Τμήμα Μηχανικών Η/Υ & Πληροφορικής

Δομές Δεδομένων
Εργαστηριακή Άσκηση
Ακαδημαϊκό Έτος 2017 – 2018

Ονοματεπώνυμο	Καλαματιανού Δήμητρα
Αριθμός Μητρώου	1054406
email	kalamatianou@ceid.upatras.gr up1054406@upnet.gr
Γλώσσα προγραμματισμού	C++

Αναφορά εργαστηριακής άσκησης

Μέρος Α

Για την υλοποίηση του μέρους Α δημιούργησα την συνάρτηση Load_Integers() η οποία "φορτώνει" τους ακέραιους αριθμούς του αρχείου integers.txt σε έναν δυναμικό πίνακα(vector) , τον οποίο έχω ονομάσει <<vec>>. Επίσης εμφανίζεται ανάλογο μήνυμα για το αν διαβάστηκε ή όχι το αρχείο.

Για να γίνεται ταξινόμηση των περιεχομένων του αρχείου έγινε χρήση του αλγορίθμου Merge Sort. Για την υλοποίηση του αλγορίθμου δημιούργησα δύο συναρτήσεις , την Merge και τη Merge_Sort. Η συνάρτηση Merge_Sort καλώντας αναδρομικά τον εαυτό της σπάει τον πίνακα σε υποπίνακες μέχρι κάθε πίνακας να αποτελείται από ένα στοιχείο και η Merge συνάρτηση ενοποιεί τους πίνακες που δημιούργησε η Merge_Sort τοποθετώντας τα στοιχεία ταξινομημένα σε πίνακα.

Μέρος Β

1.*Γραμμική αναζήτηση*

Η συνάρτηση Linearsearch() έχει στα ορίσματα της τον αριθμό που θέλουμε να αναζητήσουμε καθώς και τον δυναμικό πίνακα(vector),στον οποίο έχουν φορτωθεί όλοι οι αριθμοί του αρχείου integers.txt. Εκτελεί γραμμική αναζήτηση , δηλαδή ελέγχονται τα

στοιχεία του vector από το πρώτο μέχρι το τελευταίο ένα προς ένα ώστε να βρεθεί ή όχι ο αριθμός που αναζητούμε και εμφανίζεται ανάλογο μήνυμα με βάση αν βρέθηκε ο αριθμός ή όχι.

2. *Binary search*

Η συνάρτηση Binarysearch() έχει στα ορίσματα της τον αριθμό που θέλουμε να αναζητήσουμε, το πρώτο στοιχείο του πίνακα , το τελευταίο στοιχείο του πίνακα καθώς και τον δυναμικό πίνακα (vector),στον οποίο έχουν φορτωθεί όλοι οι αριθμοί του αρχείου integers.txt. Εκτελεί δυαδική αναζήτηση αναδρομικά αφού πρώτα υπολογιστεί το μεσαίο στοιχείο* , δηλαδή όσο το πρώτο στοιχείο του πίνακα είναι μικρότερο ή ίσο του του τελευταίου στοιχείου του πίνακα η συνάρτηση ελέγχει αν ο αριθμός που ψάχνουμε είναι το μεσαίο στοιχείο του πίνακα και αν είναι το επιστρέφει. Διαφορετικά ελέγχει αν ο αριθμός που ψάχνουμε είναι μικρότερος του μεσαίου στοιχείου , αν είναι καλείται αναδρομικά η Binarysearch() στον μισό πίνακα , από το πρώτο στοιχείο του πίνακα έως το μεσαίο, αν πάλι είναι μεγαλύτερος του μεσαίου στοιχείου τότε καλείται αναδρομικά η Binarysearch() στον μισό πίνακα , αυτή την φορά όμως από το μεσαίο στοιχείο + 1 έως το τελευταίο στοιχείο. Εμφανίζεται ανάλογο μήνυμα με βάση αν βρέθηκε ο αριθμός ή όχι.

