόριθμος που θα έχει πολυπλοκότητα $O(n^2)$ θα βρίσκεται στη χειρότερη περίπτωση. Συνήθως η χειρότερη περίπτωση αφορά για το μέγιστο κόστος ενός αλγορίθμου ενώ η καλύτερη περίπτωση αφορά το ελάχιστο κόστος. Για κάθε αλγόριθμο θα πρέπει να μετρηθεί ο χρόνος εκτέλεσης και να βρεθεί σε ποια περίπτωση ανήκει.

Έχει διαπιστωθεί πως το πρόβλημα της ταξινόμησης είναι το θεμελιώδης πρόβλημα στους αλγορίθμους, για το λόγο αυτό σχεδιάστηκαν αλγόριθμοι ταξινόμησης που είναι αποδοτικοί. Λέγοντας ταξινόμηση εννοούμε τον τρόπο οργάνωσης και τακτοποίησης κάποιων στοιχείων έτσι ώστε να είναι στη σωστή σειρά και να είναι εύκολο η επεξεργασία τους, και η τροποποίηση. Ένα παράδειγμα που χρησιμοποιείται η ταξινόμηση στην καθημερινή μας ζωή είναι η ταξινόμηση των βιβλίων με βάση τις επιστήμες, στα αρχεία, στους τηλεφωνικούς καταλόγους κ.ά. Όσον αφορά για την πληροφορική η ταξινόμηση είναι η διαδικασία που τοποθετεί το κάθε στοιχείο σε ένα πίνακα στη σωστή θέση με την σειρά. Συνήθως τα στοιχεία μπορεί να είναι αριθμοί ή και αλφαριθμητικά. Για τον λόγο αυτό σχεδιάστηκαν αλγόριθμοι ταξινόμησης. Οι αλγόριθμοι ταξινόμησης χωρίζονται σε δύο κατηγορίες τους συγκριτικούς και τους μη συγκριτικούς όπου θα αναλυθούν στην πτυχιακή εργασία παρακάτω.

- **Συγκριτικοί**: είναι οι αλγόριθμοι που βασίζονται στην σύγκριση και στην αντιμετάθεση μεταξύ των στοιχείων. Θεωρούνται από τους πιο γνωστούς αλγόριθμους της ταξινόμη-

σης. Επίσης έχει αποδειχθεί ότι οι συγκριτικοί αλγόριθμοι χρησιμοποιούν ένα κάτω φράγμα με την βοήθεια των δέντρων αποφάσεων. Σε αυτή την κατηγορία ανήκουν οι αλγόριθμοι όπως ταξινόμηση με εισαγωγή, ταξινόμηση με επιλογή, ταξινόμηση με φυσαλίδα, γρήγορη ταξινόμηση, ταξινόμηση με συγχώνευση και η ταξινόμηση με σωρό.

- **Μη συγκριτικοί**: είναι οι αλγόριθμοι ταξινόμησης που βασίζονται σε διάφορες τεχνικές, όπως στον προσδιορίσμό των στοιχείων της σωστής θέσης και σειρά. Επίσης οι μη συγκριτικοί αλγόριθμοι είναι οι αλγόριθμοι ταξινόμησης γραμμικού χρόνου. Οι αλγόριθμοι αυτοί δεν βασίζονται σε συγκρίσεις. Ένα πλεονέκτημα που έχουν είναι ότι είναι αποδοτικοί από τους συγκριτικούς επειδή έχουν χρόνο εκτέλεσης $O(n)$ δηλαδή ο χρόνος είναι γραμμικού χρόνου και θεωρείται ιδανικός χρόνος για ένα αλγόριθμο που θέλει να ταξινομήσει τα στοιχεία του σε ένα πίνακα. Άρα οι αλγόριθμοι αυτής της κατηγορίας θεωρούνται ιδανικοί και καλύτεροι σε χρόνο. Οπότε είναι ταχύτεροι από τους συγκριτικούς. Οι πιο γνωστοί αλγόριθμοι ταξινόμησης σε γραμμικό χρόνο είναι η απαριθμιτική ταξινόμηση, η αριθμοτακτική ταξινόμηση, η ταξινόμηση με δοχεία.

Στην συγκεκριμένη πτυχιακή εργασία θα ασχοληθούμε με τους αλγόριθμους ταξινόμησης γραμμικού χρόνου. Στόχος της εργασία είναι να υλοποιηθούν οι αλγόριθμοι ταξινόμησης γραμμικού χρόνου με την γλώσσα προγραμματισμού C, να αναλυθεί η πολυπλοκότητά τους και να αποδειχθεί ότι είναι καλύτεροι και γρηγορότεροι σε χρόνο εκτέλεσης. Επίσης να συγκριθεί ο χρόνος εκτέλεσης με τους αλγόριθμους ταξινόμησης της κατηγόριας των συγκριτικών.

1.2 Διάρθωση της εργασίας

Η πτυχιακή εργασία έχει προγραμματιστεί ως εξής:

Στο **Κεφάλαιο 2** γίνεται μία εισαγωγή στους αλγόριθμους, ορίζεται η έννοια του αλγόριθμου, καθώς παρουσιάζεται η ανάλυση των αλγορίθμων.

Στο **Κεφάλαιο 3** ορίζεται ο ασυμπτωτικός συμβολισμός ο οποίος εφαρμόζεται για τον υπολογισμό χρόνου με μεγάλο μέγεθος εισόδου. Επίσης ορίζονται οι συμβολισμοί O, Θ, Ω.

Στο **Κεφάλαιο 4** παρουσιάζονται οι συγκριτικοί αλγόριθμοι ταξινόμησης. Για κάθε αλγόριθμο παρουσιάζεται με τη μορφή ενός ψευδοκώδικα, καθώς αναλύεται η πολυπλοκότητα του. Επίσης γίνεται η μελέτη του κάτω φράγματος για τους συγκριτικούς αλγόριθμους με την βοήθεια των δέντρων αποφάσεων. Με την χρήση των δέντρων αποφάσεων θα αποδειχθεί ότι για οποιοδήποτε συγκριτικό αλγόριθμο απαιτεί $\Omega(n \log n)$ συγρίσεις στην χειρότερη περίπτωση, η οποία θα είναι η ιδανική περίπτωση για αυτούς τους αλγορίθμους.

Στο **Κεφάλαιο 5** παρουσιάζονται οι αλγόριθμοι ταξινόμησης γραμμικού χρόνου καθώς και η πολυπλοκότητα τους.

Στο **Κεφάλαιο 6** συγκρίνονται οι δύο κατηγορίες των αλγόριθμων ταξινόμησης που μελετήθηκαν στην πτυχιακή ως προς τον χρόνο εκτέλεσης τους καθώς παρουσιάζεται η υλοποίηση των αλγόριθμων ταξινόμησης γραμμικού χρόνου.

Στο **Κεφάλαιο 7** παρουσιάζονται τα συμπεράσματα που προέκυψαν από την εργασία.

Κεφάλαιο 2

Εισαγωγή στους αλγορίθμους

Στο κεφάλαιο αυτό γίνεται μία εισαγωγή στους αλγορίθμους, στην έννοια του αλγόριθμου καθώς και στην ανάλυση τους.

2.1 Αλγόριθμοι

Οι αλγόριθμοι εμφανίζονται κυρίως στον τομέα της πληροφορικής, καθώς και σε άλλους τομείς όπως των μαθηματικών, της ιατρικής κ.ά. Ο πρώτος αλγόριθμος ανακαλύφθηκε από τον μαθηματικό Ευκλείδη για τον υπολογισμό του κοινού διαιρέτη δύων ακέραιων αριθμών. Υπάρχουν πολλές έννοιες που έχουν δοθεί για τον αλγόριθμο. Γενικά ο αλγόριθμος είναι μια διαδικασία που εφαρμόζεται συχνά για την εκτέλεση και την επίλυση κάποιων προβλημάτων. Πιο αναλυτικά παρακάτω δίνεται ο ορισμός του αλγόριθμού.

Ορισμός 2.1. *≪Με τον όρο αλγόριθμο εννοούμε μια πεπερασμένη σειρά βημάτων, αυστηρά καθορισμένων σε πεπερασμένο χρόνο για την εκτέλεση ενός προβλήματος≫ [2].*

Ένας αλγόριθμος έχει κάποια χαρακτιριστικά όπως:

- **Είσοδος**: Στην είσοδο περιλαμβάνονται όλα τα στοιχεία που είναι απαραίτητα για την επεξεργασία τους όσο και για την εκτέλεση του αλγόριθμου.
- **Έξοδος**: Κατά την εκτέλεση του αλγόριθμου παράγονται ένα ή περισσότερα αποτελέσματα. Το αποτέλεσμα είναι η έξοδος.
- **Πεπερασμένος**: Ο αλγόριθμος θα πρέπει να τερματίζει σε κάποιο πεπερασμένο αριθμό βημάτων μέχρι τον τερματισμό του.
- **Αποτελεσματικός**: Ο αλγόριθμος θα πρέπει στο τέλος η λύση να είναι αποτελεσματική.
- **Αναλυτικός**: Ο αλγόριθμος θα πρέπει να είναι αναλυτικά γραμμένος με τέτοιο τρόπο ώστε να είναι κατανοητός [9].

≪Για την υλοποίηση ενός αλγόριθμου θα πρέπει να ακολουθήσουν κάποια βήματα: Αρχικά πριν την σχεδίαση του αλγόριθμου θα πρέπει να διατυπωθεί το πρόβλημα. Στο

δεύτερο βήμα θα πρέπει να υπάρχει κατανόηση του πρόβληματος απο τον ίδιο τον προγραμματιστή έτσι ώστε να μπορεί να το υλοποιήσει τον αλγόριθμο. Στο τρίτο βήμα είναι η υλοποίηση του προβλήματος για την επίλυση του. Τέλος γίνεται ο έλεγχος του προβλήματος και μετά η παρουσίαση≫ [3].

Οι αλγόριθμοι εφαρμόστηκαν για την επίλυση ενός προβλήματος π.χ. στο πρόβλημα της ταξινόμησης, στο πρόβλημα της αναζήτησης, στον υπολογισμό παραγοντικού κ.ά. Μέχρι σήμερα έχουν σχεδιαστεί πολλοί αλγόριθμοι. Στη συνέχεια αναφέρονται κάποια είδη αλγορίθμων όπως: οι αλγόριθμοι ταξινόμησης, οι αναδρομικοί αλγόριθμοι, οι βέλτιστοι αλγόριθμοι, οι αλγόριθμοι διαίρει και βασίλευε. Εκτός από αυτά υπάρχουν και άλλα πάρα πολλά είδη αλγόριθμων που μπορεί να συναντήσει κάποιος αναγνώστης.

- **Αλγόριθμοι ταξινόμησης**: Οι αλγόριθμοι ταξινόμησης είναι μια τεχνική οι οποίοι σχεδιάστηκαν για την τακτοποίηση στοιχείων έτσι ώστε να είναι εύκολο η αναζήτηση τους και να είναι σε σειρά. Τα στοιχεία μπορεί να αποθηκευτούν σε πίνακα, σε δομή δεδομένων, σε σωρούς κ.ά. Οι αλγόριθμοι αυτοί χωρίζονται σε δύο κατηγορίες τους συγκριτικούς και τους μη συγκριτικούς οι οποίοι θα αναλυθούν στα επόμενα κεφάλαια.

- **Αναδρομικοί αλγόριθμοι**: Είναι οι αλγόριθμοι οι οποίοι όταν επιλύουν ένα πρόβλημα, καλούν τον εαυτό τους αναδρομικά μία ή περισσότερες φορές.

- **Βέλτιστοι αλγόριθμοι**: Βέλτιστοι χαρακτηρίζονται οι αλγόριθμοι στη περίπτωση όταν δεν μπορεί να υπάρχει αλγόριθμος με καλύτερη πολυπλοκότητα από κάποιον άλλον αλγόριθμο.

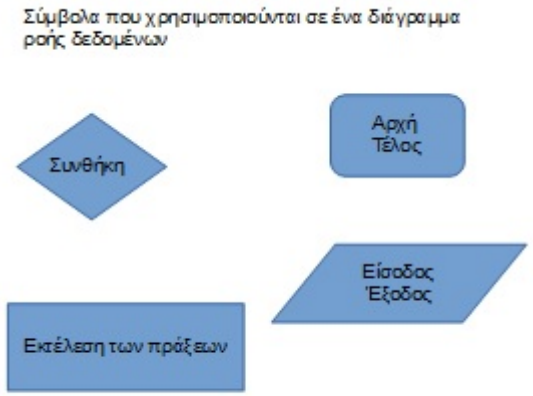
- **Διαίρει και βασίλευε**: Είναι οι αλγόριθμοι ταξινόμησης που διασπούν το πρόβλημα σε διάφορα υποπροβλήματα για την επίλυση τους χρησιμοποιώντας την αναδρομή. Τέτοιου είδους αλγόριθμων είναι η συγχωνευτική ταξινόμηση (merge sort) και η γρήγορη ταξινόμηση (quick sort).

.

Επίσης ένας αλγόριθμος μπορεί να αναπαρασταθεί με την μορφή ψευδόγλωσσας όπως απεικονίζεται στο παράδειγμα του αλγόριθμου για τον υπολογισμού αθροίσματος (βλ.2.1), είτε με την χρήση μιας γλώσσα προγραμματισμού όπως π.χ. C, C++ κ.ά., είτε με την χρήση ενός διαγράμματος ροής δεδομένων. Ένα διάγραμμα ροής δεδομένων αποτελείται από γεωμετρικά σχήματα όπως ρόμβος, έλλειψη, ορθογώνιο, παραλληλόγραμμα τα οποία συνδέονται με βέλη. Τα διαγράμματα αυτά είναι σχεδιασμένα κατάλληλα για την περιγραφή ενός αλγόριθμου. Όσον αφορά για τα σχήματα η έλλειψη συμβολίζει την αρχή και το τέλος ενός αλγόριθμου, το ορθογώνιο για την εκτέλεση των πράξεων, και το παραλληλόγραμμο δηλώνει για την είσοδο και την έξοδο των στοιχείων.

Αλγοριθμος 2.1: *Αλγόριθμος υπολογισμού αθροίσματος σε μορφή ψευδοκώδικας*

```
1: sum ← 0
2: i ← 1
3: while i ≤ n do
4:     sum ← sum + i
5:     i ← i + 1
6: end while
```



Σχήμα 2.1: *Σύμβολα διαγράμματος ροής δεδομένων*

2.2 Προβλήματα αλγόριθμων

Στην επιστήμη της πληροφορικής ο αλγόριθμος θεωρείται σημαντικός στον προγραμματισμό. Υπάρχουν πολλά προβλήματα στην πληροφορική τα οποία εντοπίζονται και τα οποία με τους αλγόριθμους έχουν λυθεί. Μερικά προβλήματα έχει αποδειχθεί ότι δεν υπάρχει κατάλληλος αλγόριθμος για την επίλυση του. Τα προβλήματα τα όποια έχουν αποδειχθεί ότι υπάρχει αλγόριθμος για αυτά και μπορούν να λυθούν λέγονται **επιλύσιμα προβλήματα** και ανήκουν στην κλάση P (Polynomial) ενώ τα προβλήματα τα οποία δεν έχει βρεθεί κάποιος αλγόριθμος για την επίλυση τους λέγονται **άλυτα προβλήματα** και ανήκουν στην κλάση NP (Non polynomial).

Ένα πρόβλημα που υπάρχει στο τομέα της πληροφορικής είναι το πρόβλημα της ταξινόμησης όπως αναφέρθηκε προηγουμένως. Η ταξινόμηση αποτελεί σημαντικό στοιχείο στον κλάδο της πληροφορικής, την οποία πολλές φορές την χρησιμοποιούν οι προγραμματιστές σε προγράμματα που θέλουν να υλοποιήσουν. Ένα παράδειγμα για το πρόβλημα της ταξινόμησης είναι ως εξής:

Παράδειγμα 2.1. *Ας υποθέσουμε ότι έχουμε ένα πίνακα μη ταξινομημένο με είσοδο [2,3,4,1,5,6], στην έξοδο θα πρέπει να προκύψει το αποτέλεσμα [1,2,3,4,5,6] έτσι ώστε το μικρότερο στοιχείο του πίνακα να είναι στην πρώτη θέση και το μεγαλύτερο στην τελευταία θέση. Αν προκύψει το αποτέλεσμα τότε ο πίνακας θα είναι ταξινομημένος.*

Σε αυτό το σημείο έχουν σχεδιαστεί αλγόριθμοι ταξινόμησης για την επίλυση του προ-

βλήματος της ταξινόμησης, οι οποίοι θα αναφερθούν στα παρακάτω κεφάλαια.

