Αναφορά Δομές Δεδομένων 1η Εργασία

Αλέξια Σούλα , Ιωάννης Μπουρίτης

Πολυπλοκότητα των αλγορίθμων αναζήτησης

	1a	1b	2a	2b	3a	3b
insert()	O(1)	O(n)	O(1)	O(n)	O(1)	O(n)
delete()	O(n)	O(n)	O(n)	O(n)	O(n)	O(n)
search()	O(n)	O(n)	O(n)	O(n)	O(n)	O(log n)

- Υλοποίηση 1a:
  - insert(): Η εισαγωγή ενός νέου στοιχείου στο τέλος της λίστας απαιτεί σταθερό χρόνο, ανεξαρτήτως του μεγέθους της λίστας.
  - delete(): Η διαγραφή ενός στοιχείου απαιτεί γραμμικό χρόνο, καθώς πρέπει να βρεθεί το επιθυμητό στοιχείο προτού διαγραφεί.
  - search(): Η αναζήτηση ενός στοιχείου μέσα στη λίστα απαιτεί γραμμικό χρόνο, αφού πρέπει να εξεταστούν όλα τα στοιχεία.

- Υλοποιηση 1b:
  - insert(): Η εισαγωγή ενός νέου στοιχείου σε μια ταξινομημένη λίστα μπορεί να απαιτήσει γραμμικό χρόνο στη χειρότερη περίπτωση, αν χρειαστεί να γίνει πλήρης επαναταξινόμηση της λίστας. Ωστόσο, σε καταστάσεις όπου η λίστα είναι ήδη ταξινομημένη, η εισαγωγή μπορεί να γίνει σε σταθερό χρόνο.
  - delete(): Η διαγραφή ενός στοιχείου απαιτεί γραμμικό χρόνο, αναλόγως τη θέση του στοιχείου στη λίστα και την ανάγκη για επαναταξινόμηση.
  - search(): Η αναζήτηση σε μια ταξινομημένη λίστα απαιτεί γραμμικό χρόνο, όπως και στη μη ταξινομημένη περίπτωση.
- Υλοποιηση 2a:
  - insert(): Η εισαγωγή ενός νέου στοιχείου γίνεται σε σταθερό χρόνο, καθώς χρησιμοποιούμε μια στοίβα ελεύθερων γραμμών για να εντοπίσουμε γρήγορα την επόμενη διαθέσιμη θέση.
  - delete(): Η διαγραφή ενός στοιχείου απαιτεί γραμμικό χρόνο, καθώς πρέπει να διατρέξουμε τις γραμμές του πίνακα για να το εντοπισουμε και να ενημερώσουμε τις δείκτες στην αντίστοιχη γραμμή για τη διαγραφή του.
  - search(): Η αναζήτηση ενός στοιχείου μέσα στον πίνακα απαιτεί γραμμικό χρόνο, καθώς πρέπει να διατρέξουμε τις γραμμές του πίνακα για να το βρούμε.

- Υλοποιηση 2b:
  - insert(): Η εισαγωγή ενός νέου στοιχείου σε μια ταξινομημένη λίστα προϋποθέτει τη διάταξη των στοιχείων σε σωστή σειρά, άρα η πολυπλοκότητα είναι O(n), όπου n είναι το πλήθος των στοιχείων στη λίστα.
  - delete():Η διαγραφή ενός στοιχείου απαιτεί γραμμικό χρόνο, αναλόγως τη θέση του στοιχείου στη λίστα και την ανάγκη για ενημέρωση των δεικτών στον πίνακα.
  - search():Η αναζήτηση ενός στοιχείου σε μια ταξινομημένη λίστα απαιτεί γραμμικό χρόνο, καθώς πρέπει να διατρέξουμε τα στοιχεία για να το βρούμε.