*για τον υπολογισμό του μεσαίου στοιχείου έχω χρησιμοποιήσει την nearbyint ώστε αν στην διαίρεση : (πρώτο στοιχείο του πίνακα + τελευταίο στοιχείο του πίνακα)/2 προκύψει δεκαδικό ο αριθμός αυτός να στρογγυλοποιηθεί στο πιο κοντινό του ακέραιο

3.*Interpolation search*

Η συνάρτηση Interpolationsearch() έχει στα ορίσματα της τον αριθμό που θέλουμε να αναζητήσουμε καθώς και τον δυναμικό πίνακα(vector),στον οποίο έχουν φορτωθεί όλοι οι αριθμοί του αρχείου integers.txt. Εκτελεί αναζήτηση παρεμβολής με βάση τον παρακάτω τύπο:

$next \leftarrow [[(x - S[left]) * (right - left)] / (S[right] - S[left])] + left$

Όσο ο αριθμός που ψάχνουμε είναι α) μεγαλύτερος ή ίσος του αριστερού άκρου του πίνακα και β) μικρότερος ή ίσος του δεξιού άκρου του πίνακα τότε υπολογίζουμε με τον παραπάνω τύπο πόσο μεγαλύτερο είναι το ζητούμενο στοιχείο από το αριστερό άκρο και πόσο μεγαλύτερο είναι το δεξί άκρο από το αριστερό. Εμφανίζεται ανάλογο μήνυμα με βάση αν βρέθηκε ο αριθμός ή όχι.

Μέρος Γ

Για την υλοποίηση αυτού του μέρους δημιούργησα header file <<RedBlack.h>> . Για την εισαγωγή αριθμού χρησιμοποιείται η συνάρτηση red_black_insert(). Αρχικά δημιουργείται ένας κόμβος που αποτελεί ρίζα και στην συνέχεια γίνεται εισαγωγή των ακέραιων αριθμών που υπήρχαν στο αρχείο integers.txt. Για κάθε εισαγωγή ενός στοιχείου στο δέντρο πρέπει να ικανοποιούνται οι εξής περιορισμοί:

α) Η ρίζα είναι μαύρη

β) Τα φύλλα είναι μαύρα

γ) Κάθε μονοπάτι από τη ρίζα ως τα φύλλα έχει τον ίδιο αριθμό μαύρων κόμβων

δ) Κάθε κόκκινος κόμβος έχει μαύρο πατέρα

Για να ικανοποιούνται αυτοί οι περιορισμοί λοιπόν δημιούργησα τις συναρτήσεις right_rotation() , left_rotation() , insertfix(). Οι κόμβοι συνδέονται με δείκτες και με βάση τις προηγούμενες συναρτήσεις γίνονται περιστροφές και αλλαγές χρωμάτων ώστε να μην παραβιαστεί κάποιος από τους παραπάνω κανόνες.

Στην συνάρτηση insertfix() αν κάποιος κόμβος είναι ρίζα , το χρώμα του κόμβου γίνεται μαύρο. Αν το χρώμα του πατέρα του κόμβου x είναι κόκκινο ή ο x είναι ρίζα τότε : α)Αν το άλλο παιδί-κόμβος(θείος) του παππού-κόμβος του x είναι κόκκινο τότε ο πατέρας και ο θείος γίνονται μαύροι , ο παππούς κόκκινος και επαναλαμβάνω την διαδικασία για x=παππούς του x , β)Αν ο θείος είναι μαύρος τότε i) αν ο πατέρας του x είναι αριστερό παιδί του παππού του x και ο x είναι αριστερό παιδί του πατέρα του γίνεται αριστερή περιστροφή με την συνάρτηση left_rotation() , ii) αν ο πατέρας του x είναι δεξί παιδί του παππού του x και ο x είναι δεξί παιδί του πατέρα του γίνεται δεξιά περιστροφή με την συνάρτηση right_rotation() , iii) αν ο πατέρας του x είναι αριστερό παιδί του παππού του x και ο x είναι δεξί παιδί του πατέρα του γίνεται αριστερή και δεξιά περιστροφή με τις συναρτήσεις left_rotation() και right_rotation() , iv) αν ο πατέρας του x είναι δεξί παιδί του παππού του x και ο x είναι αριστερό παιδί του πατέρα του γίνεται δεξιά και αριστερή περιστροφή με τις συναρτήσεις right_rotation() και left_rotation().