2.3 Η ανάλυση των αλγορίθμων

Η ανάλυση των αλγορίθμων είναι σημαντική για τον χρόνο εκτέλεσης που απαιτεί ένας αλγόριθμος για την επίλυση κάποιου προβλήματος. Όταν υλοποιηθεί σε μια γλώσσα προγραμματισμού ένας αλγόριθμος, θα πρέπει να υπολογισθεί η πολυπλοκότητά του, δηλαδή να υπολογισθεί σε πόσο χρόνο θα εκτελεσθεί ο αλγόριθμος μετά από κάποια πεπερασμένα βήματα. Οι πράξεις για τον υπολογισμό του χρόνου εκτέλεσης είναι οι συγκρίσεις μεταξύ των στοιχείων, ο πολλαπλασιασμός, η διαίρεση. Επίσης το πόσο αποδοτικός είναι ένας αλγόριθμος εξαρτάται από το πόσο γρήγορα εκτελείται. Έτσι η αποδοτικότητα είναι ένα μέρος της ανάλυσης.

Επίσης το πόσο είναι ένας αλγόριθμος βέλτιστος παίζει σημαντικό ρόλο στην ανάλυση των αλγορίθμων για το πρόβλημα. Αντίθετα αν ο αλγόριθμος δεν είναι βέλτιστος, θα πρέπει να εξεταστεί κατά πόσο μπορεί να γίνει βέλτιστος, ή το πολύ μπορεί να χρησιμοποιηθεί ένας άλλος αλγόριθμος που θα είναι αποδοτικός.

Γενικά ο χρόνος εκτέλεσης υπάρχει περίπτωση να μην είναι σταθερός αλλά να μεταβάλλεται. Αυτό εξαρτάται από τον υπολογιστή, από το χώρο μνήμης, από την γλώσσα προγραμματισμού και την κλάση που ανήκει ένας αλγόριθμος. Για το λόγο αυτό ως προς την πολυπλοκότητα χρησιμοποιείται ένας συμβολισμός που ονομάζεται Ο (όμικρον μεγάλο κεφαλαίο). Με βάση τον συμβολισμό αυτόν έχουν ορισθεί κάποιες κατηγορίες ανάλογα με τον χρόνο εκτέλεσης.

Λογαριθμικού χρόνου $O(\log n)$: Σε αυτή την κατηγορία ανήκουν οι αλγόριθμοι που έχουν χρόνο εκτέλεσης $O(\log n)$ και ο χρόνος εκτέλεσης είναι λογαριθμικός με βάση το δύο. Για παράδειγμα αν έχουμε ένα πλήθος στοιχείων n σε ένα πίνακα $A[n]$, τότε ο χρόνος εκτέλεσης θα είναι $\log n$. Οι αλγόριθμοι αυτής της κατηγόριας είναι ταχύτεροι και για μεγάλο μέγεθος στοιχείων μπορούν να εκτελεστούν σε πολύ μικρό χρόνο. Μπορούμε να πούμε λογαριθμικού χρόνου είναι η ταξινόμηση με συγχώνευση (merge sort) και η γρήγορη ταξινόμηση (quick sort).

Γραμμικού χρόνου $O(n)$: Σε αυτή την κατηγορία ανήκουν οι αλγόριθμοι που έχουν χρόνο εκτέλεσης $O(n)$. Έχει αποδειχθεί ότι είναι οι καλύτεροι αλγόριθμοι και γρηγορότεροι σε χρόνο εκτέλεσης καθώς μπορούμε να τους πούμε βέλτιστους αλγόριθμους. Για παράδειγμα αν ένας πίνακας παράγει n στοιχεία στην είσοδο τότε ο χρόνος εκτέλεσης θα είναι $O(n)$. Όταν ένας αλγόριθμος εκτελείται σε χρόνο γραμμικό τότε λέμε ότι ανήκει στην τάξη $O(n)$.

Σταθερού χρόνου $O(1)$: Στην κατηγορία αυτή ανήκουν οι αλγόριθμοι όπου ο χρόνος εκτέλεσης είναι πάντα σταθερός και δεν μεταβάλλεται.

Πολυωνυμικού χρόνου $O(n^2)$: Στην κατηγορία αυτή ανήκουν οι αλγόριθμοι όπου ο χρόνος εκτέλεσης τους είναι $O(n^2)$ και είναι τετραγωνικός και μπορεί να χρησιμοποιηθεί μόνο όταν το μέγεθος πίνακα είναι μικρό.

Εκθετικού χρόνου: Στην κατηγορία αυτή ανήκουν οι αλγόριθμοι όπου ο χρόνος εκτέλεσης είναι εκθετικός και όταν το μέγεθος n διπλασιάζεται ο χρόνος εκτέλεσης τετραγωνίζεται.

Γραμμολογαριθμικού χρόνου $O(n \log n)$: Στην κατηγορία αυτή ανήκουν οι αλγόριθμοι όπου ο χρόνος εκτέλεσης είναι $O(n \log n)$ και συνήθως ανήκουν οι αλγόριθμοι που για την

επίλυση τους διαιρούν το πρόβλημα σε μικρότερα υποπροβλήματα τα οποία επιλύονται και στη συνέχεια συνδυάζουν τις λύσεις τους έτσι ώστε να προκύψει το αποτέλεσμα. Τέτοιου είδους αλγόριθμων ανήκουν η γρήγορη ταξινόμηση και η ταξινόμηση με συγχώνευση [9].

Κεφάλαιο 3

Ασυμπτωτικός συμβολισμός

Όπως είδαμε στο προηγούμενο κεφάλαιο για να υπολογισθεί ο χρόνος εκτέλεσης ενός αλγόριθμου θα πρέπει να υπολογισθούν τα βήματα ενός αλγορίθμου. Υπάρχουν αλγόριθμοι όπου το μέγεθος στην είσοδο είναι αρκετά μεγάλο δηλαδή όταν το $n \to \infty$. Στην περίπτωση αυτή μιλάμε για ασυμπτωτικό συμβολισμό. Ο ασυμπτωτικός συμβολισμός χρησιμοποιείται σε περιπτώσεις όπου το μέγεθος στην είσοδο είναι αρκετά πολύ μεγάλο. Σε αυτό το κεφάλαιο θα ορισθούν οι ασυμπτωτικοί συμβολισμοί Θ, O, Ω.

3.1 Συμβολισμος Θ

Ο συμβολισμός Θ ορίζει ένα άνω φράγμα και ένα κάτω φράγμα. Έτσι λέμε ότι ο συμβολισμός Θ είναι ασυμπτωτικά φραγμένος άνω και κάτω. Για την κατανόηση θα ορίσουμε τον συμβολισμό Θ.

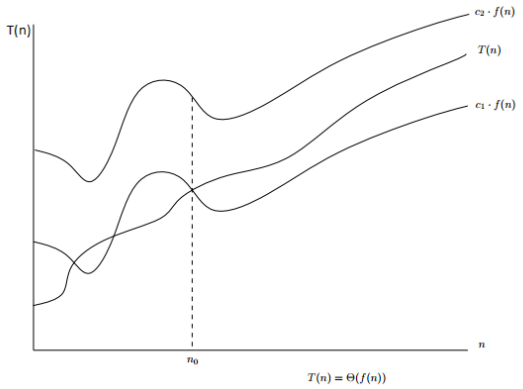
Ορισμός 3.2. $\Theta(g(n)) = \{f(n)$: *υπάρχουν θετικές σταθερές* c_1, c_2 *και* n_0 *τέτοιες ώστε* $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ *για κάθε* $n \geq n_0$ $\}$.

Σύμφωνα με τον παραπάνω ορισμό προκύπτει ότι για κάποια δεδομένη $g(n)$ συνάρτηση, το $\Theta(g(n))$ είναι το σύνολο των συναρτήσεων. Άρα η συνάρτηση $f(n) \in \Theta(g(n))$ ή $f(n) = \Theta(g(n))$, αν υπάρχουν σταθερές c_1, c_2, τέτοιες ώστε η $f(n)$ να κινείται μεταξύ των $c_1 g(n)$ και $c_2 g(n)$, για μεγάλες τιμές του n. Παρακάτω στην εικόνα 3.1 βλέπουμε την γραφική αναπαράσταση της $f(n) = \Theta(g(n))$. Από την γραφική αναπαράσταση της $f(n) = \Theta(g(n))$ για τιμές μεγαλύτερες του n παρατηρείται ότι η $c_1 g(n)$ κινείται πάνω από την $f(n)$ και η $c_2 g(n)$ κινήται κάτω από την $f(n)$.

Επίσης η συνάρτηση $f(n) = \Theta(g(n))$ θα πρέπει να είναι συμπτωτικά μη αρνητικά, δηλαδή η $f(n)$ να περιλαμβάνει τιμή θετική και όχι αρνητική από κάποια τιμή του n. Αν περιλαμβάνει αρνητικές τιμές τότε το σύνολο $\Theta(g(n))$ θα είναι κενό [8].

3.2 Συμβολισμος Ο

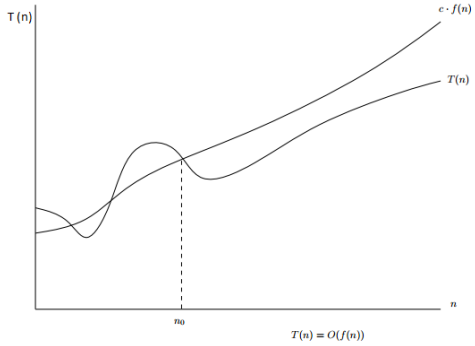
Όπως είδαμε στην προηγούμενη ενότητα ο συμβολισμός Θ αποτελεί άνω και κάτω φράγμα. Ο συμβολισμός Ο αποτελεί ένα άνω φράγμα, και χρησιμοποιείται για τον υπολογισμό πολυπλοκότητας στην χειρότερη περίπτωση. Πιο συγκεκριμένα θα ορίσουμε τον συμβολισμό Ο. Για μια συνάρτηση $g(n)$ ο συμβολισμός $O(g(n))$ διαβάζεται όμικρον κεφαλαίο του $g(n)$ και το $g(n)$ είναι ένα σύνολο συναρτήσεων.



Εικόνα 3.1: $f(n) = \Theta(g(n))$ [15]

Ορισμός 3.3. *$O(g(n))$=$\{f(n)$): υπάρχουν θετικές σταθερές c και n_0 τέτοιες ώστε $0 \leq f(n) \leq cg(n)$ για κάθε $n \leq n_0$.*

Η συνάρτηση $f(n) = O(g(n))$ δηλώνει ότι η $f(n)$ ανήκει στο σύνολο $O(g(n))$. Πιο συγκεκριμένα σύμφωνα με τον ορισμό προκύπτει ότι για κάθε $n > n_0$ υπάρχουν σταθερές c και n_0 τέτοιες ώστε να ισχύει $0 \leq f(n) \leq cg(n)$. Παρακάτω στην εικόνα 3.2 βλέπουμε την γραφική αναπαράσταση της $f(n) = O(g(n))$.



Εικόνα 3.2: $f(n) = O(g(n))$ [15]

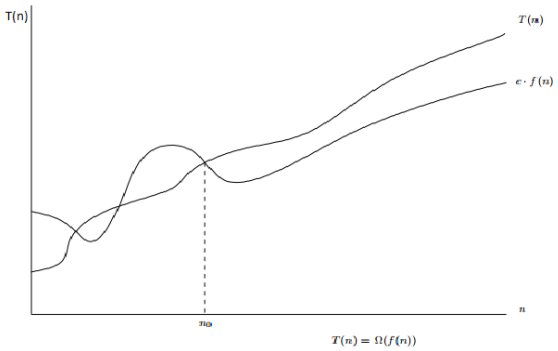
Από την γραφική αναπαράσταση παρατηρείται πως η τιμή της $f(n)$ κινείται κάτω από την $cg(n)$ για μεγάλες τιμές του n_0. Αρα ο συμβολισμός O ορίζει ένα φράγμα άνω στην χειρότερη περίπτωση. Επίσης ο συμβολισμός O μπορεί να χρησιμοποιηθεί για να περιγράψει τον χρόνο εκτέλεσης ενός αλγόριθμου [8].

3.3 Συμβολισμος Ω

Ο ασυμπτωτικός συμβολισμός Ω ορίζει ένα κάτω φράγμα. Αυτό σημαίνει ότι για κάθε $n \leq n_0$, υπάρχουν θετικές σταθερές c και n_0 τέτοιες ώστε $0 \leq cg(n) \leq f(n)$. Ο συμβολισμός $f(n) = \Omega(g(n))$ διαβάζεται ωμέγα κεφαλαίο του $g(n)$. Πιο συγκεκριμένα ας ορίσουμε τον συμβολισμό του Ω.

Ορισμός 3.4. *$\Omega(g(n))$=$\{f(n)$ υπάρχουν θετικές σταθερές c και n_0 τέτοιες ώστε $0 \leq cg(n) \leq f(n)$ για κάθε $n \leq n_0\}$.*

Παρακάτω στην εικόνα 3.3 βλέπουμε την γραφική αναπαράστασης της συνάρτησης $f(n) = \Omega(g(n))$ [8].



Εικόνα 3.3: $f(n) = \Omega(g(n))$ *[15]*

Κεφάλαιο 4

Αλγόριθμοι ταξινόμησης

Στο κεφάλαιο αυτό παρουσιάζονται οι αλγόριθμοι ταξινόμησης που ανήκουν στην κατηγορία των συγκριτικών αλγορίθμων ταξινόμησης (sorting algorithms comparative), με κάποια παραδείγματα καθώς μελετάται και η ανάλυση τους η οποία βασίζεται στις συγκρίσεις μεταξύ των στοιχείων. Μετρώντας των αριθμό των συγκρίσεων των στοιχείων και των μεταθέσεων προκύπτει ο χρόνος εκτέλεσης. Στο τέλος του κεφαλαίου θα αποδειχθεί ότι με την χρήση των δέντρων αποφάσεων για τους συγκριτικούς αλγορίθμους υπάρχει ένα κάτω φράγμα το οποίο για να ταξινομηθούν n αριθμοί, στη χειρότερη περίπτωση απαιτεί $\Omega(n \log n)$ αριθμό συγκρίσεων.

4.1 Ενθετική ταξινόμηση

Η ενθετική ταξινόμηση ή αλλιώς ταξινόμηση με εισαγωγή (insertion sort) ανήκει στη κατηγορία των συγκριτικών αλγορίθμων και η υλοποίηση θεωρείται απλή. Επίσης είναι γρήγορος ο αλγόριθμος όταν το μέγεθος της εισόδου είναι μικρό όπου ο χρόνος εκτέλεσης στην καλύτερη περίπτωση απαιτεί $O(n)$ βήματα εφόσον είναι ταξινομημένος. Παρακάτω βλέπουμε τον αλγόριθμο της ενθετικής ταξινόμησης (βλπ. 4.1).

Ο αλγόριθμος αυτός χρησιμοποιείται για την ταξινόμηση των στοιχείων που βρίσκονται σε έναν πίνακα $A[n]$, όπου n είναι το μέγεθος του πίνακα, και σαρώνοντας τον πίνακα από τα αριστερά προς τα δεξιά συγκρίνονται όλα τα στοιχεία από την αρχή του πίνακα. Η ταξινόμηση αυτή κάνει συνεχώς συγκρίσεις μεταξύ των στοιχείων από τα αριστερά προς τα δεξιά. Πιο αναλυτικά παίρνουμε το στοιχείο από τα μη ταξινομημένα και βρίσκουμε τη σωστή θέση του, και αφού βρεθεί η σωστή θέση εισάγεται στη θέση του, και τα υπόλοιπα στοιχεία μετακινούνται μια θέση προς τα δεξιά. Η διαδικασία θα συνεχιστεί με το ίδιο τρόπο και θα τερματίσει μέχρι το τελευταίο στοιχείο που θα είναι στην σωστή θέση. Η διαδικασία αυτή μοιάζει σαν την τακτοποίηση των φύλλων μιας τράπουλας, παίρνοντας τα δέκα φύλλα τοποθετώντας με την σειρά από τα αριστερά προς τα δεξιά [8].

Παρακάτω στον πίνακα 4.1 απεικονίζεται η ταξινόμηση με εισαγωγή ενός πίνακα πέντε στοιχείων A=[3 5 2 1 4].

Αρχικά η ταξινόμηση ξεκινάει από το πρώτο στοιχείο που βρίσκεται στην πρώτη θέση που είναι το 3, και συγκρίνεται με το δεύτερο στοιχείο στην δεύτερη θέση που είναι το 5. Παρατηρείται ότι δεν θα γίνει κάποια ανταλλαγή των στοιχείων αφού το 3 είναι μικρότερο από το 5, οπότε παραμένει στην θέση του.

