2023

Εργασία 3

ΜΕΛΕΤΗ ΣΤΟΥΣ ΑΛΓΟΡΙΘΜΟΥΣ ΤΑΞΙΝΟΜΗΣΗΣ

ΔΗΜΗΤΡΗΣ ΧΑΡΙΣΤΕΣ

ΑΕΜ: 604-2031

Δευτέρα | 16/01/2023

Δημήτρης Χαριστές
604-2031
16/01/2023

Abstract

Σε αυτήν την εργασία θα μελετηθεί ο χρόνος εκτέλεσης τριών βασικών αλγορίθμων ταξινόμησης(Insertion Sort, Selection Sort, Quick Sort). Για την διεξαγωγή αυτής της μελέτης θα γίνει συγγραφή ενός προγράμματος σε γλώσσα C όπου οι αλγόριθμοι θα ταξινομούν αριθμητικούς πίνακες με διαστάσεις από 10.000 έως και 100.000 θέσεις και βήμα 10.000. Επίσης για κάθε διάσταση θα καταγράφεται ο χρόνος που χρειάστηκε κάθε αλγόριθμος για την ταξινόμηση του πίνακα. Τα αποτελέσματα της ερευνάς θα παραστούν σε πίνακα μαζί με το αντίστοιχο γράφημα και τις προσεγγιστικές συναρτήσεις χρόνου- διάστασης που ακολουθεί ο κάθε αλγόριθμος, T(n)=t. Επίσης θα γίνει μελέτη της πολυπλοκότητας του κάθε αλγορίθμου.

ΥΛΟΠΟΙΗΣΗ ΠΡΟΓΡΑΜΜΑΤΟΣ ΣΕ C

Για την υλοποίηση του προγράμματος αρχικά εισάγεται ο κώδικας των τριών αλγορίθμων μαζί μετά ορίσματα τους στην αρχή του προγράμματος. Στην συνέχεια δημιουργείτε η συνάρτηση main() στην οποία θα τρέξουν οι αλγόριθμοι και θα γίνει η συλλογή των δεδομένων. Επίσης κάνουμε συμπερίληψη των απαραίτητων βιβλιοθηκών .h. Αναφορικά έχουμε: stdio.h, stdlib.h, time.h, string.h.

Ξεκινώντας την συγγραφή του κώδικά στην main() κάνουμε initialize τις μεταβλητές και τους βοηθητικούς πίνακες που θα χρειαστούμε. Αναφορικά: πίνακας sum[3] κάθε μια από τις 3 θέσεις αρχικοποιούνται σε μηδέν. Ο πίνακας αυτός θα χρησιμοποιηθεί για την εύρεση του μέσου όρου εκτέλεσης κάθε αλγορίθμου σε κάθε διάσταση αποθηκεύοντας το άθροισμα των χρόνων εκτέλεσης (οπού η θέση 0 θα είναι για τον Insertion Sort, θεση1 Selection Sort και θέση 2 Quick Sort). Ο πίνακας avg[3] θα αποθηκεύει τον μέσο ορό εκτέλεσης κάθε αλγορίθμου για κάθε διάσταση n. Οι μεταβλητές τύπου clock_t (start, end) θα χρησιμοποιηθούν για τον υπολογισμό και την χρονομέτρηση της εκτέλεσης του κάθε αλγορίθμου. Αυτή η χρονομέτρηση θα αποθηκεύεται προσωρινά (μέχρι την επόμενη εκτέλεση δηλαδή) στην μεταβλητή cpu_time_used. Οι δείκτες a, b θα χρησιμοποιηθούν για την δυναμική αρχικοποίηση του πίνακα a τον οποίο θα ταξινομεί κάθε φορά ο κάθε αλγόριθμος και ομοίως ο δείκτης b θα χρησιμοποιηθεί για τον βοηθητικό πίνακα b. Θα αναφερθούμε παρακάτω για την χρησιμότητα του καθενός αναλυτικότερα.