- Υλοποιηση 3a:
  - insert(): Η εισαγωγή ενός νέου στοιχείου γίνεται σε σταθερό χρόνο, καθώς δεν απαιτείται μετακίνηση άλλων στοιχείων.
  - delete(): Η διαγραφή ενός στοιχείου απαιτεί γραμμικό χρόνο, καθώς όλα τα στοιχεία που βρίσκονται δεξιά από το διαγραφόμενο πρέπει να μετακινηθούν κατά μία θέση προς τα αριστερά για να καλυφθεί το κενό.
  - search(): Η αναζήτηση ενός στοιχείου μέσα στον πίνακα απαιτεί γραμμικό χρόνο, καθώς πρέπει να διατρέξουμε όλα τα στοιχεία.

- Υλοποιηση 3b:
  - insert(): Η εισαγωγή ενός νέου στοιχείου σε μια ταξινομημένη λίστα απαιτεί δυαδική αναζήτηση για την εύρεση της σωστής θέσης. Εάν η θέση εισαγωγής είναι στο ενδιάμεσο, απαιτείται μετακίνηση όλων των στοιχείων που ακολουθούν μετά τη θέση εισαγωγής, με αποτέλεσμα γραμμική πολυπλοκότητα.
  - delete(): Η διαγραφή ενός στοιχείου απαιτεί δυαδική αναζήτηση για την εύρεση του στοιχείου προς διαγραφή. Εάν το στοιχείο βρίσκεται στο ενδιάμεσο, όλα τα στοιχεία που βρίσκονται δεξιά από αυτό πρέπει να μετακινηθούν κατά μία θέση αριστερά, με αποτέλεσμα γραμμική πολυπλοκότητα.
  - search(): Η αναζήτηση ενός στοιχείου μέσα σε μια ταξινομημένη λίστα γίνεται με δυαδική αναζήτηση, που έχει πολυπλοκότητα O(log n).

Απαντήστε σύντομα στις παρακάτω ερωτήσεις/θέματα

1). Χρόνοι εκτέλεσης με Πλήθος πράξεων

Σημαντικό:

Κατανοούμε ότι πιθανότατα να εχουν υπάρξει λάθη κατά την εκτύπωση των τιμών χρόνου και πράξεων τα οποία συνεπάγονται σε λάθη στους μεσους ορους. Κατά συνέπεια, τα γραφήματα δεν έχουν τη μορφή που θα έπρεπε, με βάση τη θεωρητική προσέγγιση. Στα διαγράμματα χρόνου, το 2α θα έπρεπε να είναι μια σταθερή γραμμή καθώς έχει O(1) πολυπλοκότητα, το 2β θα έπρεπε να αυξάνεται γραμμικά και το 3β θα έπρεπε να είναι

λογαριθμικό, έτσι ώστε να μην υπάρχει μεγάλη διαφορά στον χρόνο όσο και αν αυξάνεται το N. Θα έπρεπε να καταλήξουμε στο συμπέρασμα ότι η 3β υλοποίηση είναι η πιο γρήγορη λόγω του binary search και ως επέκταση της πολυπλοκότητας O(log n).

2). Μοναδικά Κλειδιά

	1a	1b	2a	2b	3a	3b
insert()	O(1)	O(n)	O(1)	O(n)	O(1)	O(n)
delete()	O(n)	O(n)	O(n)	O(n)	O(n)	O(n)
search()	O(n)	O(n)	O(n)	O(n)	O(n)	O(log n)