Αλγοριθμος 4.1: *Αλγόριθμος ενθετικής ταξινόμησης [8]*

```
Insertion_sort(A,n}
for(int j=2; j<n; j++)
{
 κλειδί = A[j]
 \\Εισάγουμε το A[j]
 \\στη ταξινομημένη ακολουθία
 i=j-1;
 while (i>0; && A[i]>κλειδί)
  {
    A[i+1]=A[i];
     i=i-1;
  }
A[i+1]=κλειδί
}
```

3	5	2	1	4
3	**5**	2	1	4
3	5	**2**	1	4
2	3	5	**1**	4
1	2	3	5	**4**
1	**2**	**3**	**4**	**5**

Πίνακας 4.1: *Ταξινόμηση με εισαγωγή*

Συνεχίζοντας με το δεύτερο στοιχείο το 5 που βρίσκεται στην δεύτερη θέση του πίνακα θα παραμείνει στη θέση του, αφού το 3 είναι μικρότερο από το 5. Στην συνέχεια προχωρώντας στην τρίτη θέση που είναι το στοιχείο 2 θα συγκριθεί με το στοιχείο 3 που βρίσκεται στην πρώτη θέση, και το στοιχείο 2 θα εισαχθεί στη πρώτη θέση αφού είναι μικρότερο από το 3, και το στοιχείο 3 θα μετακινηθεί μια θέση στα δεξιά του πίνακα δηλαδή στην δεύτερη θέση καθώς και τα υπόλοιπα στοιχεία μία θέση δεξιά, και ο πίνακας θα είναι ως εξής: A=[2 3 5 1 4].

Πηγαίνοντας στη τέταρτη θέση που βρίσκεται το στοιχείο 1 θα μετακινηθεί στην πρώτη θέση μιας και είναι το μικρότερο στοιχείο του πίνακα, και το στοιχείο 2 που βρίσκεται στην πρώτη θέση θα μετακινηθεί μία θέση δεξιά δηλαδή στην δεύτερη θέση μαζί και τα υπόλοιπα στοιχεία θα μετακινηθούν στα δεξιά και έτσι ο πίνακα θα είναι ο A=[1 2 3 5 4].

Τέλος με το στοιχείο 4 που βρίσκεται στην τελευταία θέση θα συγκριθεί με το στοιχείο 5 που βρίσκεται στην τέταρτη θέση και θα μετακινηθεί στην τέταρτη θέση μιας και το 4 είναι μικρότερο από το πίνακα. Στο σημείο αυτό η ταξινόμηση τελειώνει και ο πίνακας είναι ταξινομημένος με την σειρά. Οπότε ο πίνακας είναι A=[1 2 3 4 5].

ΑΝΑΛΥΣΗ ΠΟΛΥΠΛΟΚΟΤΗΤΑΣ: Ο χρόνος εκτέλεσης της ενθετικής ταξινόμησης εξαρτάται από το πλήθος των στοιχείων που υπάρχουν στη είσοδο ενός πίνακα. Όσο περισσότερα στοιχεία υπάρχουν στον πίνακα τόσο περισσότερο χρόνο χρειάζεται για να εκτελεστεί ο χρόνος εκτέλεσης. Αντίθετα όσο λιγότερα στοιχεία υπάρχουν στον πίνακα τόσο πιο γρήγορα θα εκτελεστεί. Αν είχαμε στη είσοδο τέσσερα στοιχεία θα χρειαζόταν λιγότερο χρόνο να εκτελεστεί γρηγορότερα, από ότι αν είχαμε πεντακόσια στοιχεία στην είσοδο. Αυτό σημαίνει ότι ο χρόνος εκτέλεσης γίνεται αργός καθώς αυξάνεται το μέγεθος του πίνακα. Για τον υπολογισμό του χρόνου εκτέλεσης υπολογίζονται τα βήματα των συγκρίσεων και των επαναλήψεων.

Έχει αποδειχθεί ότι ο αλγόριθμος της ταξινόμησης με εισαγωγή (insertion sort) στην καλύτερη περίπτωση κάνει $O(n)$ βήματα έχοντας χρόνο γραμμικό. Στην περίπτωση αυτή δεν υπάρχουν συγκρίσεις και μετακινήσεις των στοιχείων και ο πίνακας είναι ταξινομημένος. Αντίθετα ο αλγόριθμος της ταξινόμησης με εισαγωγής στην χειρότερη περίπτωση κάνει $O(n^2)$ βήματα όταν στην είσοδο ο πίνακας είναι ταξινομημένος αντίστροφα δηλαδή σε φθίνουσα σειρά. Κάθε στοιχείο συγκρίνεται και μετακινείται στα αριστερά για να ταξινομηθεί. Στην συνέχεια θα αποδείξουμε ότι στη καλύτερη περίπτωση κάνει $O(n)$ βήματα και στην χειρότερη $O(n^2)$ βήματα. Για να υπολογίσουμε τον χρόνο εκτέλεσης της ενθετικής ταξινόμησης ας υποθέσουμε ότι $T(n)$ είναι ο χρόνος εκτέλεσης. Άρα για τον υπολογισμό του χρόνου εκτέλεσης πολλαπλασιάζεται το κάθε στοιχείο της στήλης κόστος με το αντίστοιχο στοιχείο του κόστους και αθροίζεται. Επομένως

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1)[8]$$

Καλύτερη περίπτωση: Ο αλγόριθμος απαιτεί $O(n)$ βήματα.

Απόδειξη: ≪ Όπως αναφέρεται στην καλύτερη περίπτωση κάνει $O(n)$ βήματα όταν ο πίνακας είναι ήδη ταξινομημένος. Στη περίπτωση αυτή ισχύει ότι $A[i] \leq key$. Η συνθήκη αυτή ισχύει όταν στοιχείο βρίσκεται στην σωστή θέση και σειρά οπότε το στοιχείο μένει στην θέση του και δεν μετακινείται και ο χρόνος δεν μεταβάλλεται δηλαδή παραμένει σταθερός. Άρα για $j = 1, 2, 3, 4, 5, \ldots . n$ προκύπτει ότι $t_j = 1$ δηλαδή ο χρόνος είναι σταθερός. Άρα ο χρόνος εκτέλεσης στη καλύτερη περίπτωση θα είναι: $T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) = (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_6)[8]$ ≫.

Χειρότερη περίπτωση: Στην χειρότερη περίπτωση απαιτεί $O(n^2)$ βήματα όταν είναι αντίστροφα ταξινομημένος καθώς ο πίνακας είναι σε φθίνουσα σειρά.

Απόδειξη: ≪Σε αυτή την περίπτωση για την ταξινόμηση θα πρέπει να συγκριθεί το κάθε στοιχείο $A[j]$ με όλα τα υπόλοιπα στοιχεία της ταξινομημένης υποσυστοιχίας A[1..θ-1], οπότε για όλα τα $j = 1, 2, 3 \ldots n$ θα ισχύει ότι $t_j = j$.Όμως

$$\sum_{j=2}^{n} j = \frac{n(n-1)}{2} - 1 = O(n^2)$$

και

$$\sum_{j=2}^{n} (j-1) = \frac{n(n-1)}{2} = O(n^2)$$

.

Οπότε ο χρόνος εκτέλεσης στη χειρότερη περίπτωση θα είναι:

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1) = c_1 n + c_2(n-1) + c_4(n-1) + c_5(\frac{n(n-1)}{2} - 1) + c_6(\frac{n(n-1)}{2}) + c_7(\frac{n(n-1)}{2}) + c_8(n-1) = (\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2})n^2 + (c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8)n - (c_2 + c_4 + c_5 + c_8) = O(n^2)[8] \gg .$$

4.2 Ταξινόμηση με επιλογή

Η ταξινόμηση με επιλογή (selection sort) ανήκει στην κατηγορία των συγκριτικών αλγορίθμων, κάνοντας συνεχώς συγκρίσεις μεταξύ των στοιχείων μέχρι να βρεθεί το μικρότερο στοιχείο στον πίνακα. Σε αυτή την περίπτωση ψάχουμε το μικρότερο στοιχείο από όλο τον πίνακα και αφού βρεθεί ανταλλάζεται με το πρώτο στοιχείο που βρίσκεται στη 1η θέση δηλαδή στη θέση A[0]. Στη συνέχεια με την ίδια διαδικασία βρίσκουμε το δεύτερο μικρότερο στοιχείο με την μικρότερη τιμή και ανταλλάζεται με το στοιχείο που βρίσκεται στη δεύτερη θέση του πίνακα δηλαδή A[1]. Με τον ίδιο τρόπο η διαδικασία συνεχίζεται. Η ταξινόμηση θα τελειώσει όταν το το μεγολύτερο βρίσκεται στην τελευταία θέση, και ο πίνακας θα είναι ταξινομημένος από το μικρότερο στοιχείο στη 1η θέση και στην τελευταία θέση το μεγαλύτερο. Στη περίπτωση αυτή όπου όλα τα τα στοιχεία θα έχουν βρεθεί στη σωστή σειρά τότε θα λέμε ότι ο πίνακας είναι ταξινομημένος. Στον πίνακα 4.2 απεικονίζεται ένα παράδειγμα της ταξινόμηση με επιλογή ενός πίνακα με 4 στοιχεία.

	11	6	**2**	7
1ο βήμα	2	**6**	**11**	7
2ο βήμα	2	6	**11**	**7**
3ο βήμα	2	6	**7**	**11**
Ταξινομημένος πίνακας	**2**	**6**	**7**	**11**

Πίνακας 4.2: *Ταξινόμηση με επιλογή*

- 1_0 **Βήμα**: Αρχικά βρίσκουμε το πρώτο μικρότερο στοιχείο του πίνακα που είναι το 2 που βρίσκεται στη 3η θέση και συγκρίνεται με το πρώτο στοιχείο που είναι το 11. Επειδή το 2 είναι μικρότερο από το 11, τότε θα γίνει ανταλλαγή των δύο αυτών στοιχείων και το 2 θα μεταφερθεί στην πρώτη θέση και το 11 στη θέση που βρισκόταν το 2. Έτσι ο πίνακας θα θα προκύψει ωε εξής: A[2 6 11 7].

- 2_0 **Βήμα**: Στην συνέχεια το 6 είναι το δεύτερο μικρότερο στοιχείο. Το στοιχείο 6 βρίσκεται στην δεύτερη θέση και αφού είναι το δεύτερο μικρότερο θα παραμείνει στην θέση του. Άρα ο πίνακας θα προκύψει ως εξής: A[2 6 11 7].

- 3_0 **Βήμα**: Συνεχίζοντας με το τρίτο μικρότερο στοιχείο που είναι το 7 και βρίσκεται στην τέταρτη θέση συγκρίνεται με το στοιχείο που βρίσκεται στην τρίτη θέση που είναι το 11. Επειδή το 7 είναι μικρότερο από το στοιχείο 11 τότε θα γίνει ανταλλαγή των στοιχείων και το 7 θα μεταφερθεί στην τρίτη θέση. Άρα ο πίνακας θα προκύψει ως εξής: A[2 6 7 11].

Παρατηρείται ότι ο πίνακας είναι ήδη ταξινομημένος και ο αλγόριθμος τερματίζει. Οπότε ο πίνακας είναι ταξινομημένος. Παρακάτω βλέπουμε τον αλγόριθμο της ταξινόμησης με επιλογής (βλπ. 4.2).

ΠΟΛΥΠΛΟΚΟΤΗΤΑ: Για τον υπολογισμό του χρόνου εκτέλεση της ταξινόμησης με επιλογή υπολογίζονται οι συγκρίσεις μέχρι να βρεθεί το πρώτο μικρότερο στοιχείο στο πίνακα. Άρα επιλέγοντας το πρώτο μικρότερο στοιχείο ο αλγόριθμος θα κάνει $n-1$ βήματα μέχρι

Αλγοριθμος 4.2: *Αλγόριθμος ταξινόμησης με επιλογής [8]*

```
Selection_sort(A,n)
for (i=0; i < n-1; i++)
{
 min=a[i];
 minj=i;
   for (j=i+1; j<n; j++)
     if (a[i] < min)
       {
       min= a[j];
       min=j;
       }
       a[min]=a[i];
       a[i]=min;
       }
       }
```

να βρεθεί και να αντικατασταθεί με το πρώτο στοιχείο που βρίσκεται στην πρώτη θέση, στην συνέχεια βρίσκοντας το δεύτερο μικρότερο στοιχείο ο αλγόριθμος θα κάνει $n-2$ βήματα κ.ο.κ. Άρα σύμφωνα με αυτά που προκύπτουν ο αλγόριθμος κάνει συνολικό χρόνο:

$$(n-1)+(n-2)+(n-3)+....+2+1=\sum_{i=2}^{n} n-i=\frac{n(n-1)}{2}\in O(n^2)$$

Άρα ο αλγόριθμος με επιλογή ανήκει στους αργούς αλγορίθμους με πολυπλοκότητα $O(n^2)$ [9].

Αλγοριθμος 4.3: *Αλγόριθμος ταξινόμησης με φυσαλίδα [9]*

```
Bubble_sort(A,n)
for (i=1; i<n; i++)
{
 for (j=n-1; j<1; j++)
 {
 if (A[j] < A[j-1])
  {
  swap A[j-1],A[j]
}
}
}
```

4.3 Ταξινόμηση φυσαλίδας

Η ταξινόμηση φυσαλίδας (bubble sort) είναι μια μέθοδος αντιμετάθεσης και ανήκει στους συγκριτικούς αλγόριθμους μιας και κάνει συνεχώς συγκρίσεις με τα γειτονικά στοιχεία ανά δύο στοιχεία του πίνακα, καθώς μπορεί να κάνει αντιμεταθέσεις με τα στοιχεία τα οποία δεν βρίσκονται στην σωστή σειρά. Επίσης μπορεί να κάνει πολλές επαναλήψεις μέχρι το τέλος της ταξινόμησης. Η μέθοδός της αντιμετάθεσης θα τερματίσει όταν τα στοιχεία θα βρίσκονται στην σωστή σειρά. Τότε λέμε ότι ο πίνακας είναι ταξινομημένος. Παρακάτω βλέπουμε τον αλγόριθμο της ταξινόμησης με φυσαλίδας (βλπ. 4.3).

Η διαδικασία έναρξης του αλγορίθμου μπορεί να ξεκινήσει από την αρχή του πίνακα είτε από το τέλος. Όταν η ταξινόμηση ξεκινήσει από την αρχή του πίνακα, τότε στο τέλος ο πίνακας θα πρέπει να είναι ταξινομημένος από τα αριστερά προς τα δεξιά δηλαδή σε αύξουσα σειρά, διαφορετικά αν ξεκινήσει από το τέλος τότε ο πίνακας θα είναι ταξινομημένος από τα δεξιά προς τα αριστερά σε φθίνουσα σειρά. Στη συνέχεια περιγράφεται η μέθοδος του αλγόριθμου ταξινόμησης φυσαλίδας (bubble sort):

- Αρχικά ξεκινώντας από την αρχή του πίνακα συγκρίνεται το πρώτο με το δεύτερο στοιχείο.

- Αν το πρώτο στοιχείο είναι μεγαλύτερο από το δεύτερο τότε ανταλλάζεται το δεύτερο με το πρώτο στοιχείο, και το δεύτερο στοιχείο μετακινείται στην πρώτη θέση.

- Στη συνέχεια συγκρίνεται το δεύτερο με το τρίτο στοιχείο και επαναλαμβάνεται η διαδικασία ξεκινώντας από το πρώτο στοιχείο συγκρίνοντας όλα τα στοιχεία του τελευταίου.

- Επαναλαμβάνεται η διαδικασία εφόσον υπάρξει έστω μια ανταλλαγή. Η ταξινόμηση θα τερματίσει όταν δεν υπάρχουν στοιχεία για ανταλλαγή και ο πίνακας τότε θα είναι ταξινομημένος.

Στον πίνακα 4.3 βλέπουμε ένα παράδειγμα της ταξινόμηση με φυσαλίδας ενός πίνακα με είσοδο τεσσάρων στοιχείων A[4, 2, 3, 1].

4	**2**	3	1	1ο πέρασμα
2	**4**	**3**	1	
2	3	**4**	**1**	
2	**3**	1	4	2ο πέρασμα
2	**3**	**1**	4	
2	1	**3**	**4**	
2	**1**	3	4	3ο πέρασμα
1	**2**	**3**	4	
1	2	**3**	**4**	
1	2	3	4	ταξινομημένος πίνακς

Πίνακας 4.3: *Ταξινόμηση με φυσαλίδα*

Αρχικά η ταξινόμηση ξεκινάει από την αρχή του πίνακα. Οπότε στο τέλος της ταξινόμησης ο πίνακας θα πρέπει να είναι ταξινομημένος σε αύξουσα σειρά με την προυπόθεση ότι τα στοιχεία θα είναι διατεταγμένα από το μικρότερο στο μεγαλύτερο στοιχείο. Δηλαδή η έξοδος θα πρέπει να προκύψει σε αυτή τη μορφή A[1 2 3 4]. Αν προκύψει σε αυτή την μορφή τότε ο αγόριθμος δουλεύει σωστά.