Αρχικά φτιάχνουμε ένα for loop με βήμα ένα για το k και συνθήκη τερματισμού k<10. Στην πρώτη γραμμή μέσα στο for θα χρησιμοποιήσουμε την μεταβλητή n η οποία θα αποτελεί το μέγεθος του πίνακα που ταξινομούμε κάθε φορά(έχει αρχικοποιηθεί σε 0 στην αρχή της main()). Το n θα αυξάνεται κατά 10.000 σε κάθε επανάληψη μαζί με το k που θα αυξάνεται κατά 1. Οπότε όταν το n φτάσει στο 100.000 το loop θα σταματήσει καθώς το k θα έχει γίνει 10. Οπότε επιτυγχάνουμε το επιθυμητό αποτέλεσμα του βήματος του n να είναι 10.000 με τελευταίο μέγεθος το 100.000.

Στην συνέχεια θα αρχικοποιήσουμε τον δυναμικό πίνακα a. Χρησιμοποιώντας την malloc() αρχικοποιούμε τον πίνακα με θέσεις όσες είναι το n κάθε φορά. Επίσης αρχικοποιείται ένας βοηθητικός πίνακας με ακριβώς τον ίδιο τρόπο. Η ανάγκη του θα αναφερθεί παρακάτω. Με ένα for loop αρχικοποιούμε την τιμή κάθε θέσης του πίνακα sum[] σε μηδέν διότι μετά από κάθε επανάληψη δε θέλουμε να κρατάει τα προηγούμενα αθροίσματα καθώς η διάσταση του πίνακα αλλάζει. Κλείνουμε αυτό το for loop και συνεχίζουμε.

Για να παίρνουμε σωστά αποτελέσματα χρόνου εκτέλεσης από τους αλγορίθμους μας θα βρίσκουμε τον μέσο χρόνο εκτέλεσης σε κάθε διάσταση από 10 ταξινομήσεις του πίνακα a ο οποίος θα έχει διαφορετικούς αριθμούς κάθε φορά (πχ. Για την διάσταση των 10.000 ο Insertion sort θα κάνει 10 ταξινομήσεις αριθμητικών πινάκων και από εκεί θα πάρουμε τον μέσο χρόνο ταξινόμησης πίνακα 10.000 θέσεων). Οπότε λοιπόν φτιάχνουμε ένα for loop της μορφής (i=0 i<10 i++) ώστε να συλλέξουμε 10 στιγμιότυπα χρόνων εκτέλεσης πριν το loop τερματιστεί. Εντός αυτού του for δίνουμε με ένα μικρό for loop(j=0 j<n j++) random τιμές σε κάθε θέση του πίνακα a ενώ κάθε τιμή την περνάμε και στον πίνακα b (πρακτικά ο πίνακας b είναι αντίγραφο του a). Κλείνουμε αυτό το for loop και συνεχίζουμε.

Ανοίγουμε ένα for loop καθώς ο κώδικας που θα χρησιμοποιήσουμε για την χρονομέτρηση και την αποθήκευση του αθροίσματος των χρόνων για τον μετέπειτα υπολογισμό του μέσου ορού είναι ίδιος και για τους τρεις

αλγορίθμους. Εφόσον έχουμε τρεις αλγορίθμους η συνθήκη τερματισμού θα είναι l<3 με βήμα ένα. Εντός του for αυτού κάνουμε start το clock(καθώς ακολουθεί η εκτέλεση του αλγορίθμου) και με ένα switch(l) - case επιλέγουμε ποιος από τους τρεις αλγορίθμους θα τρέχει σε κάθε επανάληψη (για l=0 τρέχει ο insertion sort, για l=1 ο Selection Sort κλπ.). Έξω από switch-case κάνουμε end clock(). Υπολογίζουμε - αποθηκεύουμε το χρονικό αποτέλεσμα στην μεταβλητή cpu_time_used[l]. Αυτή η μεταβλητή είναι πίνακας όπως και η start-end για να ξεχωρίζουν για κάθε αλγόριθμο. Πχ. Εφόσον για l=0 θα τρέξει ο Insertion sort θα έχουμε το start-end[0] και το cpu_time_used[0] αντίστοιχα. Αποθηκεύουμε τον χρόνο cpu_time_used[l] στο counter sum[l] οπού κάθε φορά γίνεται η πράξη sum[l] + cpu_time_used[l] που σημαίνει ότι αθροίζει τελικά όλους του χρόνους των 10 εκτελέσεων του κάθε αλγορίθμου για κάθε διάσταση. Η sum[] είναι επίσης πίνακας ώστε να ξεχωρίζονται τα αθροίσματα κάθε αλγορίθμου.