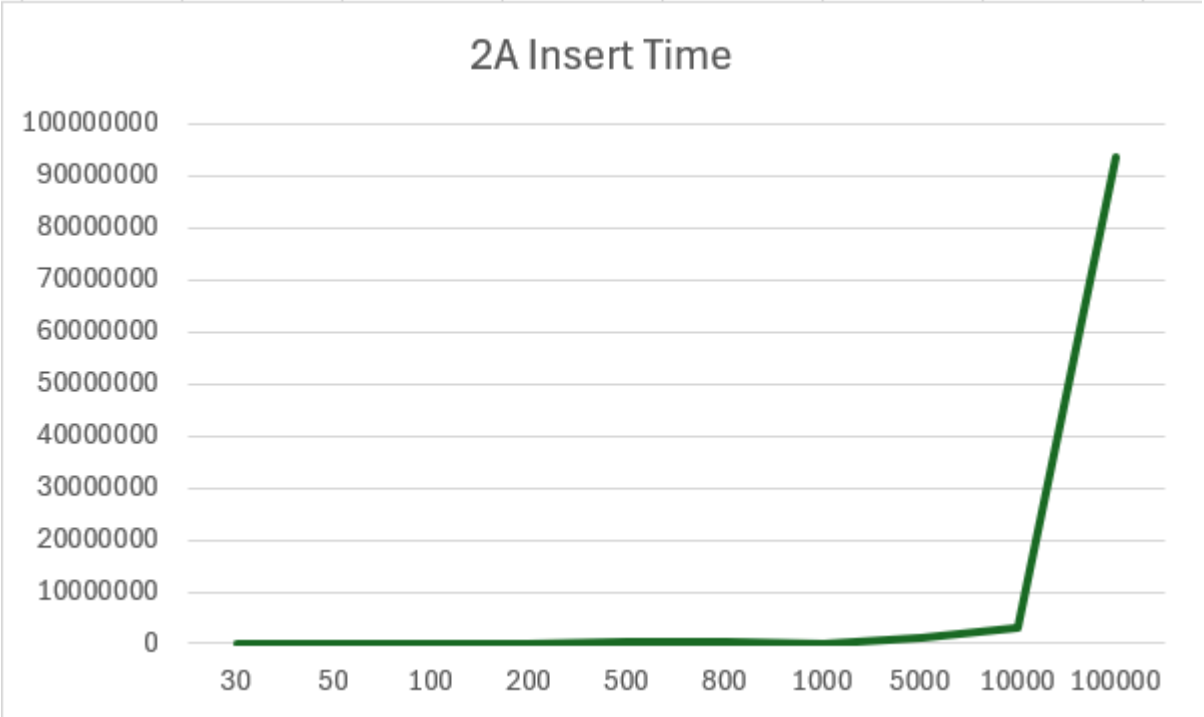
Η επιβολή μοναδικών κλειδιών στις λίστες δεν αλλάζει σημαντικά την πολυπλοκότητα. Αν και διασφαλίζει τη μοναδικότητα, δεν αλλάζει τις θεμελιώδεις λειτουργίες που απαιτούνται για την εισαγωγή, τη διαγραφή και την αναζήτηση, οι οποίες εξακολουθούν να περιλαμβάνουν διέλευση της δομής δεδομένων.

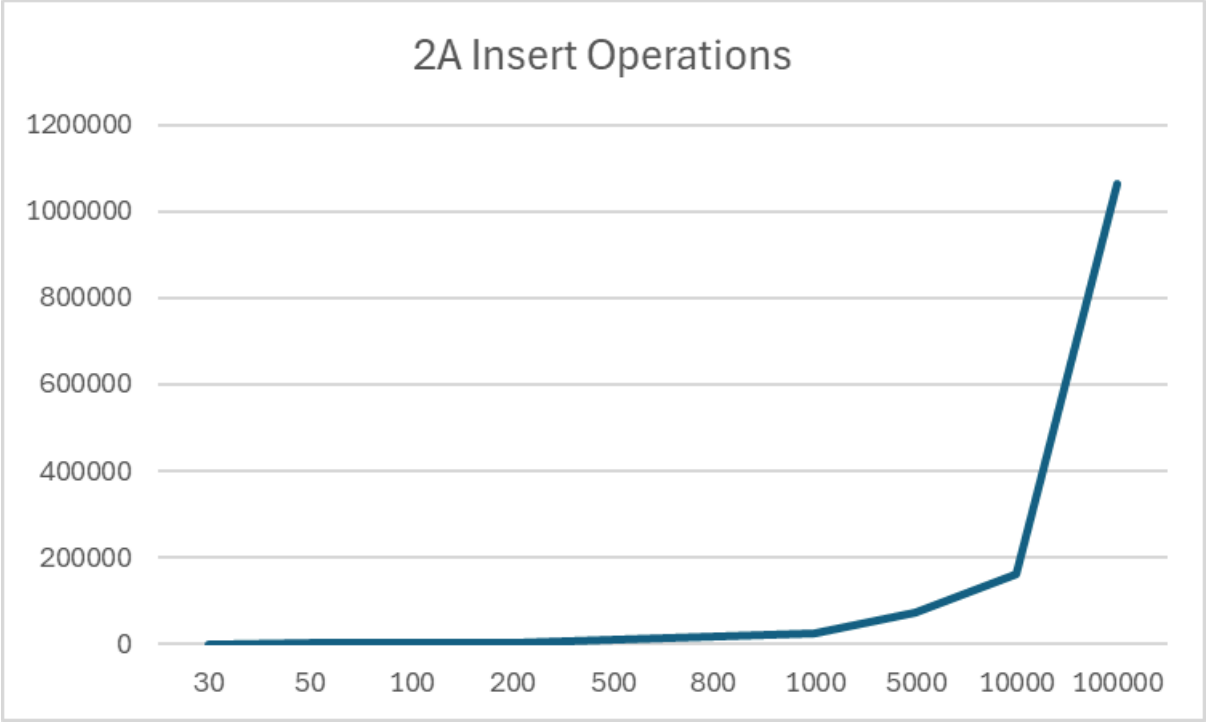
3). Μεταξύ των υλοποιήσεων 1a και 2a, και 1b και 2b, σε απόλυτους χρόνους, ποια περιμένετε να είναι πιο γρήγορη;