- **Πέρασμα 1ο**: Πρώτα συγκρίνονται τα στοιχεία 4 και 2 που βρίσκονται στις θέσεις 1 και 2. Επειδή το 4 είναι μεγαλύτερο από το 2 τότε θα γίνει αντιμετάθεση και το στοιχείο 2 που βρίσκεται στην δεύτερη θέση θα μετακινηθεί και θα πάει στην πρώτη θέση. Όποτε ο πίνακας θα είναι A[2 4 3 1]. Συνεχίζοντας τον πίνακα προς τα δεξιά συγκρίνονται τα στοιχεία που βρίσκονται στις θέσεις 2 και 3. Στη δεύτερη θέση βρίσκεται το στοιχείο 4 και στην τρίτη θέση το 3. Επειδή το 4 είναι μεγαλύτερο από το 3 τότε το στοιχείο που βρίσκεται στην τρίτη θέση θα μετακινηθεί και πάει στην δεύτερη θέση άρα ο πίνακας θα είναι στη μορφή A[2 3 4 1]. Σαρώνοντας πάλι τον πίνακα προς τα δεξιά συγκρίνονται τα στοιχεία που βρίσκονται στην τρίτη θέση και στην τέταρτη θέση. Επειδή το στοιχείο που βρίσκεται στην τρίτη θέση (δηλαδή το 4) είναι μεγαλύτερο από το στοιχείο που βρίσκεται στην τέταρτη θέση που είναι το στοιχείο 1, τότε θα μετακινηθεί το 1 στην θέση τρίτη και ο πίνακας θα είναι A[2 3 1 4].

- **Πέρασμα 2ο**: Στο δεύτερο βήμα σαρώνουμε τον πίνακα πάλι από την αρχή, σε αυτή την περίπτωση έχουμε επανάληψη. Σαρώνοντας τον πίνακα προς τα δεξιά παρατηρείται ότι το στοιχείο 3 που βρίσκεται στην δεύτερη θέση είναι μεγαλύτερο από το στοιχείο 1 που βρίσκεται στην τρίτη θέση, όποτε γίνεται αντιμετάθεση των τιμών και ο πίνακας θα είναι στην μορφή A[2 1 3 4].

- **Πέρασμα 3ο**: Τέλος στο βήμα τρίτο σαρώνουμε τον πίνακα πάλι από την αρχή ως το τέλος και προκύπτει ότι το στοιχείο 2 που βρίσκεται στην πρώτη θέση είναι μεγαλύτερο από το στοιχείο 1 που βρίσκεται στην δεύτερη θέση και τότε το 1 μετακινείται στην πρώτη θέση, δηλαδή A[1 2 3 4]. Σε αυτό το σημείο η ταξινόμηση τελειώνει και ο πίνακας είναι ταξινομημένος.

ΠΟΛΥΠΛΟΚΟΤΗΤΑ:Ο χρόνος εκτέλεσης του αλγόριθμου ταξινόμησης φυσαλίδας γίνεται στη χειρότερη περίπτωση με N-1 συγκρίσεις αρχικά, και N-2 στο δεύτερο βήμα. Οπότε συμπεραίνεται ότι $1 + 2 + 3 + ... + N - 1 = \frac{N(N-1)}{2} = O(N^2)$ [9].

Αλγοριθμος 4.4: *Αλγόριθμος γρήγορης ταξινόμησης [8]*

```
quicksort(A,p,r)
if p < r:
    q=partition(A,p,r)
    quicksort(A,p,q-1)
    quicksort(A,q+1,r)
```

4.4 Διαίρει και βασίλευε

Η Διαίρει και βασίλευε ή αλλιώς (διαίρει και κυρίευε) είναι μια τεχνητή που χρησιμοποιείται σε αποδοτικούς αλγόριθμους όπως είναι οι αλγόριθμοι ταξινόμησης της γρήγορης ταξινόμησης (quick sort) και της ταξινόμησης με συγχώνευση (merge sort) όπως θα δούμε παρακάτω. Γενικά οι αλγόριθμοι διαίρει και βασίλευε αρχικά παίρνουν το πρόβλημα και το διαιρούν σε μικρότερα υποπροβλήματα τα οποία επιλύονται αναδρομικά καλώντας τον εαυτό τους και στη συνέχεια συνδυάζουν τις λύσεις τους ώστε να πάρουν το αποτέλεσμα. Τα βήματα για την επίλυση του διαίρει και βασίλευε είναι τα εξής:

- **Διαιρεί** το πρόβλημα σε μικρότερα υποπροβλήματα μικρότερου μεγέθους.
- **Επιλύει** το κάθε πρόβλημα αναδρομικά μία ή περισσότερες φορές καλώντας τον εαυτό τους για την επίλυση.
- **Συνδιάζει** τις λύσεις ώστε να συνθέσει την λύση [8].

4.4.1 Γρήγορη ταξινόμηση

Η γρήγορη ταξινόμηση (quick sort) είναι ένας από τους πιο αποδοτικούς και γρήγορους αλγορίθμους των συγκριτικών αλγορίθμων σε ταχύτητα και χρόνο και ανήκει στην κατηγορία του διαίρει και βασίλευε διασπώντας το πρόβλημα σε υποπροβλήματα και χρησιμοποιώντας την αναδρομή για την επίλυση. Ένα μειονέκτημα είναι ότι στη χειρότερη περίπτωση απαιτεί $O(n^2)$ χρόνο εκτέλεσης με αποτέλεσμα να μην χρησιμοποιείται συχνά αυτή η περίπτωση. Στη χειρότερη περίπτωση βρίσκεται, όταν τα στοιχεία στο πίνακα είναι ταξινομήμενα και το pivot στοιχείο επιλέγεται να είναι στη πρώτη θέση του πίνακα.

Κατά την διαδικασία της γρήγορης ταξινόμησης αρχικά αν περιέχει ο πίνακας κανένα ή ένα στοιχείο τότε δεν γίνεται τίποτα και επιστρέφεται ο πίνακας, διαφορετικά επιλέγεται ένα στοιχείο p που ονομάζεται pivot. Το στοιχείο p επιλέγεται τυχαία στο πίνακα και για να βρεθεί κάνει χρόνο $O(n)$ βήματα. Παρακάτω βλέπουμε τον αλγόριθμο της γρήγορης ταξινόμησης (βλπ. 4.4).

Στη συνέχεια χωρίζεται ο πίνακα σε δύο μέρη. Ο διαχωρισμός του πίνακα γίνεται με την μέθοδο της partition η αλλιώς διαμέρισης στα ελληνικά. Η partition χωρίζει τον πίνακα σε δύο μέρη. Το πρώτο μέρος περιέχει όλα τα στοιχεία που είναι μικρότερο του στοιχείου p και το δεξί μέρος περιέχει τα στοιχεία που είναι μεγαλύτερο του p. Παρακάτω βλέπουμε τον αλγόριθμο της partition (βπλ. 4.5).

Αλγοριθμος 4.5: *Κλήση της Partition [8]*

```
partition(A,p,r}:
x=A[r];
i = p − 1;
for j=0 to j−1;
  if A[j]<x
  i= i + 1;
   swap(A[i], A[j])
   swap(A[i+1], A[r])
    return i+1
```

Συνεχίζοντας καλείται αναδρομικά ο αλγόριθμος της ταξινόμησης για την επίλυση του, πρώτα από το αριστερό μέρος και μετά στο δεξί μέρος και παίρνουμε τα αποτελέσματα. Τέλος επιστρέφεται ο πίνακας να είναι ταξινομημένος.

Στο σχήμα 4.1 απεικονίζεται ένα παράδειγμα της γρήγορης ταξινόμησης ενός πίνακα οκτό στοιχείων A[67 25 30 42 15 18 23 38]. Αρχικά επιλέγουμε το στοιχείο p (pivot). Το στοιχείο pivot σε αυτή την περίπτωση επιλέγεται να είναι το τελευταίο στοιχείο που είναι το 38. Με την μέθοδο ≪διαίρει≫ χωρίζεται ο πίνακας σε δύο ίσα μέρη τέτοιος ώστε στο αριστερό μέρος να βρίσκονται τα στοιχεία που είναι μικρότερα του p=38 και στο δεξί μέρος να βρίσκονται τα στοιχεία που είναι μεγαλύτερα του p=38. Στην συνέχεια ταξινομούνται τα δύο μέρη καλώντας την αναδρομή για την επίλυση και τέλος συνδέονται με τέτοιο τρόπο ώστε ο τελικός πίνακας να είναι ταξινομημένος στα αριστερά τα στοιχεία είναι μικρότερα από το pivot στοιχείο και δεξιά τα στοιχεία να είναι μικρότερα από το pivot, και το pivot να βρίσκεται στη σωστή θέση.

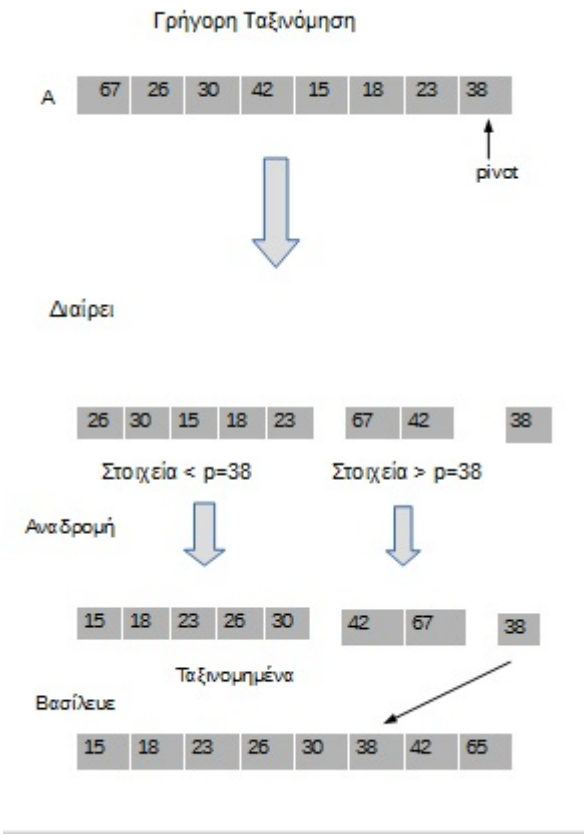
ΠΟΛΥΠΛΟΚΟΤΗΤΑ: Έχει αποδειχθεί ότι σε χρόνο εκτέλεσης στην καλύτερη και μέση περίπτωση κάνει $O(n \log n)$ βήματα για να ταξινομήσει τα στοιχεία μιας και ειναι η καλύτερη λύση του αλγόριθμου. Επίσης όταν το στοιχείο pivot είναι στην αρχή του πίνακα και ο πίνακας είναι ταξινομημένος τότε ο αλγόριθμος της γρήγορης ταξινόμησης στην χειρότερη περίπτωση έχει χρόνο εκτέλεσης $O(n^2)$. Η καλύτερη περίπτωση είναι όταν το στοιχείο pivot είναι στην μέση του πίνακα και ο πίνακας καλό θα είναι να μην είναι ταξινομημένος. Στη περίπτωση αυτή ο χρόνος εκτέλεσης χρειάζεται $O(n \log n)$ για να ταξινομηθεί. Ας υποθέσουμε ότι $T(n)$ είναι ο χρόνος εκτέλεσης. Άρα σύμφωνα με τα παραπάνω τότε προκύπτει ότι:

$$T(n) = \begin{cases} 1, & n \leq 1 \\ 2T(\frac{n}{2}) + \Theta(n), & n > 1 \end{cases}$$

Λύνωντας την εξίσωση $2T(\frac{n}{2}) + \Theta(n)$ προκύπτει $O(n \log n)$ [8].

4.4.2 Ταξινόμηση με συγχώνευση

Η ταξινόμηση με συγχώνευση (merge sort) ανήκει στους συγκριτικούς αλγορίθμους και είναι μία τεχνική που ανήκει στην κατηγορία διαίρει και βασίλευε και έχει αποδειχθεί ότι είναι ένας από τους πιο αποτελεσματικούς αλγορίθμους. Κατά την διαδικασία της ταξινόμη-



Σχήμα 4.1: *Γρήγορη ταξινόμηση*

σης ο πίνακας διαιρείται σε μικρότερα τμήματα μεγέθους ίσο $\frac{n}{2}$. Η διαίρεση των τμημάτων θα σταματήσει μέχρι να ισχύει $\frac{n}{2} = 1$. Στη συνέχεια ταξινομεί τα μέρη χρησιμοποιώντας την ταξινόμηση με συγχώνευση επιλύοντας αναδρομικά, και στη συνέχεια τα συγχωνεύει έτσι ώστε να προκύψει ο ταξινομημένος πίνακας. Η μέθοδος υλοποιείται χρησιμοποιώντας τον εαυτό της αναδρομικά για να επιλύσει το πρόβλημα. Η λειτουργία της ταξινόμησης με συγχώνευσης είναι ως εξής:

- Διαιρεί τον πίνακα A με n στοιχεία σε $\frac{n}{2}$ μέρη.
- Ταξινομεί τα δύο μέρη καλώντας αναδρομικά τον εαυτό της.
- Συγχωνεύει.

. Παρακάτω βλέπουμε τον αλγόριθμο της ταξινόμησης με συγχώνευσης (βλπ. 4.6).

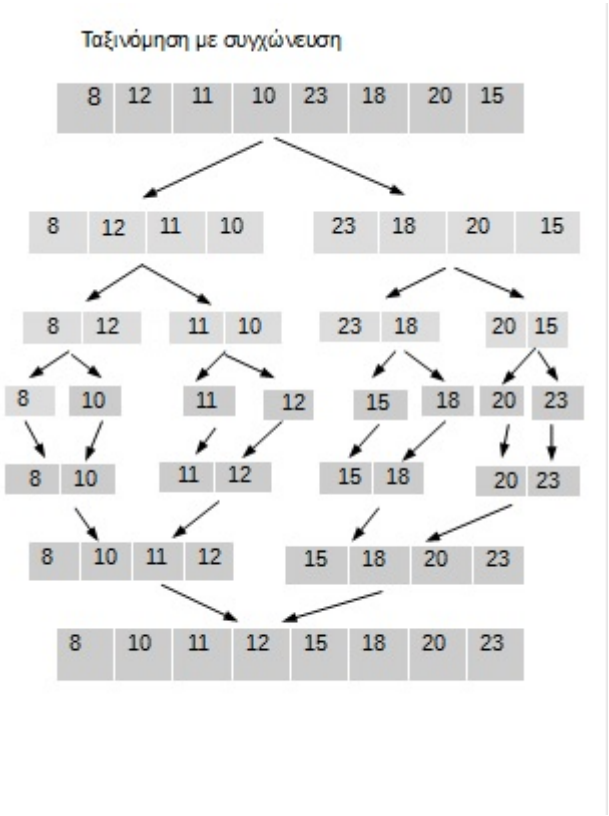
Για να ταξινομηθεί ο πίνακας τότε καλείται η κλήση της MERGE-SORT(A,p,r). Για να συγχωνευτούν οι υποπίνακες καλείται η MERGE(A,p,q,r) που είναι υπεύθυνη για την συνένωση των δύων πινάκων. Η μέθοδος της αναδρομής θα τελειώσει μέχρι το μήκος του πίνακα να είναι ίσο με ένα. Δηλαδή θα τελειώσει όταν ισχύει $\frac{n}{2} = 1$ [8].

Στο σχήμα 4.2 βλέπουμε την ταξινόμηση με συγχώνευση ενός πίνακα οκτώ στοιχείων A=[8 12 11 10 23 18 20 15].

ΠΟΛΥΠΛΟΚΟΤΗΤΑ: Όσον αφορά για τον χρόνο εκτέλεσης θα ορίσουμε με $T(n)$ τον χρόνο εκτέλεσης του αλγορίθμου της ταξινόμησης με συγχώνευσης. Αν το $n = 1$ όπου n

Αλγοριθμος 4.6: *Αλγόριθμος ταξινόμησης με συγχώνευση [8]*

```
MERGE-SORT(A,p,r)
 if p<r
Then q=(p+r)/2
   MERGE-SORT(A,p,q)
   MERGE-SORT(A,q+1,r)
   MERGE(A,p,q,r)
```



Σχήμα 4.2: *Ταξινόμηση συγχώνευσης*

το μέγεθος του πίνακα τότε ο χρόνος εκτέλεσης θα είναι σταθερός και δεν θα μεταβάλεται δηλαδή $\Theta(1)$, διαφορετικά αν το $n \geq 2$ τότε ο πίνακας θα διαιρεθεί σε μικρότερα μέρη. Όσον αφορά για την διαμέριση σε μικρότερα τμήματα ο αλγόριθμος χρειάζεται $2T(\frac{n}{2})$ μέχρι να ταξινομηθούν οι δύο υποπίνακες και $\Theta(n)$ χρειάζεται για την συγχώνευση των υποπινάκων δηλαδή μέχρι να συγχωνευθούν καλώντας την MERGE για την συγχώνευση των δύων ταξινομημένων υποπινακων. Όπότε ο χρόνος εκτέλεσης της ταξινόμησης με συγχώνευσης θα είναι $T(n) = 2T(\frac{n}{2}) + \Theta(n)$.

$$T(n) = \begin{cases} \Theta(1), & n = 1 \\ 2T(\frac{n}{2}) + \Theta(n), & n > 1 \end{cases}$$

Στη περίπτωση που το $n \geq 1$ τότε λύνοντας την αναδρομή προκύπτει $O(n \log n)$ [12].