Επειδή θέλουμε για κάθε στιγμιότυπο (από τα 10 που θα πάρουμε για κάθε αλγόριθμο) ο πίνακας που θα ταξινομεί κάθε αλγόριθμος να είναι ο ίδιος αλλιώς δε παίρνουμε σωστά αποτελέσματα. Κάτω λοιπόν από τον υπολογισμό του sum θα μεταβιβάζουμε στον a πίνακα το αντίγραφο b που δημιουργήσαμε έτσι ώστε κάθε φορά οι random τιμές που δίνονται στο πίνακα a να περνάνε και στον b πριν αυτός ταξινομηθεί ώστε να μπορέσουμε να τον χρησιμοποιήσουμε για την αντίστροφη διαδικασία όταν ο a έχει πλέον ταξινομηθεί. Με αυτόν τον τρόπο όταν το loop τρέξει πχ. στις 10.000 στο 2ο στιγμιότυπο και για l=1 ο Selection sort θα ταξινομήσει τον ίδιο πίνακα που ταξινόμησε και ο Insertion sort και όχι τον ήδη ταξινομημένο πίνακα που θα "ταξινομούσε" αν δεν κάναμε re-assign στον πίνακα a το αντίγραφο του, b[]. Με αυτή γραμμή κώδικα κλείνουμε το loop του switch-case οπού γίνετε η χρονομέτρηση και επιτόπου κλείνουμε και το από πάνω loop που δημιουργήθηκε για την λήψη των δέκα στιγμιότυπων.

Επανερχόμαστε στο αρχικό loop οπού μένει να υπολογίσουμε τον μέσο χρόνο ταξινόμησης του πίνακα για την διάσταση που μόλις έτρεξαν οι αλγόριθμοι δέκα φορές και να εκτυπώσουμε το αποτέλεσμα και για του τρεις αλγορίθμους σε αυτή την διάσταση.

Με ένα for loop βήματος 1 και συνθήκη τερματισμού l<3 υπολογίζουμε τον μέσο ορό avg[l] (όπου το l κάθε φορά αντιστοιχεί σε έναν από τους τρεις αλγόριθμους για τιμές 0, 1, 2) κάνοντας την πράξη sum[l]/10. Από κάτω εκτυπώνουμε το αποτέλεσμα με το μήνυμα: Αλγόριθμος: Δίνετε από την συνάρτηση choice(int l) όπου για κάθε l επιλέγεται αντίστοιχα το αλφαριθμητικό: Insertion Sort, Selection Sort, Quick Sort - Διάσταση: n – Μέσος χρόνος εκτέλεσης: avg[l]. Τέλος γίνετε αποδέσμευση των πινάκων a, b από τις θέσεις που δεσμεύονται κάθε φορά για κάθε διάσταση με την συνάρτηση free(). Κλείνει και το αρχικό for loop.