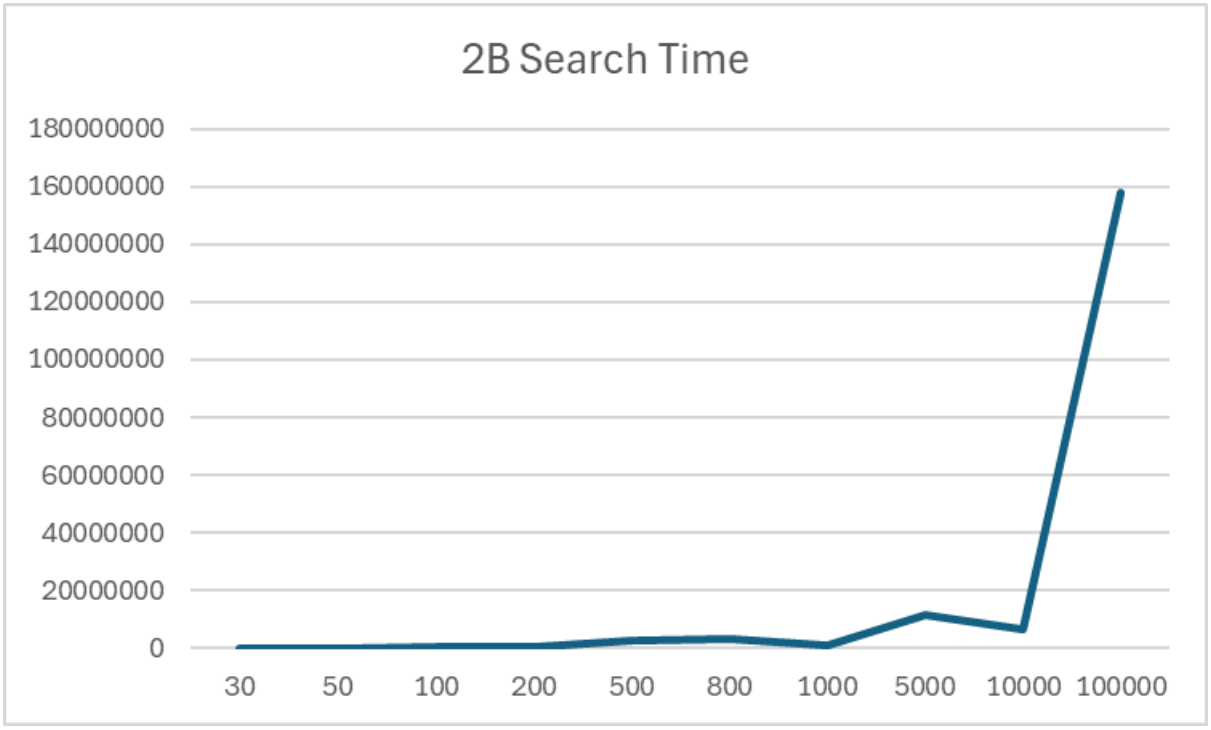
Και οι 4 αυτές υλοποιήσεις έχουν πολυπλοκότητα O(n). Παρόλο που η 1b και 2b είναι sorted υλοποιήσεις, δεν χρησιμοποιούμε καποιου είδους search οπως το binary search (αυτό το χρησιμοποιούμε μόνο στη 3β), με αποτέλεσμα σε ολες τις μεθόδους να ψάχνουμε τα κλειδιά σειριακά. Αρα, το αν η μέθοδος είναι sorted η unsorted δεν παίζει ρόλο στην ταχύτητα της εκτέλεσης (θα έπαιζε αμα δεν χρησιμοποιούσαμε σειριακή αναζήτηση και στα δυο). Παρολαυτα, ενδεχεται οι υλοποιήσεις 2a και 2b να ειναι γρηγοροτερες, καθως στη χρηση του στατικού πίνακα 2D, τα στοιχεία αποθηκεύονται συνεχόμενα στη μνήμη, γεγονός που μπορεί να οδηγήσει σε ταχύτερους χρόνους πρόσβασης σε σύγκριση με υλοποιήσεις που χρησιμοποιούν δυναμική εκχώρηση μνήμης, όπου οι κόμβοι μπορεί να είναι διάσπαρτοι στη μνήμη. Αρα καταλήγουμε στο αποτέλεσμα ότι οι 2α και 2β είναι γρηγορότερες υλοποιήσεις.