4.5 Κατω φράγματα

Μέχρι στιγμή γνωρίζουμε πως η καλύτερη περίπτωση σε χρόνο εκτέλεσης για να λυθεί ένας αλγόριθμος της συγκριτικής ταξινόμησης είναι $O(n \log n)$ όπως είδαμε στην ταξινόμηση με συγχώνευση και στην γρήγορη ταξινόμηση, και στη χειρότερη περίπτωση είναι $O(n^2)$ όπως συναντήαμε στην ταξινόμηση με εισαγωγή, στην ταξινόμηση με φυσαλίδα, και στην ταξινόμηση με επιλογή. Στην ενότητα αυτή θα μελετηθεί το κάτω φράγμα για αυτούς τους αλγορίθμους. Το κάτω φράγμα αφορά για συγκριτικούς αλγόριθμους και που χρησιμοποιούν συγκρίσεις απαιτώντας χρόνο εκτέλεσης $\Omega(n \log n)$. Επίσης το κάτω φράγμα είναι αποδοτικό για το χρόνο εκτέλεσης στην χειρότερη περίπτωση. Σε αυτό το σημείο θα αποδειχθεί ότι οι αλγόριθμοι συγκριτικής ταξινόμησης κάνουν $\Omega(n \log n)$ συγκρίσεις στη χειρότερη περίπτωση.

Οι συγκριτικοί αλγόριθμοι αναπαρίσταται με το δέντρο αποφάσεων. Το δέντρο αποφάσεων χρησιμοποιείται για την λήψη αποφάσεων. ≪Πιο συγκεκριμένα το δέντρο αποφάσεων είναι ένα πλήρες δυαδικό δέντρο που αντιπροσωπεύει τις συγκρίσεις που πραγματοποιούνται μεταξύ στοιχείων από έναν συγκεκριμένο αλγόριθμο ταξινόμηση [8]≫. Ένα δέντρο αποφάσεων αποτελείται από την ρίζα και τους κόμβους ή αλλιώς φύλλα. Η ρίζα είναι η κεφαλή και οι κόμβοι τα παιδιά της. Για κάθε κόμβο ισχύει ο εξής κανόνας:

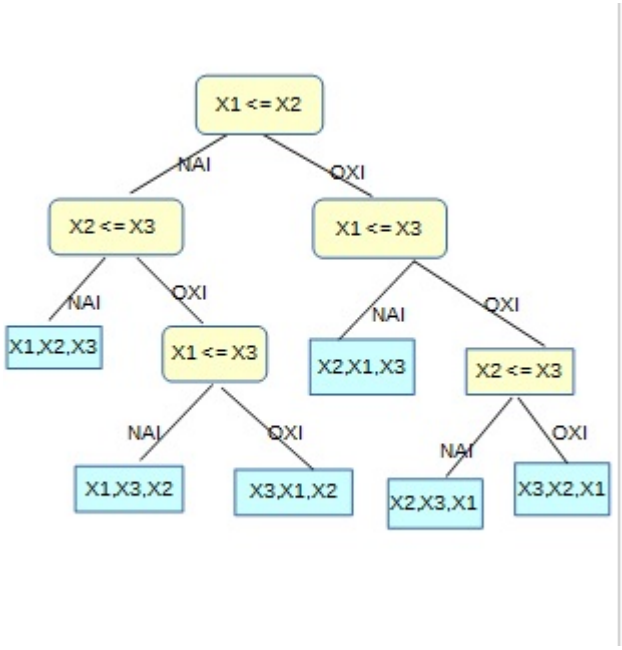
- Για το αριστερό υποδέντρο ισχύει ότι ο κάθε κόμβος θα πρέπει να είναι μικρότερος από την ρίζα.
- Για κάθε δεξί υποδέντρο ισχύει ότι ο κάθε κόμβος θα πρέπει να είναι μεγαλύτερος από την ρίζα.

Πιο συγκεκριμένα η ρίζα του δέντρου αποτελεί την είδοσο ενός πίνακα $A[n]$ όπου n είναι ο αριθμός των στοιχείων του πίνακα. Για την κατάληξη ενός τερματικού κόμβου ξεκινώντας πάντα από την ρίζα ακολουθούν συγκρίσεις μεταξύ των στοιχείων ανά δύο. Στο σχήμα 4.3 απεικονίζεται ένα δυαδικό δέντρο αποφάσεων με είσοδο τρίων στοιχείων χρησιμοποιώντας την ταξινόμηση εισαγωγής. Αρχικά ξεκινάει από την ρίζα και συγκρίνεται το στοιχείο $x1$ και $x2$. Αν το $x1$ είναι μικρότερο του $x2$ τότε η μεταβίβαση θα γίνει στο αριστερό υποδέντρο, διαφορετικα η μεταβίβαση θα γίνει στο δεξιό υποδέντρο. Με την ίδια διαδικασία θα συνεχιστεί μέχρι να καταλήξουμε σε ένα τερματικό κόμβο. Ο κάθε κόμβος αναπαριστά στην μορφή $\langle \pi(1), \pi(2), \pi(3) \rangle$. Η μετάβαση από την ρίζα μέχρι την κατάληξη ενός τερματικού κόμβου κάνει $n!$. Οπότε το μονοπάτι που θα ακολουθήσει από την ρίζα του δέντρου σε ένα τερματικό κόμβο θα είναι $n!$ καθώς επίσης κσι ο αριθμός των φύλλων του δέντρου. Παρακάτω στο θεώρημα 4.1 θα αποδειχθεί ότι με την βοήθεια των δέντρων αποφάσεων υπάρχει ένα κάτω φράγμα $\Omega(n \log n)$.

Θεώρημα 4.1. *Οποιοσδήποτε αλγόριθμος συγκριτικής ταξινόμησης χρησιμοποιεί ένα κάτω φράγμα το οποίο απαιτεί $\Omega(n \log n)$ συγκρίσεις.*

Απόδειξη

Ας υποθέσουμε ότι h είναι το ύψος ενός δέντρου αποφάσεων για κάποια είσοδο με n στοιχεία. Ένα δυαδικό δέντρο αποφάσεων έχει ύψος δέντρο 2^h κόμβους, και αριθμό φύλλων έχει $n!$. Όμως $n! = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 2$.



Σχήμα 4.3: *Δέντρο αποφάσεων*

Άρα, $2^h \geq n! \Rightarrow \log(2^h) \geq \log(n!) \Rightarrow h \cdot log2 \geq \log(n!) \Rightarrow h \geq \log(n!)$
Όμως $\log(n!) = \log(n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 2) = \log(n + \log(n-1) + \log(n-2) + \ldots + \log 2 = \sum_{i=2}^{n} \log i = n \log n = \Omega(n \log n)$
Άρα $\log(n!) \in \Omega(n \log n)$ *οπότε* $h = \Omega(n \log n)$ *[12].*

1

[1] Σύμφωνα με την προσέγγιση του Striling ισχύει ότι $n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1 + \Theta\left(\frac{1}{n}\right)\right) = \Omega(n \log n)$ Για περισσότερες πληροφορίες βλπ. [8]

Κεφάλαιο 5

Αλγόριθμοι ταξινόμηση σε γραμμικό χρόνο $O(n)$

Σε αυτό το κεφάλαιο θα αναφερθούμε στους αλγορίθμους ταξινόμησης γραμμικού χρόνου στους οποίους θα αναλυθεί και η πολυπλοκότητα τους.

5.1 Γραμμικός Χρόνος $O(n)$

Στο προηγούμενο κεφάλαιο είδαμε τους αλγόριθμους συγκριτικής ταξινόμησης η οποίοι βασίζονταν στις συγκρίσεις μεταξύ των στοιχείων σε ένα πίνακα για την διεξαγωγή των αποτελεσμάτων και στις μετακινήσεις, και συναντήσαμε αλγόριθμους όπου στην καλύτερη περίπτωση έχουν χρόνο εκτέλεσης $O(n \log n)$ όπως είναι ο αλγόριθμος της γρήγορης ταξινόμησης και της ταξινόμησης με συγχώνευσης. Επίσης αποδείχθηκε ότι οι αλγόριθμοι αυτοί χρησιμοποιούν ένα κάτω φράγμα και απαιτούν $\Omega(n \log n)$ βήματα στην χειρότερη περίπτωση μιας και είναι η κατάλληλη χρήση.

Όμως υπάρχουν αλγόριθμοι όπου έχουν χρόνο εκτέλεσης στη χειρότερη περίπτωση καλύτερο από $O(n \log n)$ είναι και αποδοτικότεροι και η ανάλυση τους δεν βασίζονται στην σύγκριση αλλά σε άλλες τεχνικές όπως για παράδειγμα ότι ταξινομούν τους αριθμούς σε κάποιο διάστημα. Αυτοί οι λεγόμενοι είναι οι μη συγκριτικοί ή αλλιώς αλγόριθμοι γραμμικού χρόνου οι οποίοι δεν χρησιμοποιούν τις συγκρίσεις. Ένας αλγόριθμος λέγεται γραμμικού χρόνου όταν ο χρόνος εκτέλεσης είναι $O(n)$. Επίσης ισχύει ότι για αρκετά μεγάλο μέγεθος εισόδου, ο χρόνος εκτέλεσης αυξάνεται γραμμικά. Επίσης οι αλγόριθμοι τέτοιου είδους μπορούν να χρησιμοποιηθούν στην περίπτωση που εμφανίζουν κάποια ιδιότυπα χαρακτηριστικά όπως για παράδειγμα να ακολουθούν ομοιόμορφη κατανομή ή εμφανίζουν αποκλίσεις. Οι αλγόριθμοι ταξινόμηση που τρέχουν σε γραμμικό χρόνο εκτέλεσης είναι τρεις: η Απαριθμιτική ταξινόμηση (counting sort), η αριθμοτακτική ταξινόμηση (radix sort) και η ταξινόμηση με δοχεία (bucket sort).

5.2 Απαριθμιτική ταξινόμηση

Η απαριθμιτική ταξινόμηση (counting sort) είναι αλγόριθμος γραμμικού χρόνου $O(n)$ και ανήκει στους μη συγκριτικούς αλγόριθμους. Η ταξινόμηση δεν βασίζεται στις συγκρίσεις, οπότε δεν ισχύει το κάτω φράγμα για αυτόν τον αλγόριθμο. Το ζητούμενο της Απαριθμιτική

Αλγοριθμος 5.1: *Αλγόριθμος απαριθμιτικής ταξινόμησης [8]*

```
Counting_sort(X,Y,C,n,m)
for i=0 μέχρι m
  C[i]=0
for j=0 μέχρι n−1
  C[X[i]] = C[X[i]] + 1
for i=1 μέχρι k
  C[i] = C[i] + C[i−1]
for j=n−1 μέχρι 1
Y[C[X[j]] = Y[j]
C[X[j]] = C[X[j]] − 1
```

ταξινόμησης είναι ότι για κάθε στοιχείο x, να βρεθεί το πλήθος των στοιχείων που είναι μικρότερα από το στοιχείο x και στο τέλος να τοποθετηθεί στη σωστή θέση στην έξοδο.

Πιο συγκεκριμένα κατά την διαδικασία της ταξινόμησης αρχικά μετράει για κάθε τιμή πόσες φόρες εμφανίζεται στον πίνακα, στην συνέχεια υπολογίζει το άθροισμα των στοιχείων του x στοιχείου που είναι μικρότερα του, και στο τέλος δημιουργεί έναν άλλον πίνακα όπου τα στοιχεία είναι ταξινομημένα. Όσον αφορά για τα δεδομένα της υλοποίησης της ταξινόμησης χρειάζονται τρεις πίνακες. Ο πίνακας $X[1..n]$ θα είναι ο πίνακας της εισόδου με n στοιχεία, ο πίνακας $C[0..m]$ θα είναι ο προσωρινός αποθηκευτικός χώρος όπου m θα είναι η μεγαλύτερη ακέραια τιμή που υπάρχει στον πίνακα $X[1..n]$ και δηλώνει το εύρος τιμών, και ο πίνακας $Y[0..n]$ θα είναι ο πίνακας της εξόδου ο οποίος θα δίνει τα αποτελέσματα και τα στοιχεία να είναι ταξινομημένα. Επίσης ο αλγόριθμος λειτουργεί μόνο για ακέραιους αριθμούς ο οποίος κάθε ακέραιος θα είναι μεταξύ των τιμών από 0 έως m.

Σκοπός της ταξινόμησης είναι την πρώτη φόρα να υπολογίσει για κάθε τιμή i που υπάρχει στο διάστημα μεταξύ 0 έως m, το πλήθος των στοιχείων που ισούται με i, δηλαδή να υπολογίσει πόσες φορές υπάρχει η τιμή στον πίνακα $X[n]$ που ισούται με το i. Με άλλα λόγια για $i = 0$ το $C[0]$ θα περιέχει το άθροισμα των στοιχείων του $X[n]$ που θα έχουν τις τιμές μηδέν, ομοίως το $C[2]$ θα περιέχει το άθροισμα των στοιχείων του πίνακα $X[n]$ που θα έχουν την τιμή 2 κ.ο.κ.

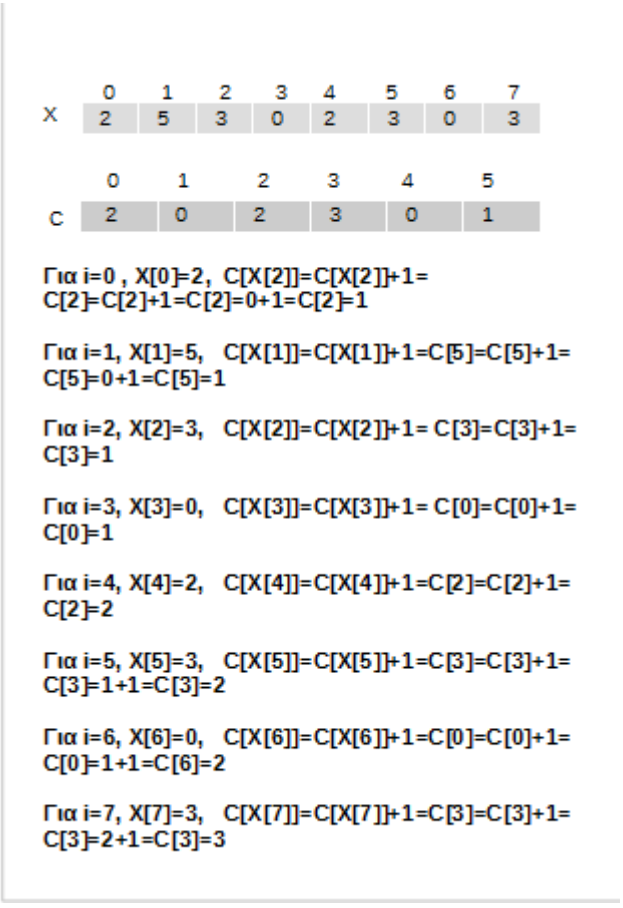
Στην συνέχεια θα πρέπει υπολογισθεί το νέο $C[i..m]$ το οποίο θα περιέχει το άθροισμα των στοιχείων που υπάρχουν στο διάστημα 0 έως m που είναι μικρότερο από την τιμή τους. Δηλαδή στη θέση 4 το νέο $C[4]$ θα πρέπει να περιέχει το πλήθος των στοιχείων που είναι μικρότερο ή ίσο από το 4.

Τέλος το στοιχείο $X[i]$ τοποθετείται στην σωστή θέση στον ταξινομημένο πίνακα της εξόδου $Y[1..n]$ με την προϋπόθεση ξεκινώντας από την τελευταία θέση του πίνακα $X[n]$.

Μια σημαντική ιδιότητα για την απαριθμιτική ταξινόμηση είναι η ευστάθεια του αλγόριθμου. Λέγοντας ένα αλγόριθμο ευσταθή σημαίνει όταν τα στοιχεία με τις ίδιες τιμές που εμφανίζονται στημ είσοδο, η διάταξη τους στην έξοδο του πίνακα παραμένει ίδια και δεν αλλάζει όπως ήταν η διάταξη τους και στην είσοδο. Αντίθετα ένας αλγόριθμος λέγεται ασταθής όταν τα στοιχεία δεν διατηρούν την διάταξη του πίνακα. [8].