Επίσης στην αρχή του κώδικα, πριν το πρώτο for loop (που ανεβάζει σε κάθε επανάληψη την διάσταση του πίνακα με το βήμα 10.000), ξεκίνησε ένα start clock() με σκοπό την καταγραφή του χρόνου εκτέλεσης του προγράμματος για περαιτέρω βελτίωση στον συνολικό χρόνο εκτέλεσης του. Στο τέλος λοιπόν του προγράμματος βάζουμε το end clock(). Αποθηκεύουμε σε μια μεταβλητή cpu_time used_p την αφαίρεση του start από το end επί CLOCKS_PER_SEC και εκτυπώνουμε τον χρόνο εκτέλεσης του προγράμματος σε δευτερόλεπτα.

ΜΕΛΕΤΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ

Η εκτέλεση του προγράμματος ολοκληρώθηκε σε 7 λεπτά και 35 δευτερόλεπτα δίνοντας τους μέσους χρόνους ταξινόμησης των τριών αλγορίθμων για διαστάσεις από 10.000-100.000 και βήμα 10.000. Επίσης δίνετε ο μέσος χρόνος εκτέλεσης κάθε αλγορίθμου συνολικά.

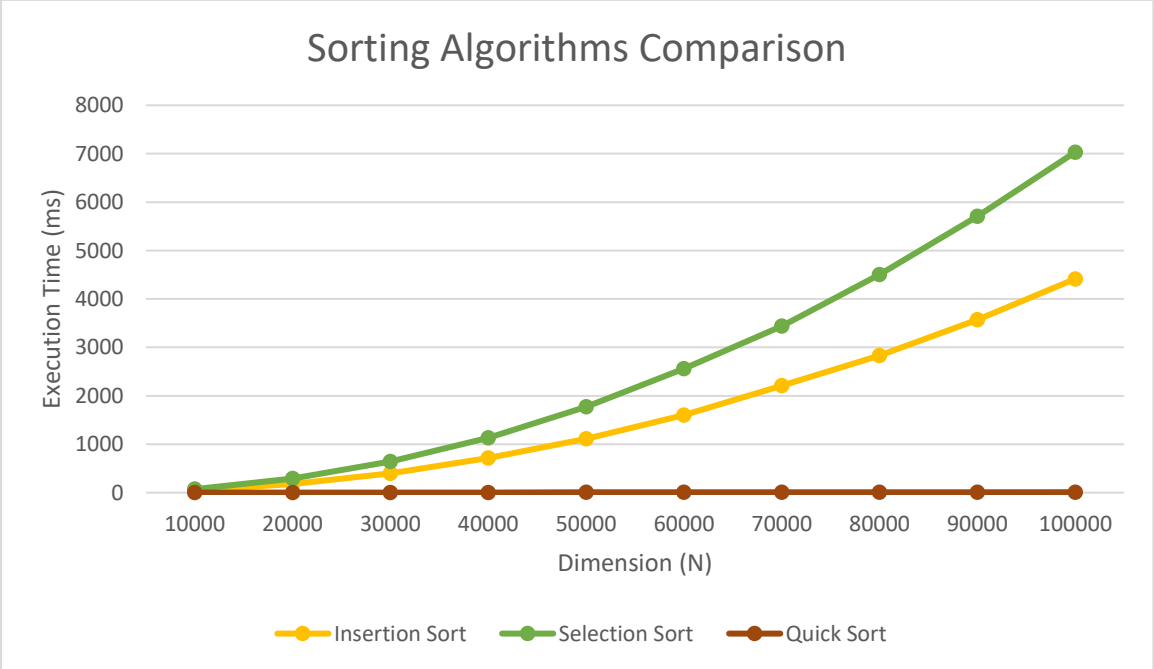
Στον παρακάτω πίνακα φαίνονται για κάθε αλγόριθμο οι μεσοί χρόνοι ταξινόμησης σε κάθε διάσταση. Ο χρόνος είναι σε ms καθώς τα αποτελέσματα δε θα ήταν αρκετά ευδιάκριτα στο γράφημα. Είναι φανερό και επιβεβαιώνει την θεωρία ότι ο Quick sort σε αντίθεση με τους άλλους δυο αλγορίθμους ταξινομεί τον πίνακα κάθε διάστασης τάξεις γρηγορότερα και αποδοτικότερα. Σε αποδοτικότητα ακολουθεί ο Insertion sort και τελευταίος ο Selection sort. Σύμφωνα με την θεωρία ο Insertion sort έχει μέση πολυπλοκότητα $\Theta(n^2)$, ο Selection sort $\Theta(n^2)$ και ο Quick sort Θ(nlogn). Μένει να μελετηθεί η πολυπλοκότητα με τα πειραματικά δεδομένα του πίνακα και το γράφημα για εξακρίβωση.