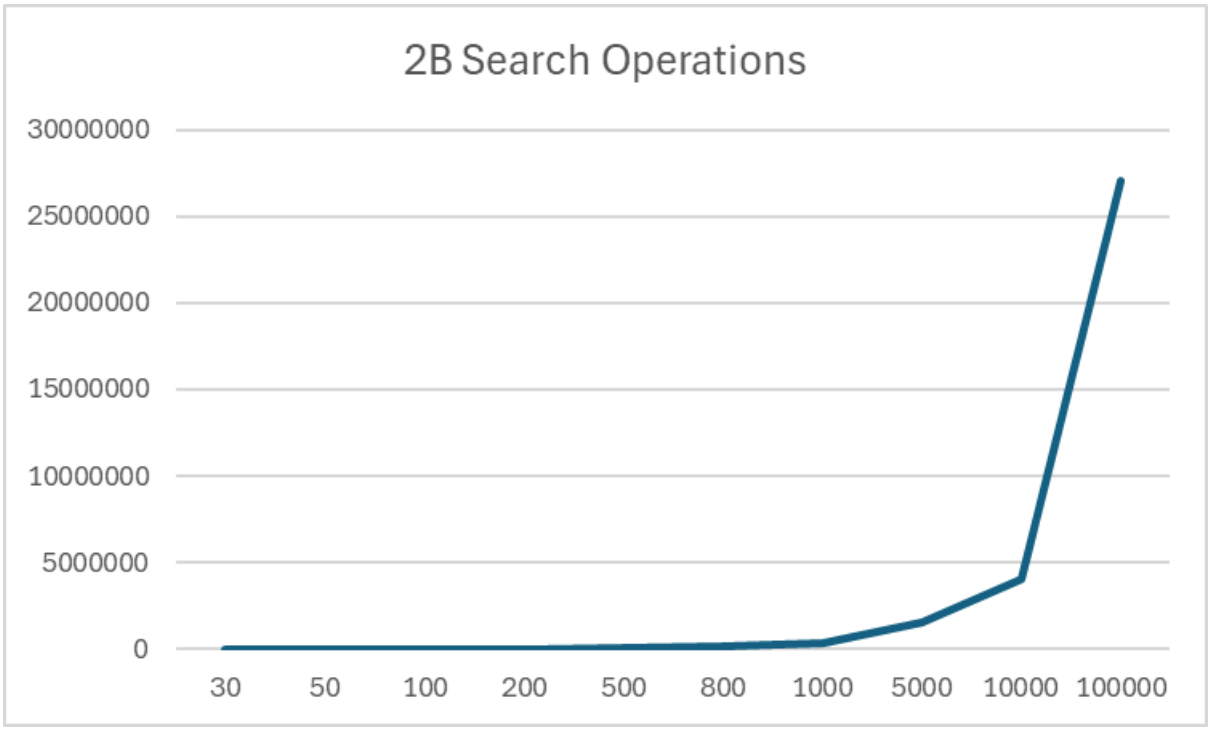
2a vs 2b:
Αφού χρησιμοποιούμε και στα δύο σειριακή αναζήτηση στη μέθοδο search(), αυτό σημαίνει ότι ψάχνουμε ένα-ένα τα κλειδιά μέχρι να βρούμε αυτό που μας έχει δοθεί. Για παράδειγμα, μπορεί στην unsorted μέθοδο το κλειδί 100 να βρίσκεται στην αρχή της λίστας και να το βρούμε αμέσως, ενώ στην sorted το κλειδί αυτό θα μετακινηθεί προς το τέλος καθώς είναι μεγάλο, άρα θα χρειαστεί περισσότερος χρόνος. Αντίθετα, αν μας δοθεί μικρό κλειδί μπορεί στην unsorted μέθοδο να είναι στο τέλος της λίστας αλλά στην sorted θα είναι πάντα στην αρχή. Άρα δεν μπορούν να συγκριθούν αυτά τα δύο προς την ταχύτητα.