Παρακάτω βλέπουμε ένα παράδειγμα της απαριθμιτικής ταξινόμησης. Έστω ο πίνακας X

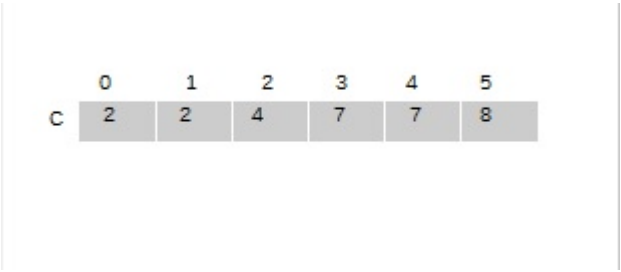
με είσοδο οκτώ στοιχεία X=[2 5 3 0 2 3 0 3] και ας υποθέσουμε ότι το εύρος τιμών είναι $k = 5$ δηλαδή παίρνει τιμές από το 0 έως το 5. Στο σχήμα 5.1 απεικονίζει τον πίνακα $X[1..n]$, και το $C[1..m]$ πίνακα μαζί και τα αποτελέσματα. Ο πίνακας C υπολογίζει για κάθε στοιχείο το πλήθος του που εμφανίζεται στον πίνακα X. Πιο αναλυτικά θα το δούμε στην συνέχεια. Κατά των βρόχο 1 και του 2 για $i = 0$ έως $m = 5$ τότε $c[0] = c[1] = c[2] = c[3] = c[4] = c[5] = 0$. Στην γραμμή 3 και 4 υπολογίζεται το πλήθος των στοιχείων του $C[A[j]]$ για κάθε τιμή που υπάρχει στο διάστημα 0 έως 5, δηλαδή υπολογίζει πόσες φορές εμφανίζεται το στοιχείο στον πίνακα $X[]$. Άρα για $j = 0$, τότε υπολογίζεται η πράξη $C[X[j]] = C[X[j]] + 1 \rightarrow C[X[0]] = C[X[0]] + 1 \rightarrow C[2] = C[2] + 1 \rightarrow C[2] = 1$. Με αυτόν τον τρόπο υπολογίζονται και τα υπόλοιπα στοιχεία. Δηλαδή $C[1] = 0, C[3] = 3, C[4] = 0, C[5] = 1, C[0] = 2$. Άρα θα προκύψει ο πίνακας C=[2 0 2 3 0 1].



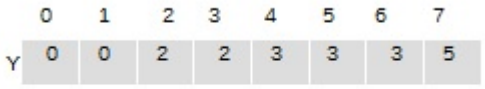
Σχήμα 5.1: *Το C[i] περιέχει το πλήθος των στοιχείων πόσες φορές εμφανίζεται το ι*

Στο σχήμα 5.2 το $C[i]$ υπολογίζει το πλήθος των στοιχείων που είναι μικρότερα ή ίσα από i. Για την κατανόηση ας υπολογίσουμε το $C[1]$. Άρα για $i = 1$, τότε $C[i] = C[i] + 1 \Rightarrow C[1] = C[1] + C[0] \Rightarrow C[1] = 0 + 1 \Rightarrow C[1] = 1$. Ομοίως για $i = 2, C[2] = C[2] + C[1] \Rightarrow C[2] = 2 + 0 \Rightarrow C[2] = 2$ κ.ο.κ.

Στο σχήμα 5.4 βλέπουμε την έξοδο του πίνακα. Για να ταξινομηθεί ο πίνακας ξεκινάμε από την τελευταία θέση του πίνακα $X[i]$ της εισόδου. Στο συγκεκριμένο παράδειγμα το τελευταίο στοιχείο που βρίσκεται στη τελευταία θέση είναι το στοιχείο $X[7] = 3$ δηλαδή το τρία. Όμως $C[3] = 7$, αυτό σημαίνει ότι υπάρχουν 7 αριθμοί που είναι μικρότεροι ή ίσοι

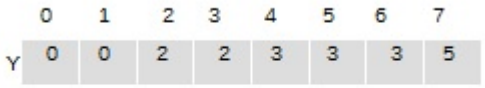


Σχήμα 5.2: *Το C[i] περιέχει το πλήθος των στοιχείων που είναι ίσα ή μικρότερο με i*



Σχήμα 5.3: *Η απαριθμητική ταξινόμηση*

του αριθμού τρία. Άρα το τρία θα τοποθετηθεί στην έξοδο ξεκινώντας από το $Y[0]$ στη θέση $Y[6]$. Στη συνέχεια το $C[i]$ μειώνεται κατά μία μονάδα. Ομοίως γίνεται με το ίδιο τρόπο και για τα υπόλοιπα στοιχεία. Άρα ο ταξινομημένος πίνακας θα είναι: Y=[0 0 2 2 3 3 3 5].



Σχήμα 5.4: *Η απαριθμητική ταξινόμηση*

Πολυπλοκότητα: Ο αλγόριθμος της απαριθμητικής δεν ισχύει για το κάτω φράγμα μιας και δεν βασίζεται σε συγκρίσεις αλλά στην απαρίθμιση τιμών. Ο χρόνος εκτέλεσης της απαριθμητικής ταξινόμησης απαιτεί $O(n+m)$ βήματα σε χρόνο γραμμικού χρόνου, όπου m είναι η μεγαλύτερη τιμή του πίνακα $X[n]$ και δηλώνει το εύρος τιμών. Όταν $m \in O(n)$ τότε η πολυπλοκότητα του αλγόριθμου απαριθμητικής ταξινόμησης είναι $O(n)$ γραμμικού χρόνου [8].

5.3 Αριθμοτακτική ταξινόμηση

Η αριθμοτακτική ταξινόμηση είναι γραμμικού χρόνου και ανήκει στους μη συγκριτικούς αλγορίθμους και στην ταξινόμηση οι αριθμοί ταξινομούνται με το βάση το ψηφίο τους. Το πόσα ψηφία θα περιέχει ένας αριθμός εξαρτάται από το d-ψήφιος αριθμό. Αν είχαμε τον ακέραιο αριθμό 321 τότε τα ψηφία θα ήταν τρία. Επίσης ανάλογα με τον d-ψήφιο αριθμό, τόσες στήλες θα περιλαμβάνουν. Δηλαδή, ένας τριψήφιος αριθμός θα αποτελούσε από τρεις στήλες.

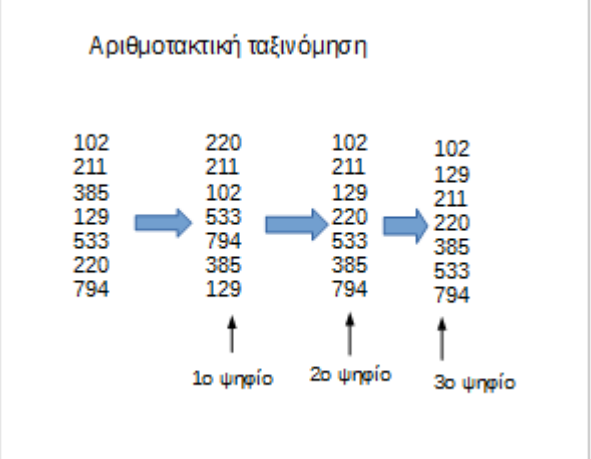
Η βασική ιδέα του αλγόριθμου είναι να ταξινομήσει τους αριθμούς με βάση το ψηφίο

Αλγοριθμος 5.2: *Αριθμοτακτική ταξινόμηση [8]*

for i=0 εώς n
Ταξινομούμε τον πίνακα X ως προς το μικρότερο ψηφίο i με ένα ευσταθή αλγόριθμο χρησιμοποιώντας την απαριθμιτική ταξινόμηση

τους από το λιγότερο ψηφίο πρώτα, προς το μεγαλύτερο ψηφίο τάξης. Δηλαδή να ταξινομήσει πρώτα τους αριθμούς με βάση το πρώτο ψηφίο που βρίσκεται από τα δεξιά της στήλης και που είναι οι είναι οι μονάδες, και μετά να ταξινομήσει τους αριθμούς με βάση το δεύτερο ψηφίο τους τις δεκάδες, και στη συνέχεια τους αριθμούς με το τρίτο ψηφίο τις εκατοντάδες κ.ο.κ. Για να λειτουργήσει σωστά ο αλγόριθμος θα πρέπει η ταξινόμηση ως προς το ψηφίο να γίνει με έναν ευσταθή αλγόριθμο χρησιμοποιώντας την ταξινόμηση της απαριθμιτικής. Επίσης η ευστάθεια παίζει σημαντικό ρόλο στην αριθμοτακτική ταξινόμηση.

Παρακάτω στο σχήμα 5.5 απεικονίζεται η αριθμοτακτική ταξινόμησης ενός πίνακα A[102 211 385 129 633 220 794] ο οποίος αποτελείται από επτά τριψήφιους αριθμούς. Αρχικά ταξινομούνται πρώτα οι αριθμοί ως προς το πρώτο ψηφίο από τα δεξιά, στη συνέχεια ταξινομούνται οι αριθμοί ως προς το δεύτερο ψηφίο και τέλος ταξινομούνται οι αριθμοί ως προς το τρίτο ψηφίο. Η ταξινόμηση σε όλες τις περιπτώσεις ξεκινάει από το μικρότερο προς το μεγαλύτερο ψηφίο τους.

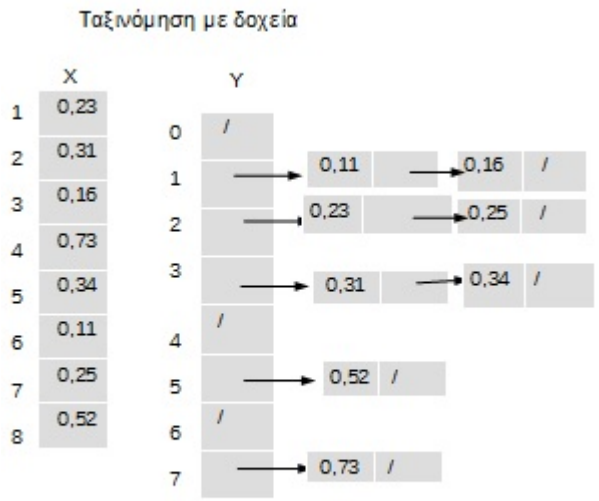


Σχήμα 5.5: *Αριθμοτακτική ταξινόμηση*

ΠΟΛΥΠΠΛΟΚΟΤΗΤΑ: Ο χρόνος εκτέλεσης της αριθμοτακτικής ταξινόμησης εξαρτάται από το πλήθος των ψηφίων που θα ταξινομηθούν. Αν το κάθε στοιχείο του πίνακα $X[n]$ έχει d ψηφία, και το κάθε ψηφίο μπορεί να λάβει μία τιμή μεταξύ του m τιμών τότε ο χρόνος εκτέλεσης της αριθμοτακτικής ταξινόμησης είναι $O(d(n+m))$ όπου n είναι το μέγεθος του πίνακα και m είναι το πλήθος των ψηφίων. Στην περίπτωση όπου d είναι σταθερό και το $m = O(n)$, τότε ο χρόνος εκτέλεσης είναι γραμμικού δηλαδή $O(n)$ [8].

5.4 Η ταξινόμηση με δοχεία

Η ταξινόμηση με δοχεία είναι αλγόριθμος με χρόνο αναμενόμενου γραμμικού $O(n)$. Πιο συγκεκριμένα ας υποθέσουμε ότι έχουμε n αριθμούς οι οποίοι είναι κατανεμημένοι σε ένα διάστημα $[0, 1)$ και $X[1..n]$ είναι η είσοδος και $Y[0...n-1]$ τα δοχεία που πρέπει να τοποθετηθούν τα στοιχεία. Για να είναι ομοιόμορφα κατανεμημένα στην είσοδο τα στοιχεία, θα πρέπει ότι η τιμή του κάθε στοιχείου $X[i]$ να κυμαίνεται στο διάστημα $[0, 1)$ δηλαδή να ισχύει ότι $0 \leq X[i] \leq 1$. Κατά την διαδικασία της ταξινόμησης, αφού γεμίσει ο πίνακας $X[n]$ με ομοιόμορφη κατανομή, στην συνέχεια θα πρέπει να τοποθετηθούν οι n αριθμοί στο σωστό δοχείο ανάλογα με την τιμή του κάθε στοιχείου. Αφού τοποθετηθούν τα στοιχεία στα δοχεία, ταξινομεί τα δοχεία χρησιμοποιώντας κάποιον ευσταθή αλγόριθμο όπως είναι η ταξινόμηση με εισαγωγή insertion sort και τέλος τα στοιχεία κάθε λίστα θα πρέπει είναι ταξινομημένα ώστε να είναι σε σειρά και να προκύψει η έξοδος.



Σχήμα 5.6: *Ταξινόμηση με δοχεία*

Όσον αφορά για την ορθότητα του αλγόριθμου εάν τα στοιχεία βρίσκονται στο ίδιο δοχείο τότε θα είναι σχετικά στην σωστή σειρά, καθώς επίσης το ίδιο θα βρίσκονται στην σωστή σειρά εάν τα στοιχεία είναι σε δύο διαφορετικά δοχεία.

ΠΟΛΥΠΛΟΚΟΤΗΤΑ: Στο σημείο αυτό θα αναλύσουμε τον χρόνο εκτέλεσης της ταξινόμησης με δοχεία στην χειρότερη περίπτωση. Για να βρούμε τον χρόνο εκτέλεσης θα εξετάσουμε την γραμμή 4 στον αλγόριθμο επειδή οι υπόλοιπες γραμμές έχουν γραμμικό χρόνο. Η γραμμή 4 χρησιμοποιεί την ταξινόμηση με εισαγωγής. Η ταξινόμηση με εισαγωγή έχει χρόνο εκτέλεσης $O(n^2)$ μιας και είναι τετραγωνικός. Οπότε θα ορίσουμε ότι n_i^2 είναι μια τυχαία μεταβλητή η οποία δηλώνει τον αριθμό των στοιχείων που θα πέσουν στο δοχείο $Y[i]$. Άρα ο χρόνος εκτέλεσης της ταξινόμησης με δοχεία στην χειρότερη περίπτωση χρησιμοποιώντας την ταξινόμηση με εισαγωγή θα είναι

Αλγοριθμος 5.3: *Ταξινόμηση με δοχεία [8]*

```
Ταξινόμηση με δοχεία(X,Y,n)
n=length[X]
for i=1 εώς n
  Εισάγουμε το X[i] στην αλυσίδα B[nX[i]]
for i=0 εώς n=1
  Ταξινομούμε την αλυσίδα Y[i] με την ταξινόμηση με εισαγωγή
συναρμόζουμε τις αλυσιδές Y[0] Y[1]...Y[n−1], με αυτή τη σειρά
```

$$T(n) = \Theta(n) + \sum_{i=1}^{n-1} O(n_i^2)$$

Στη συνέχεια θα υπολογισθεί η αναμενόμενη τιμή του $T(n)$ των δύων μελών δηλαδή τον αναμενόμενο χρόνο, και χρησιμοποιώντας την γραμμικότητα και των δύων μελών τότε προκύπτει ως εξής:

$$E[T(n)] = E[\Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)] = \Theta(n) + \sum_{i=0}^{n-1} E[O(n_i^2)] = \Theta(n) + \sum_{i=0}^{n-1} O(E[n_i^2])$$

Το $E[n_i^2]$ δηλώνει το αναμενόμενο πλήθος που θα πέσουν στο δοχείο $Y[i]$. Θα αποδειχθεί ότο $E[n_i^2] = 2 - \frac{1}{n}$.
Στο σημείο αυτό θα ορίσουμε ότι X_ij είναι δείκτριες τυχαίες μεταβλητές που αντιστοιχούν στο ενδεχόμενο ότι το στοιχείο $Y[i]$ κατανέμεται στο δοχείο i.
Άρα $X_ij = I\{$ Το $X[i]$ κατανέμεται στο δοχείο $i\}$ για τιμές $i = 0, 1, ..n-1$ και $j = 1, 2, ..n$.
Άρα $n_i \sum_{j=1}^{n} X_ij$.
Αφού τα στοιχεία είναι ομοιόμορφα κατανεμημένα και ισχύει ότι $0 \leq X[i] \leq 1$ τότε θα έχουν ισοπίθανη πιθανότητα για να πέσουν στα δοχεία. Άρα η πιθανότητα η δείκτρια μεταβλητή θα έχει 1 με πιθανότητα $\frac{1}{n}$ διαφορετικά 0. Επομένως

$$I = E[X_ij] = \begin{cases} 1, & \frac{1}{n} \\ 0, & 1 - \frac{1}{n} \end{cases}$$

$$E[X_ij^2] = 1^2 \cdot \frac{1}{n} + 0^2 \cdot (1 - \frac{1}{n}) = \frac{1}{n}$$

Υπολογίζοντας την αναμενόμενη τιμή $E[n_i^2]$ και χρησιμοποιώντας την γραμμικότητα και των δύων μελών τότε:

$$E[n_i^2] = E\Big[\Big(\sum_{j=1}^{n} X_ij\Big)^2\Big] = E\Big[\sum_{j=1}^{n}\sum_{k=1}^{n} X_ij \cdot X_ik\Big] = E\Big[\sum_{j=1}^{n} X_ij^2 + \sum_{1\leq j\leq n}\sum_{\substack{1\leq k\leq n \\ k\neq j}} X_ij \cdot X_ik\Big] = \sum_{j=1}^{n} E[X_ij^2] +$$

$$\sum_{1\leq j\leq n}\sum_{\substack{1\leq k\leq n \\ k\neq j}} E\Big[X_ij \cdot X_ik\Big] = \sum_{j=1}^{n} \frac{1}{n} + \sum_{1\leq j\leq n}\sum_{\substack{1\leq k\leq n \\ k\neq j}} \frac{1}{n^2} = n \cdot \frac{1}{n} + n \cdot (n-1)\frac{1}{n^2} = 1 + \frac{n \cdot (n-1)}{n^2} =$$

$$\frac{n^2 + n^2 - n}{n^2} = \frac{2n^2 - n}{n^2} = \frac{2n^2}{n^2} - \frac{n}{n^2} = 2 - \frac{1}{n}.$$

Άρα ισχύει ότι $E\left[n_i^2\right] = 2 - \frac{1}{n}$.