Δημήτρης Χαριστές
604-2031
16/01/2023

Algorithms / Dimension	Insertion Sort (ms)	Selection Sort (ms)	Quick Sort (ms)
10.000	45	71.9	0.8
20.000	179.4	288.2	1.4
30.000	398.1	639.8	2.2
40.000	712.4	1134.1	3.2
50.000	1113.0	1772.8	3.9
60.000	1599.6	2560.7	4.9
70.000	2208.8	3617.3	5.4
80.000	2827.3	4500.7	6.3
90.000	3574.1	5703.4	7.1
100.000	4413.6	7031.3	8.0
Average Execution Time	1707.13	2732.02	4.32

Πίνακας 1: Ο Insertion Sort παρουσιάζει πολυπλοκότητα $\Theta(n^2)$, ο Selection Sort $\Theta(n^2)$ και ο Quick Sort $\Theta(n)$.

Σύμφωνα με τον παραπάνω πίνακα δίνετε το γράφημα Διάστασης(n) - Χρόνου(ms) για κάθε αλγόριθμο.



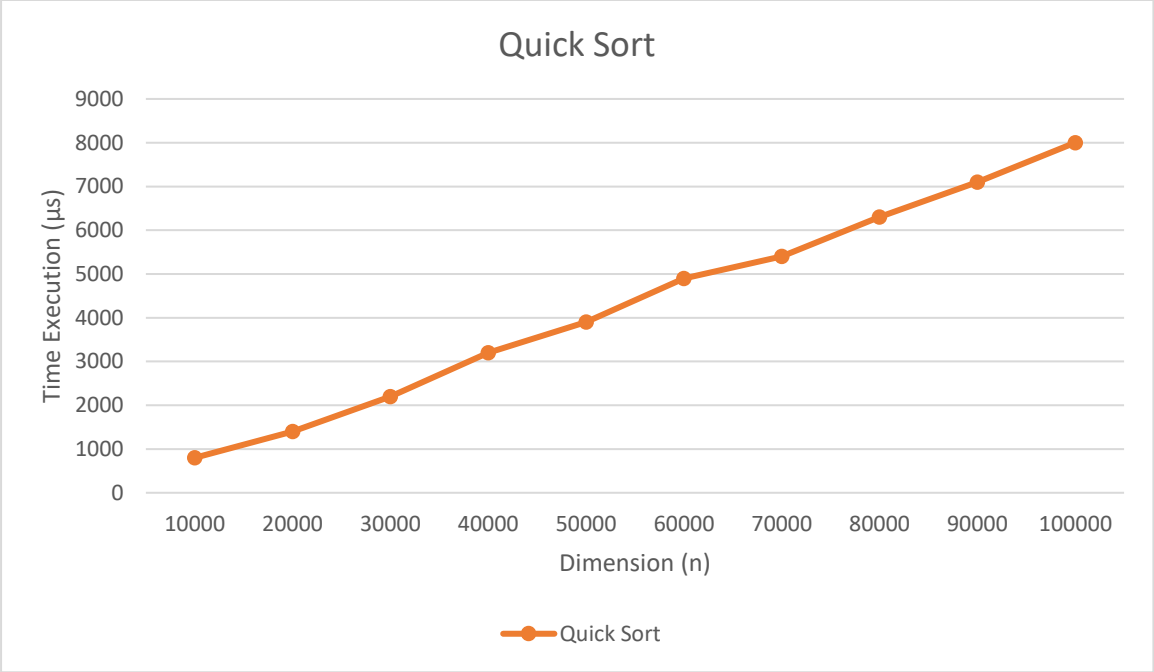
Γράφημα 1:Η καμπύλη των Insertion Sort και Selection sort υποδεικνύει ότι έχουν πολυπλοκότητα $\Theta(n^2)$. Όσο για την Quick Sort δίνετε ευδιάκριτο διάγραμμα στο Γράφημα 2.