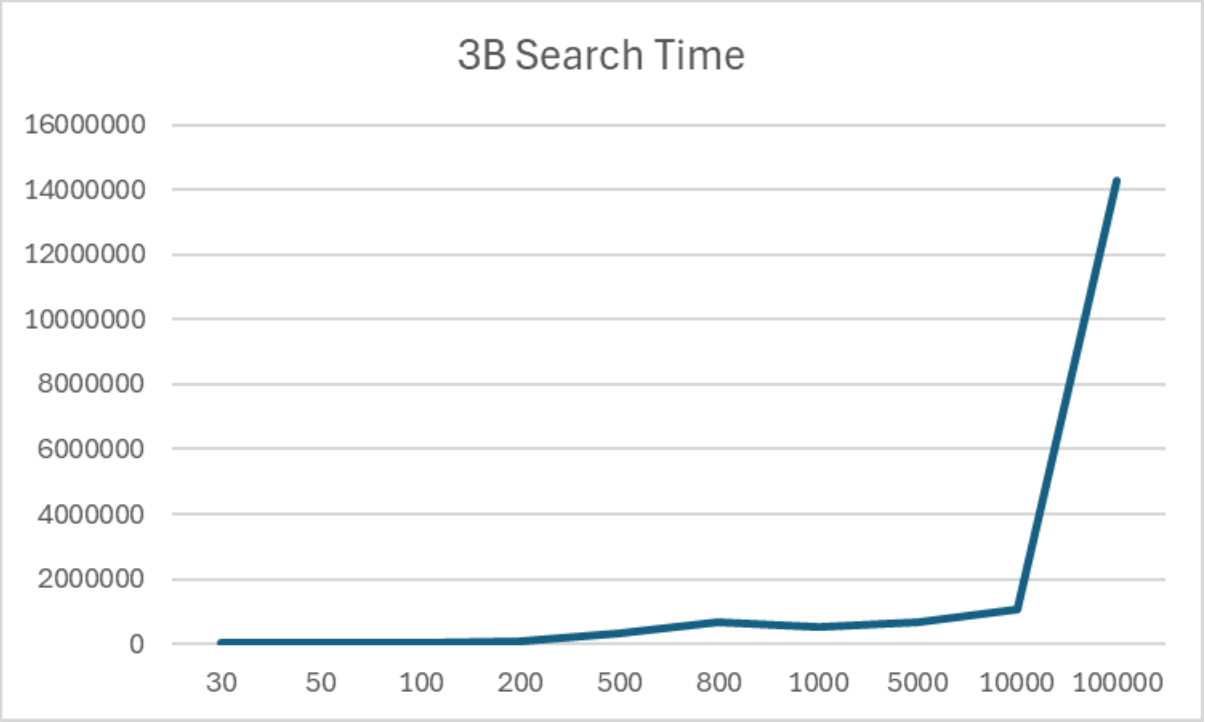
ΓΡΑΦΗΜΑΤΑ

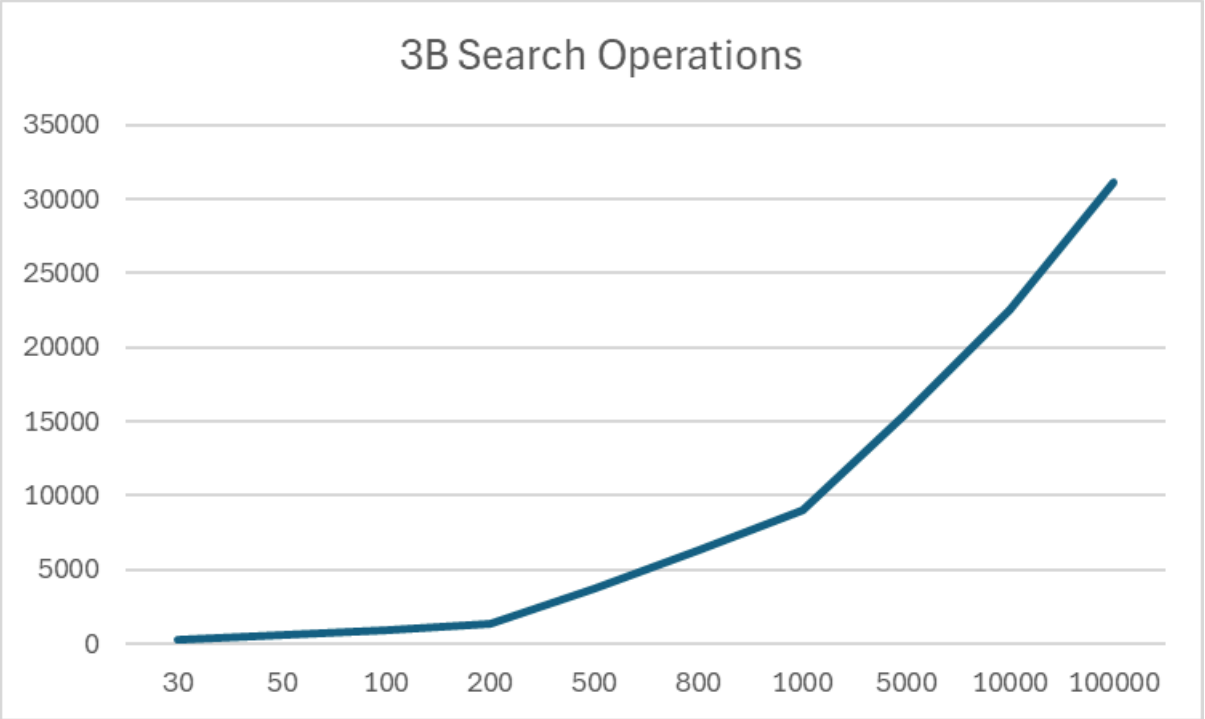

2A Insert Time
100000000
90000000
80000000
70000000
60000000
50000000
40000000
30000000
20000000
10000000
0
30
50
100
200
500
800
1000
5000
10000
100000


2A Insert Operations
1200000
1000000
800000
600000
400000
200000
0
30
50
100
200
500
800
1000
5000
10000
100000


2B Search Time
180000000
160000000
140000000
120000000
100000000
80000000
60000000
40000000
20000000
0
30
50
100
200
500
800
1000
5000
10000
100000


2B Search Operations
30000000
25000000
20000000
15000000
10000000
5000000
0
30
50
100
200
500
800
1000
5000
10000
100000





Πηγές:
Η συνάρτηση BinarySearch είναι βασισμένη σε ένα thread από το StackOverflow.