Επειδή οι μεταβλητές X_ij και X_ik είναι ανεξάρτητες και $i \neq k$ τότε παίρνουμε το γινόμενο της γραμμικότητας και των δύων μελών. Άρα

$$E\left[X_ij \cdot X_ik\right] = \sum_{j=1}^{n}\sum_{k=1}^{n} X_ij \cdot X_ik = \left(\sum_{j=1}^{n} X_ij\right)\left(\sum_{X_ik}^{n} X_ik\right) = E\left[X_ij\right]E\left[x_ik\right] = \frac{1}{n} \cdot \frac{1}{n} = \frac{1}{n^2}$$

Συμπερασματικά αποδείχθηκε με την χρήση δείκτριων τυχαίων μεταβλητών ότι $E\left[n_i^2\right] = 2 - \frac{1}{n}$ και αντικαθιστώντας με την εξίσωση $T[n] = \Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)$ προκύπτει ως εξής:

$$T(n) = \Theta(n) + \sum_{i=0}^{n-1} O\left(E\left[n_i^2\right]\right) = \Theta(n) + n \cdot O\left(2 - \frac{1}{n}\right) = \Theta(n)$$

Άρα ο αναμενόμενος χρόνος εκτέλεσης της ταξινόμησης με δοχεία έχει χρόνο $\Theta(n) + n \cdot O(2 - \frac{1}{n}) = \Theta(n)$. Άρα η ταξινόμηση με δοχεία έχει αναμενόμενο γραμμικό χρόνο εκτέλεσης.

Η ταξινόμηση με δοχεία έχει γραμμικό γραμμικό χρόνο $O(n)$ μόνο στη περίπτωση όπου τα στοιχεία στην είσοδο δεν θα είναι ομοιόμορφα κατανεμημένα [8].

1

[1] Οι δείκτριες τυχαίων μεταβλητών χρησιμοποιούνται στην ανάλυση πολλών αλγορίθμων οι οποίες είναι μια μέθοδος μετατροπής πιθανοτήτων σε αναμενόμενες τιμές.

Κεφάλαιο 6

Υλοποίηση αλγόριθμων

Σε αυτήν την ενότητα γίνεται μια σύγκριση των αλγόριθμων ταξινόμησης τωνσυγκριτικών αλγορίθμων και της ταξινόμησης γραμμικού χρόνου καθώς παρουσιάζεται η υλοποίηση των αλγόριθμων γραμμικού χρόνου υλοποιημένοι σε γλώσσα C καθώς και τα εκτελέσιμα αρχεία με τα αποτελέσματα τους.

6.1 Σύγκριση αλγορίθμων

Παρακάτω βλέπουμε τους πίνακες με τους χρόνους εκτέλεσης των δύων κατηγορίων στη καλύτερης και χειρότερη περίπτωση. Στον πίνακα 6.1 βλέπουμε τους χρόνους εκτέλεσης των συγκριτικών αλγορίθμων.

Συγκρίνωντας τους αλγόριθμους της συγκριτικής ταξινόμησης προκύπτουν κάποιες διαφορές στους αλγορίθμους. Ας ξεκινήσουμε με τον αλγόριθμο της εισαγωγής. Ας ξεκινήσουμε με τον αλγόριθμο της ταξινόμησης με εισαγωγής. Η ταξινόμηση εισαγωγή δουλεύει περίπου με τον αλγόριθμο της ταξινόμησης με φυσαλίδας με την διαφορά ότι τα στοιχεία της ταξινόμησης με εισαγωγής ξεκινούν από την αρχή του πίνακα προς τα δεξιά. Ένα πλεονέκτημα του αλγόριθμου της εισαγωγή είναι ότι η υλοποίηση είναι εύκολη και ότι είναι γρηγορότερος από τον αλγόριθμο της ταξινόμησης με φυσαλίδας και την ταξινόμηση με επιλογής. Ένα μειονέκτημα είναι ότι ο χρόνος είναι αργός για πολύ μεγάλο μέγεθος του n, όπου n είναι το πλήθος του πίνακα. Όσον αφορά για τον χρόνο εκτέλεσης προκύπτει ότι στη έχει γραμμικό χρόνο $O(n)$ στη περίπτωση που τα στοιχεία του πίνακα είναι ταξινομημένα ενώ στη χειρότερη περίπτωση απαιτεί $O(n^2)$.

Ο αλγόριθμος ταξινόμησης με επιλογής επιλέγει το πρώτο μικρότερο στοιχείο και το αντικαθιστά με το στοιχείο που βρίσκεται στην πρώτη θέση του πίνακα. Η διαδικασία θα συνεχιστεί μέχρι βρεθεί το τελευταίο στοιχείο που θα είναι στην θέση του. Είναι πιο αργός από την ταξινόμηση με εισαγωγή με αποτέλεσμα ο αλγόριθμος εισαγωγή να είναι η καλύτερη επιλογή, αλλά είναι πιο γρήγορος από την ταξινόμηση με φυσαλίδα. Ένα πλεονέκτημα είναι ότι η υλοποίηση είναι εύκολη αλλά ένα μειονέκτημα είναι στο χρόνο ο οποίος είναι αργός. Από ότι βλέπουμε και στον πίνακα 6.1 ο χρόνος εκτέλεσης στη χειρότερη περίπτωση είναι $O(n^2)$ όπως και της ταξινόμησης με εισαγωγής.

Συνεχίζοντας, ο αλγόριθμος με φυσαλίδα κάνει συνεχώς συγκρίσεις μεταξύ των στοιχείων ανά δύο με πολλά περάσματα, και αντιμεταθέσεις μέχρι να ταξινομηθούν στη σειρά τα στοιχεία και ο πίνακας να είναι ταξινομημένος. Ο αλγόριθμος θεωρείται από τους πιο αργούς

αλγόριθμους εκτός και το μέγεθος του πίνακα είναι μικρό, αλλά είναι πολύ εύκολος στην υλοποίηση του. Όσον αφορά για τον χρόνο εκτέλεσης και αυτός στη χειρότερη περίπτωση είναι $O(n^2)$.

Η ταξινόμηση με συγχώνευση είναι της κατηγορίας της διαίρει και βασίλευε. Κατά την διαδικασία του αλγόριθμου διαιρεί το πίνακα σε δύο υποπίνακες και τους ταξινομεί αναδρομικά το κάθε ένα υποπίνακα και στο τέλος συνδυάζει τους υπόπινακες αυτούς σε ένα πίνακα. Ένα πλεονέκτημα είναι ότι γρήγορος αλγόριθμος σχετικά από άλους αλγόριθμους μιας και έχει χρόνο εκτέλεσης $\Omega(n \log n)$.

Όπως και η ταξινόμησης με συγχώνευση έτσι και ο αλγόριθμος της γρήγορης ανήκει στην κατηγορία του διαίρει και βασίλευε. Κατά την διαδικασία της υλοποίησης, εάν υπάρχει ένα μόνο ένα στοιχείο στον πίνακα τότε δεν κάνει τίποτα διαφορετικά επιλέγεται ένα στοιχείο το pivot. Στη συνέχεια χωρίζει το πίνακα σε δύο μέρη τέτοιο ώστε τα στοιχεία που είναι μικρότερο του pivot να υπάρχουν αριστερά, και τα στοιχείο που είναι μεγαλύτερα του pivot να βρίσκονται δεξιά και στο τέλος ταξινομεί τα δύο μέρη αναδρομικά. Γενικά ο χρόνος εκτέλεσης επηρεάζεται από την επιλογή του στοιχείου pivot. Η χειρότερη περίπτωση $O(n^2)$ είναι όταν τα στοιχεία βρίσκονται είδη ταξινομημένα και το στοιχείο pivot βρίσκεται στην πρώτη θέση του πίνακα διαφορετικά στην μέση περίπτωση θα είναι όταν επιλέξουμε το στοιχείο pivot τυχαία στο πίνακα όπου ο χρόνος εκτέλεσης στην μεσαία περίπτωση είναι $O(n \log n)$. Επίσης ένα πλεονέκτημα είναι ότι είναι γρήγορος αλγόριθμος σε χρόνο και ταχύτητα από την ταξινόμηση της συγχώνευση και από άλλους αλγόριθμους.

Αλγόριθμος	καλύτερης περίπτωσης	χειρότερης περίπτωσης
Ταξινόμηση με εισαγωγή	$O(n)$	$O(n^2)$
Ταξινόμηση με επιλογή	$O(n^2)$	$O(n^2)$
Ταξινόμηση με φυσαλλίδα	$O(n^2)$	$O(n^2)$
Γρήγορη ταξινόμηση	$\Omega(n \log n)$	$O(n^2)$
Ταξινόμηση με συγχώνευση	$O(n \log n)$	$O(n \log n)$

Πίνακας 6.1: *Χρόνος εκτέλεσης αλγόριθμων συγκριτικής ταξινόμησης [8]*

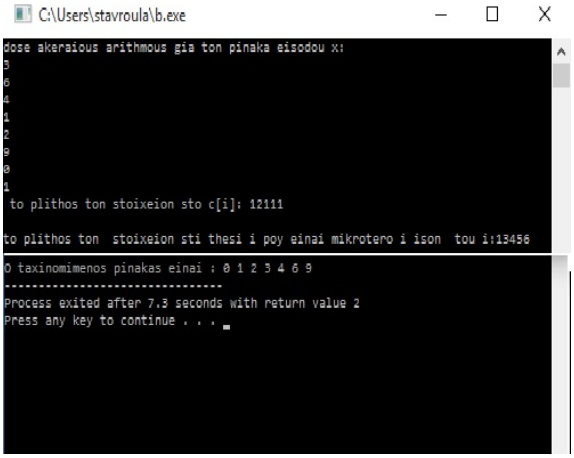
Στον πίνακα 6.2 βλέπουμε τους αλγόριθμους ταξινόμησης γραμμικού χρόνου στην καλύτερη περίπτωση, στην μέση περίπτωση και στην χειρότερη περίπτωση. Συγκρίνοντας τους αλγόριθμους σε γραμμικό χρόνο συμπερένουμε πως η ταξινόμηση με δοχεία έχει αναμένομενο χρόνο γραμμικό $O(n)$ αν τα στοιχεία ακολουθούν ομοιόμορφη κατανομή. Επείσης έχει γραμμικό χρόνο εκτέλεσης όταν δεν υπάρχει ομοιόμορφη κατανομή στα στοιχεία της εισόδου. Στη συνέχεια αναλύοντας τον χρόνο εκτέλεσης της απαριθμιτικής ταξινόμησης συνολικά έχει χρόνο $O(n + k)$. Όμως η απαριθμιτική ταξινόμηση χρησιμοποιείται μόνο όταν $k = O(n)$. Άρα ο χρόνος εκτέλεσης θα είναι $O(n)$. Η αριθμοτακτική ταξινόμηση έχει συνολικό χρόνο εκτέλεση $O(nm)$. Στη περίπτωση όμως που το $d = O(n)$, τότε ο χρόνος εκτέλεσης είναι γραμμικός δηλαδη $O(n)$. [8].

6.2 Υλοποίηση της απαριθμιτική ταξινόμηση

Παρακάτω απεικονίζεται το πρόγραμμα της αριθμοτακτικής ταξινόμησης υλοποιημένο με την γλώσσα προγραμματισμού C (βλπ. 6.1). Στην εικόνα απεικονίζεται το εκτελέσιμο αρχείο με τα αποτελέσματα.

Αλγόριθμος	καλύτερης περίπτωσης	χειρότερης περίπτωσης
Απαριθμιτική Ταξινόμηση	$O(n)$	$O(n)$
Αριθμοτακτική Ταξινόμηση	$O(n)$	$O(n)$
Ταξινόμηση με δοχεία	$O(n)$	$O(n)$

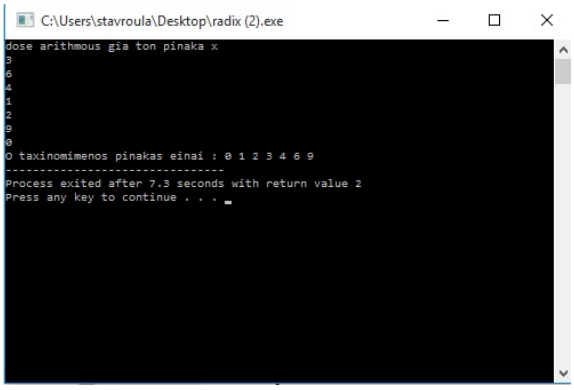
Πίνακας 6.2: *Χρόνος εκτέλεσης αλγόριθμων ταξινόμησης γραμμικού χρόνου [8]*



Εικόνα 6.1: *Αριθμοτακτικής ταξινόμησης*

6.3 Υλοποιήση της αριθμοτακτικής ταξινόμησης

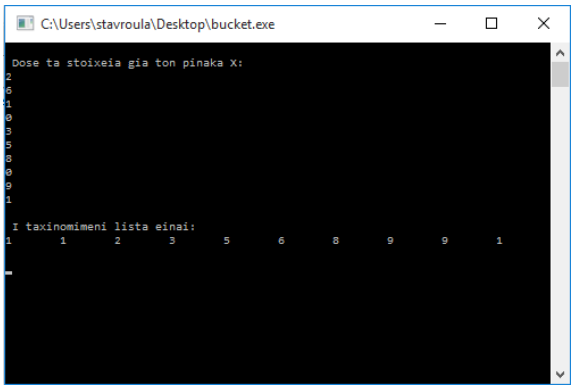
Στο 6.2 βλέπουμε το πρόγραμμα της αριθμοτακτικής ταξινόμησης. Επίσης στην εικόνα 6.2 απεικονίζεται το εκτελέσιμο αρχείο με τα αποτελέσματα της αριθμοτακτικής ταξινόμησης.



Εικόνα 6.2: *Αριθμοτακτικής ταξινόμησης*

6.4 Υλοποίηση της ταξινόμηση με δοχεία

Στο 6.3 βλέπουμε το πρόγραμμα της ταξινόμησης με δοχεία υλοποιημένο στη γλώσσα C. Επίσης στην εικόνα 6.3 απεικονίζεται το εκτελέσιμο αρχείο με τα αποτελέσματα της ταξινόμησης με δοχεία.