Εκτίμηση πολυπλοκότητας των αλγορίθμων

Από τα δεδομένα του πίνακα βρίσκουμε ότι η Insertion Sort έχει μέση πολυπλοκότητα $\Theta(n^2)$. Για διάσταση 10.000 ο χρόνος είναι 45 ms και για διπλάσια διάσταση 20.000 ο χρόνος σχεδόν τετραπλασιάζεται σε 179.4 ~ 4·45. Αυτή η εκτίμηση συμφωνεί και με τα πειραματικά δεδομένα των άλλων διαστάσεων.

Δημήτρης Χαριστές
604-2031
16/01/2023

Παρόμοια για την πολυπλοκότητα της Selection Sort. Για διάσταση 20.000 ο χρόνος είναι 288,2 ms και για διπλάσια διάσταση 40.000 ο χρόνος γίνεται 1134,1 ms που είναι σχεδόν τετραπλάσιος με σφάλμα +18.7 ms.



Γράφημα 2: Η γραμμή του Quick sort υποδεικνύει πολυπλοκότητα Θ(n) και όχι Θ(nlogn)

Όσο για τον Quick sort παρατηρούμε τόσο από τον πίνακα όσο και από το Γράφημα 2 ότι ο χρόνος δεν φαίνεται να αυξάνεται γραμμολογαριθμικά συναρτήσει της διάστασης. Αντιθέτως παρατηρούμε ότι αυξάνεται με βήμα 0,8±0,2. Οπότε δεν μπορεί να έχει πολυπλοκότητα Θ(nlogn). Τείνει σε μια γραμμική αύξηση του χρόνου εκτέλεσης. Δηλαδή για διπλάσια διάσταση ο χρόνος διπλασιάζεται. Για n=10.000 ο χρόνος είναι 0,8ms και για 40.000 ο χρόνος γίνεται 3.2 που σημαίνει ότι τετραπλασιάστηκε όπως και η διάσταση. Παρ' ολ' αυτά για τις υπόλοιπες διαστάσεις έχουμε ένα σφάλμα e (θα υπολογιστεί παρακάτω). Οπότε είναι πολυπλοκότητας Θ(n); Παρότι τείνει περισσότερο σε γραμμική συνάρτηση θα μπορούσε να παραστεί προσεγγιστικά και από γραμμολογαριθμική συνάρτηση αν και με μεγαλύτερο σφάλμα. Προκύπτει, όπως θα φανεί παρακάτω με την βοήθεια του desmos, μια γραμμολογαριθμικη συνάρτηση για τον χρόνο του Quick sort. Δηλαδή προτείνεται και πολυπλοκότητα Θ(nlogn).

Υπολογισμός προσεγγιστικών συναρτήσεων – Χρήση Desmos

Με την βοήθεια του desmos βρίσκουμε τις προσεγγιστικές συναρτήσεις για κάθε αλγόριθμο. Εισάγουμε τον πίνακα στο desmos και καθώς ο Insertion sort έχει πολυπλοκότητα $\Theta(n^2)$ υποθέτουμε ότι η προσεγγιστική συνάρτηση του θα είναι της μορφής $T(n)=c\cdot n^2$. Από το desmos λοιπόν βρίσκουμε ότι το c θα είναι $44{,}3\cdot10^{-8}$ άρα: $\mathbf{t_{insort} \sim 44{,}3\cdot10^{-8}\cdot n^2\ [ms]}$.