Εικόνα 6.3: *Ταξινόμηση με δοχεία*

Αλγοριθμος 6.1: *Πρόγραμμα counting sort [9]*

```
#include<stdlib.h>
#define N 8
int main()
{
  int m=5, x[N], c[m], y[N];
  int i, j;
  {
 printf ("dose akeraious arithmous gia ton pinaka eisodou :");
  for (i=0; i < N; i++)
   scanf ("%d",&x[i]");
   }
   printf ("\n\n");

   for (i=0; i < m; i++)
    c[i]= 0;
   for (j=0; j < N; j++)
  c[x[j]] = c[x[j]] + 1;
  printf ("to plithos ton stoixeion sto c[i]:");
   for (i=0; i < m; i++)
  printf ("%d",c[i]);
   for (i=1; i < m; i++)
   c[i]= c[i] + c[i-1];

  printf ("\n\n to plithos ton stoixeion sto c[i]: ");
   for(i=0; i< m; i++)
   printf ("%d", c[i]);
   printf ("\n\n");
  for (i=(N-1); i > 0; i--)
  {
     x[i]= c[y[i]];
    y[c[x[i-1]]]=x[i]-1;
   c[x[i]]=c[x[i]] -1;
   }
  printf ("o taxinomimenos pinakas einai:");
  for (i = 0; i < N; i++)
  {
   x[i] = y[i];
   printf ("%d", x[i]);
  }
printf ("\n\n");
system ("pause");
return 0;
}
```

Αλγοριθμος 6.2: *Πρόγραμμα radix sort [13]*

```
#include<stdio.h>
#include<stdlib.h>
#define N 7
int main()
{
        int i,m,j, tmp, temp, min, X[N], Y[N];

        printf ("dose arithmous gia ton pinaka x \n");
        for (i=0; i<N; i++)
        {

        scanf ("%d", &X[i]);
        Y[i]=X[i];
  }
   for (m=0; m<6; m++)
   {
        for (i=0; i<N; i++)
        {
                min = X[i] % 10;
                tmp=i;

                for (j=i+1; j<N; j++)
                {
                        if (min > (X[j] % 10 ))
                        {
                          min = X[j] % 10;
                           tmp=j;
                           }
                   }
                   temp = Y[tmp];
                   Y[tmp]= Y[i];
                   Y[i]= temp;
                   temp = X[tmp];
                   X[tmp]= X[i];
                   X[i]= temp;

           }

           for (j=0; j<N; j++)
           X[j]=X[j] / 10;
      }
          printf("O taxinomimenos pinakas einai : ");
   for (i = 0; i < N; i++)
      printf("%d ", Y[i]);
}
```

Αλγοριθμος 6.3: *Πρόγραμμα bucket sort [14]*

```
#include<stdio.h>
#include<stdlib.h>
#define N 10
int main()
{
  int X[50], bucket_sort[10][50]={[0]}, Y[50];
  int j,m,k,p,flaq=0;

  printf ("dose ta stoixeia gia ton pinaka X:\n");
  for (k=0; k<N; k++)
  scanf ("%d",&n);

  Y[k]=X[k]=n;

  for (p=1; flag !=N; p*=10){
  flag=0;

  for (k=0; k<N; k++){
  bucket_sort[(Y[k]/p)%10][k] = Y[k];
  if (Y[k]/p)%10 ==0){
  flag++;
  }

  if (flag==N){
  printf ("\n i taxinomimeni lista einai:\n");

  for (j=0; j<N; j++)
  {
  printf ("%d\t", Y[j]);
  }
  printf ("\n");
  getch();
  return 0;
  }
  for (j=0,m=0; j<10; j++)
  for (k=0; k<N; k++)
  {
  if( bucket_sort[j][k] >0){
  Y[m] = bucket_sort[j][k];
  bucket_sort[j][k]=0;
  m++;
  }
  }
  }
  }
  return 0;
  }
```

Κεφάλαιο 7

Επίλογος

Στο κεφάλαιο αυτό καταγράφονται όλα τα συμπεράσματα που προέκυψαν στην πτυχιακή εργασία.

7.1 Συμπεράσματα

Στην παρούσα πτυχιακή εργασία μελετήθηκαν και παρουσιάστηκαν οι αλγόριθμοι ταξινόμησης γραμμικού χρόνου $O(n)$ στους οποίους μελετήθηκε ο χρόνος εκτέλεσης. Οι αλγόριθμοι ταξινόμησης που μελετήθηκαν ήταν ως εξής: η απαριθμιτική ταξινόμηση (counting sort), η αριθμοτακτική ταξινόμηση (radix sort), και η ταξινόμηση με δοχεία (bucket sort).

Συγκρίνοντας τους αλγόριθμους ταξινόμησης της κατηγορίας των συγκριτικών συμπεραίνουμε ότι ο κατάλληλος αλγόριθμος για αυτούς τους αλγορίθμους είναι της γρήγορης ταξινόμησης με χρόνο εκτέλεσης $O(n \log n)$ μιας και είναι γρήγορος σε σχέση με τους άλλους σε χρόνο και ταχύτητα. Άρα είναι αποδοτικός από τους υπόλοιπους αλγόριθμους συγκριτικής ταξινόμησης.

Επίσης για τους αλγόριθμους της συγκριτικής ταξινόμησης και με την βοήθεια των δέντρων αποφάσεων αποδείχθηκε ότι υπάρχει το κάτω φράγμα για οποιοδήποτε αλγόριθμους που βασίζονται στις συγκρίσεις απαιτώντας $\Omega(n \log n)$ συγκρίσεις στη χειρότερη περίπτωση. Αξίζει να σημειωθεί πως αυτή η περίπτωση είναι η ιδανική για αυτούς τους αλγόριθμους.

Όσον αφορά τους αλγορίθμους γραμμικού χρόνου σε σχέση με τους αλγορίθμους της συγκριτικής ταξινόμησης οι αλγόριθμοι αυτοί δεν χρησιμοποιούν τις συγκρίσεις για τις πράξεις τους αλλά σε άλλες τεχνικές και προέκυψε στο συμπέρασμα ότι είναι γρηγορότεροι από τους συγκριτικούς αλγόριθμους ταξινόμησης σε χρόνο και ταχύτητα λόγω ότι ο χρόνος τους είναι γραμμικός δηλαδή $O(n)$. Οπότε φαίνεται να είναι ταχύτεροι. Αναλύοντας την πολυπλοκότητα των αλγόριθμων προκύπτει ότι η πολυπλοκότητα είναι γραμμική $O(n)$ στη καλύτερη περίπτωση.

Παραρτήματα

Παράρτημα Α΄

ΠΑΡΑΡΤΗΜΑΤΑ

Στο παράρτημα αυτό δίνονται οι βασικές ένοιες των πιθανοτήτων και τις ιδιότητες τους καθώς ορίζονται οι οι τυχαίες μεταβλητές, η αναμενόμενη τιμή και η γραμμικότητα

Α΄.1 Οι πιθανότητες στους αλγόριθμους

Οι πιθανότητες είναι τομέας των μαθηματικών που χρησιμοποιούνται για να αναλύσουν ένα τυχαίο πείραμα όπως για παράδειγμα στην ρίψη ενός νομίσματος για να προκύψει κεφαλή ή κορώνα, στην ρίψη ενός ζαριού, στην γέννηση ενός παιδιού αν προκύψει αγόρι ή κορίτσι κ.ά. Επίσης μπορούν να χρησιμοποιηθούν στους αλγόριθμους. Μέσω της πιθανότητας μπορεί να αναλυθεί ο χρόνος ενός πιθανοτικού αλγορίθμου. Γενικά μία πιθανότητα αποτελεί ενα χώρο που λέγεται δειγματικός χώρος. Ο δειγματικός χώρος περιλαμβάνει το σύνολο όλων των δυνατών αποτελεσμάτων που μπορεί να πάρει ένα τυχαίο πείραμα και συμβολίζεται με κεφαλαία γράματα όπως για παράδειγμα Ω. Το Ω είναι ο δειγματικός χώρος.

Το ενδεχόμενο είναι ένα σύνολο του δειγματικού χώρου Ω που αποτελεί. Για παράδειγμα έστω ότι ρίχνουμε ένα ζάρι και το ενδεχόμενο είναι κατά την ρίψη να τύχει ο αριθμός 4. Ο δειγματικός χώρος αποτελεί όλα τα δυνατά αποτελέσματα που μπορεί να πάρει το ζάρι δηλαδή $\Omega = \{1, 2, 3, 4, 5, 6\}$. Στην περίπτωση αυτή το ενδεχόμενο είναι μετά την ρήψη του ζαριού να τύχει το 4 δηλαδή η πιθανότητα $\frac{1}{6}$. Επίσης κάθε σημείο του δειγματικού χώρου λέγεται δειγματικό στοιχείο. Για παράδειγμα το $1, 2, 3, 4, 5$ και το 6 είναι δειγματικά στοιχεία του δειγματικού χώρου. Όσον αφορά για τον συμβολισμό της πιθανότητας ας υποθέσουμε ότι Ω το δειγματικό χώρο και Α το γεγονός που μπορεί να συμβεί. Τότε η πιθανότητα για να συμβεί είναι $P(A)$. Στην περίπτωση αυτή με την ρίψη του ζαριού η πιθανότητα να τύχει 4 μπορεί να γραφτεί με τον εξής τρόπο: $P(6) = \frac{1}{6}$.

Α΄.2 Ιδιότητες πιθανοτήτων

Παρακάτω υπάρχουν κάποιες ιδιότητες των πιθανοτήτων.

$$P(S) = 1$$

$$p(A) \geq 0$$

$$P\left\{\bigcup_{j=1}^{n} A_j\right\} = \sum_j P(A_j)$$

Αν $P(A)$ είναι η πιθανότητα και Ω ο δειγματικός χώρος τότε το συμπλήρωμα του $P(\overline{A}) = \Omega - P(A)$ [8].

Α΄.3 Τυχαίες μεταβλητες

Στις πιθανότητες μια τυχαία μεταβλητή μπορεί να πάρει το σύνολο όλων των δυνατών τιμών. Μια τυχαία μεταβλητή μπορεί να είναι διακριτή ή συνεχής.

Μια τυχαία μεταβλητή λέγεται διακριτή όταν οι τιμές που παίρνει σε ένα διάστημα είναι πεπερασμένες ή αριθμίσιμες δηλαδή παίρνει συγκεκριμένες τιμές, ενώ μία τυχαία μεταβλητή λέγεται συνεχής όταν παίρνει οποιαδήποτε τιμή σε ένα διάστημα τιμών [8].

Α΄.4 Αναμενομένη τιμή μιας τυχαίας μεταβλητής

Η αναμενόμενη τιμή μιας τυχαίας μεταβλήτης Y ορίζει την μέση τιμή των τιμών της μέτρησης ενός πειράματος το οποίο επαναλαμβάνεται πολλές φορές. Πιο αναλυτικά ας υποθέσουμε ότι έχουμε μία διακριτή τυχαία μεταβλητή Y που παίρνει τιμές y_i όπου $y_i \in N = \{1, 2, 3...\}$, και $N \subset \mathbb{N}$ και $p_j = P(Y = y_j)$ είναι οι πιθανότητες που μπορεί να πάρει οποιαδήποτε τιμή. Η αναμενόμενη τιμή συμβολίζεται με $E(Y)$, και για να υπολογιστεί παίρνουμε το άθροισμα της πιθανότητα της κάθε τιμή επί την τιμή που μπορεί να πάρει. Άρα η αναμενόμενη τιμής της μεταβλητής αυτής θα είναι:

$$E[Y] = \sum_{j \in N} y_j \cdot p_j$$

Για παράδειγμα θέλουμε να υπολογίσουμε την αναμενόμενη τιμή στο ρίξιμο ενός ζαριού. Ο δειγματικός χώρος είναι $\Omega = \{1, 2, 3, 4, 5, 6\}$ και με πιθανότητα $P = P(1) = P(2) = P(3) = P(4) = P(5) = P(6) = \frac{1}{6}$. Άρα η αναμενόμενη τιμή θα είναι θα είναι το άθροισμα του δειγματικού χώρου επί την πιθανότητα για κάθε τιμή. Όποτε

$$E[Y] = 1 \cdot \frac{1}{6} + 2 \cdot \frac{1}{6} + 3 \cdot \frac{1}{6} + 4 \cdot \frac{1}{6} + 5 \cdot 16 + 6 \cdot \frac{1}{6} = 3,5$$

Για τον υπολογισμό της αναμενόμενης τιμής μιας συνεχής τυχαίας μεταβλητής με συνάρτηση πυκνότητα πιθανότητας $f(j)$ δίνεται από τον τύπο:

$$E[Y] = \int j \cdot f(j) dx$$

Στην περίπτωση που έχουμε δύο τυχαίες μεταβλητές X και Y, τότε η αναμενόμενη τιμή των μεταβλητών αυτών ισούται με το άθροισμα των αναμενόμενων τιμών δηλαδή:

$$E[X + Y] = E[X] + E[Y]$$

Η ιδιότητα αυτή λέγεται γραμμικότητα των αναμενόμενων τιμών και ισχύει ακόμη και αν οι δύο μεταβλητές δεν είναι ανεξάρτητες. Στην περίπτωση που οι δύο μεταβλητές είναι

ανεξάρτητες τότε παίρνουμε το γίνομενο των δύων αναμενόμενων τιμών και θα ισχύει ότι:

$$E[XY] = E[X]E[Y]$$

για κάθε $X \neq Y$ και $E[X] = \sum_x xP\{X = x\}$ και $E[Y] = \sum_y yP\{Y = y\}$.

Έστω ότι a είναι μία σταθερά, και X και Y δύο τυχαίες μεταβλητές τότε θα ισχυεί ι ηδιότητα

$$E[aX + Y] = aE[X] + E[Y][8]$$

Α΄.5 Δείκτριες τυχαίων μεταβλητών

Οι δείκτριες τυχαίων μεταβλητών χρησιμοποιούνται πολλές φορές στην πολυπλοκότητα των αλγορίθμων για να αναλύσουμε έναν αλγόριθμο και ο ρόλος τους είναι η μετατροπή της πιθανότητας σε αναμενόμενες τιμές και αντίστροφα. Ας υποθέσουμε ότι έχουμε ένα ζάρι και ο δειγματικός χώρος του ζαριού είναι $\Omega = (1, 2, 3, 4, 5, 6)$ και έστω ότι $P = P(1) = P(2) = P(3) = P(4) = P(5) = P(6) = \frac{1}{6}$, και X_4 ορίζουμε την δείκτρια τυχαία μεταβλητή η οποία αντιστοιχεί στο ενδεχόμενο να έχουμε τον αριθμό 4 στο ζάρι δηλαδή $P(4)$. Η τιμή αυτή ισούται με την πιθανότητα να φέρει 4 το ζάρι, δηλαδή 1 αν φέρει $P(4)$, διαφορετικά $1 - P(4)$. Άρα

$$X_4 = I4 = \begin{cases} 1 & P(4) \\ 0 & 1 - P(4) \end{cases}$$

$E[X_k] = E[I(4)] = 1 \cdot P(4) + 0 \cdot P(4) = 1 \cdot \frac{1}{6} + 0 \cdot (1 - \frac{1}{6}) = \frac{1}{6}$

Άρα ο αναμενόμενος αριθμός να φέρει το ζάρι 4 σε μία ρίψη ενός ζαριού είναι $\frac{1}{6}$. Επομένως η αναμενόμενη τιμή μιας δείκτριας μεταβλητής που αντιστοιχεί σε ένα ενδεχόμενο ισούται με την πιθανότητα να συμβεί το Α [8].

Βιβλιογραφία

[1] *Counting sort*. http://en.wikipedia.org/wiki/Counting_sort.

[2] *Αλγόριθμοι και Δομές Δεδομένων*. http://infoman.teikav.edu.gr/e_education/67/files/alg%20eis.pdf.

[3] *Αλγόριθμος*. https://el.wikipedia.org/wiki/.

[4] *Γρήγορη Ταξινόμηση*. http://el.wikipedia.org/wiki/.

[5] *Ταξινόμηση με συγχώνευση*. http://el.wikipedia.org/wiki/.

[6] *Bucket sort*. https://en.wikipedia.org/wiki/Bucket_sort.

[7] *Radix sort*. https://en.wikipedia.org/wiki/Radix_sort.

[8] Ronald L. Rivest Clifford Stein Thomas H. Cormen, Charles E. Leiserson. *Εισαγωγή στους Αλγορίθμους*. 2013.

[9] Ι. Παπουτσής. *Εισαγωγή στις Δομές Δεδομένων και στους Αλγόριθμους, Υλοποιήση σε C*. ΑΘ. ΣΤΑΜΟΥΛΗΣ, Αθήνα, 2010.

[10] Δ. Φωτάκης. *Ασυμπτωτικός Συμβολισμός*. 2012.

[11] *Κάτω φράγμα στο χρόνο ταξινόμησης*. http://www.softlab.ntua.gr/~fotakis/data_structures/LowerBound4.pdf.

[12] *Sorting (Part II) CSE 373 Data structures Unit 17*. http://courses.cs.washington.edu/courses/cse373/03au/lectures/sort2.pdf.

[13] *C Program for radix sort*. http://www.sanfoundry.com/c-program-lsdradix-sort-algorithm/.

[14] *C Program for Bucket sort*. http://www.dit.hua.gr/~michail/teaching/algs/slides/020_Analysis.pdf.

[15] *Ανάλυση αλγορίθμων*. http://www.dit.hua.gr/~michail/teaching/algs/slides/020_Analysis.pdf.

Συντομογραφίες - Αρκτικόλεξα - Ακρωνύμια

βλπ.	βλέπε
κ.λπ.	και λοιπά
κ.ά.	και άλλα
κ.ο.κ	και ούτω καθεξής
ΤΕΙ	Τεχνολογικό Εκπαιδευτικό Ίδρυμα

Απόδοση ξενόγλωσσων όρων

Απόδοση	Ξενόγλωσσος όρος
χρόνος εκτέλεσης	execution time
ταξινόμηση με εισαγωγή	insertion sort
ταξινόμηση με επιλογή	selection sort
ταξιμόμηση με φυσαλλίδα	bubble sort
ταξινόμηση με συγχώνευση	merge sort
γρήγορη ταξινόμηση	quick sort
απαριθμιτική Ταξινόμηση	counting sort
ταξινόμηση με δοχεία	bucket sort
γραμμικός χρόνος	time linear
ευσταθής	stable
συγκριτικοί αλγόριθμοι ταξινόμησης	sorting algorithms comparative
μη συγκριτικοί	non comparisons

Ευρετήριο ελληνικών όρων

Ευρετήριο ξενόγλωσσων όρων