Παρόμοια υποθέτουμε ότι ο Selection Sort θα έχει προσεγγιστική συνάρτηση της μορφής $T(n)=c\cdot n^2$ καθώς έχει μέση πολυπλοκότητα $\Theta(n^2)$. Οπότε βρίσκουμε $c=70{,}4\cdot10^{-8}$ άρα: $\mathbf{t_{selsort} \sim 70{,}4\cdot10^{-8}\cdot n^2\ [ms]}$. Εισάγοντας έναν δείκτη σφάλματος που προκύπτει από τον μέσο χρόνο εκτέλεσης του αλγορίθμου σύμφωνα με την συνάρτηση ($t_{selsort\_avg} = 2710.4ms$) και τα πειραματικά δεδομένα ($\tau = 2732{,}02$) έχουμε: $e = \frac{2710{,}4-2732.02}{2710{,}4} \times 100\% = \frac{-21{,}6}{2710{,}4} \times 100\% \cong 0{,}8\%$.

Δημήτρης Χαριστές
604-2031
16/01/2023

Υποθέτουμε για τον Quick Sort ότι έχει προσεγγιστική συνάρτηση της μορφής T(n)=c·n καθώς έχει μέση πολυπλοκότητα Θ(n). Οπότε βρίσκουμε $c=0,79\cdot10^{-8}$ άρα: $\mathbf{t_{qsort} \sim 0,79\cdot10^{-8} \cdot n\ [ms]}$. Με δείκτη σφάλματος $e = \frac{4,345-4,32}{4,345} \times 100\% = \frac{0,025}{4,345} \times 100\% \cong 0,58\%$. Όπως αναφέρεται στην προηγούμενη υποενότητα, προτείνεται αν και με μικρότερη ακρίβεια, πολυπλοκότητα Θ(nlogn). Οπότε η προσεγγιστική συνάρτηση για τον χρόνο εκτέλεσης της Quick sort θα μπορούσε να δίνεται και από την γραμμολογαριθμικη T(n)=c·n·log(n). Αυτό προκύπτει υποθέτοντας ότι ο Quick sort παρουσιάζει πολυπλοκότητα Θ(nlogn) και αρά θα ακολουθεί μια προσεγγιστική συνάρτηση με τύπο: T(n)=c·n·log(n). Με την βοήθεια του desmos λοιπόν βρίσκουμε το c=0.016 και άρα $\mathbf{t_{qsort} \sim 0,016 \cdot n \cdot log(n)\ [\mu s]}$.

ΣΥΜΠΕΡΑΣΜΑΤΑ

Το πρόγραμμα που γράφτηκε σε C για την διεξαγωγή της μελέτης έδωσε σωστά πειραματικά αποτελέσματα. Από την μελέτη λοιπόν συμπεραίνουμε ότι ο Quick Sort είναι ο πιο αποδοτικός από τους τρεις αλγορίθμους όπως υποδεικνύει και η εγκεκριμένη βιβλιογραφία - θεωρία. Επίσης επιβεβαιώσαμε ότι η μέση πολυπλοκότητα των Insertion sort και Selection Sort συμφωνεί με την βιβλιογραφία καθώς είναι $\Theta(n^2)$. Παρ' ολ' αυτά στον Quick Sort βλέπουμε ότι προσεγγίζει περισσότερο μια γραμμική πολυπλοκότητα από τι γραμμολογαριθμική που σημαίνει ότι η μέση περίπτωση που βρήκαμε είναι καλύτερη από την μέση περίπτωση που ξέρουμε από την θεωρία.

Δημήτρης Χαριστές
604-2031
16/01/2023

Βιβλιογραφία:

Sedgewick, R. , *Αλγόριθμοι σε C*
Πλόσκας, Ν. , *Σημειώσεις στου Αλγόριθμους.* Πανεπιστήμιο Δυτικής Μακεδονίας | ΤΗΜΜΥ
Desmos Graphing Calculator, Διαθέσιμο στον ισότοπο: *https://www.desmos.com/calculator*