Για την αναζήτηση χρησιμοποιείται η συνάρτηση rbsearch() η οποία ξεκινάει από την ρίζα και αν ο αριθμός που ψάχνουμε για κάθε κόμβο που βρίσκουμε είναι i) ίσος με τον κόμβο τότε επιστρέφει τον κόμβο ii) μεγαλύτερος του επισκέπτεται το δεξί παιδί του κόμβου iii) μικρότερος του επισκέπτεται το αριστερό παιδί του κόμβου

Μέρος Δ

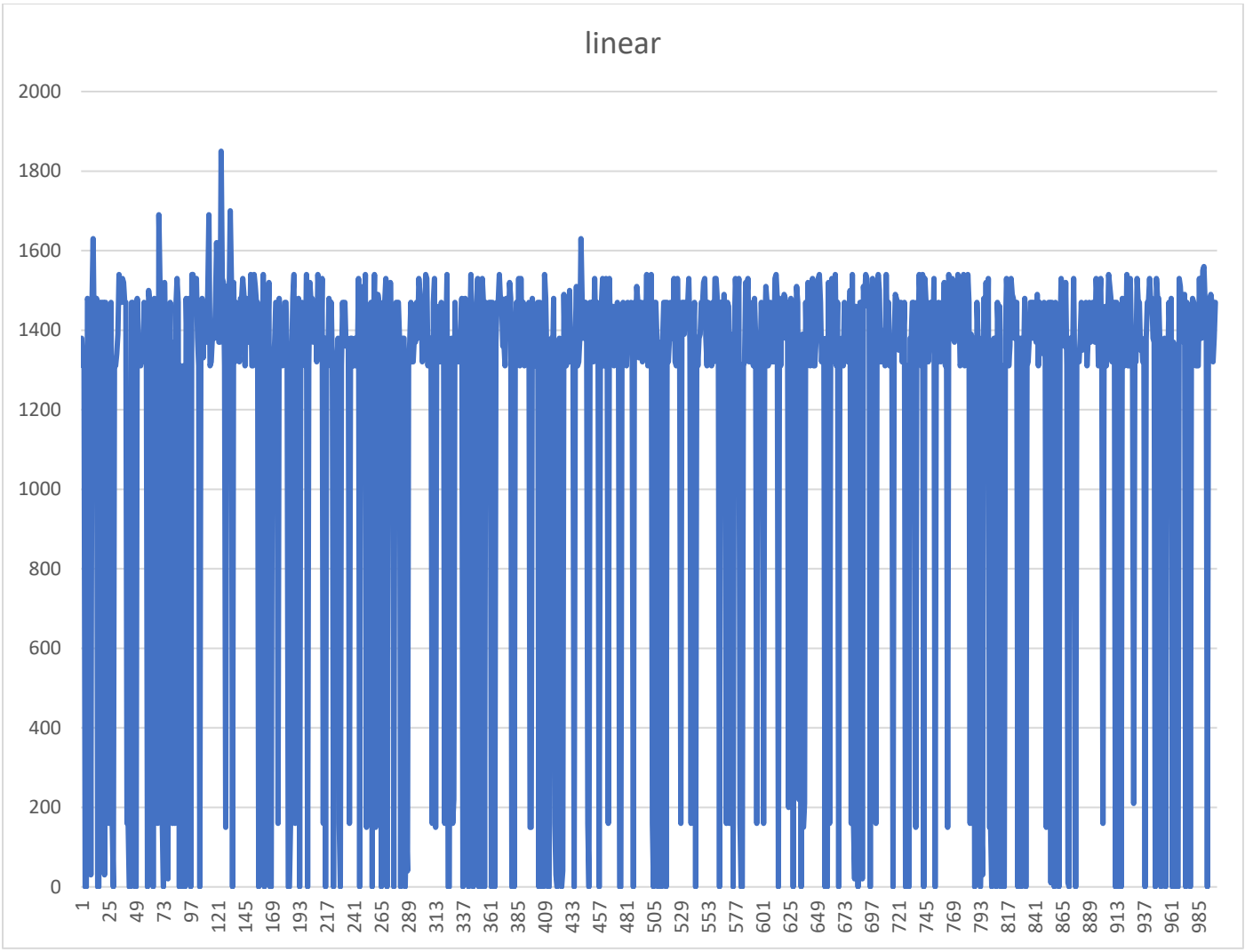
	Χειρότερος χρόνος αναζήτησης(ms)	Μέσος χρόνος αναζήτησης(ms)
Linear search	1850	1162.92
Binary search	2.3	0.8129
Interpolation search	540	184.79
Red Black search	0.21	0.05285

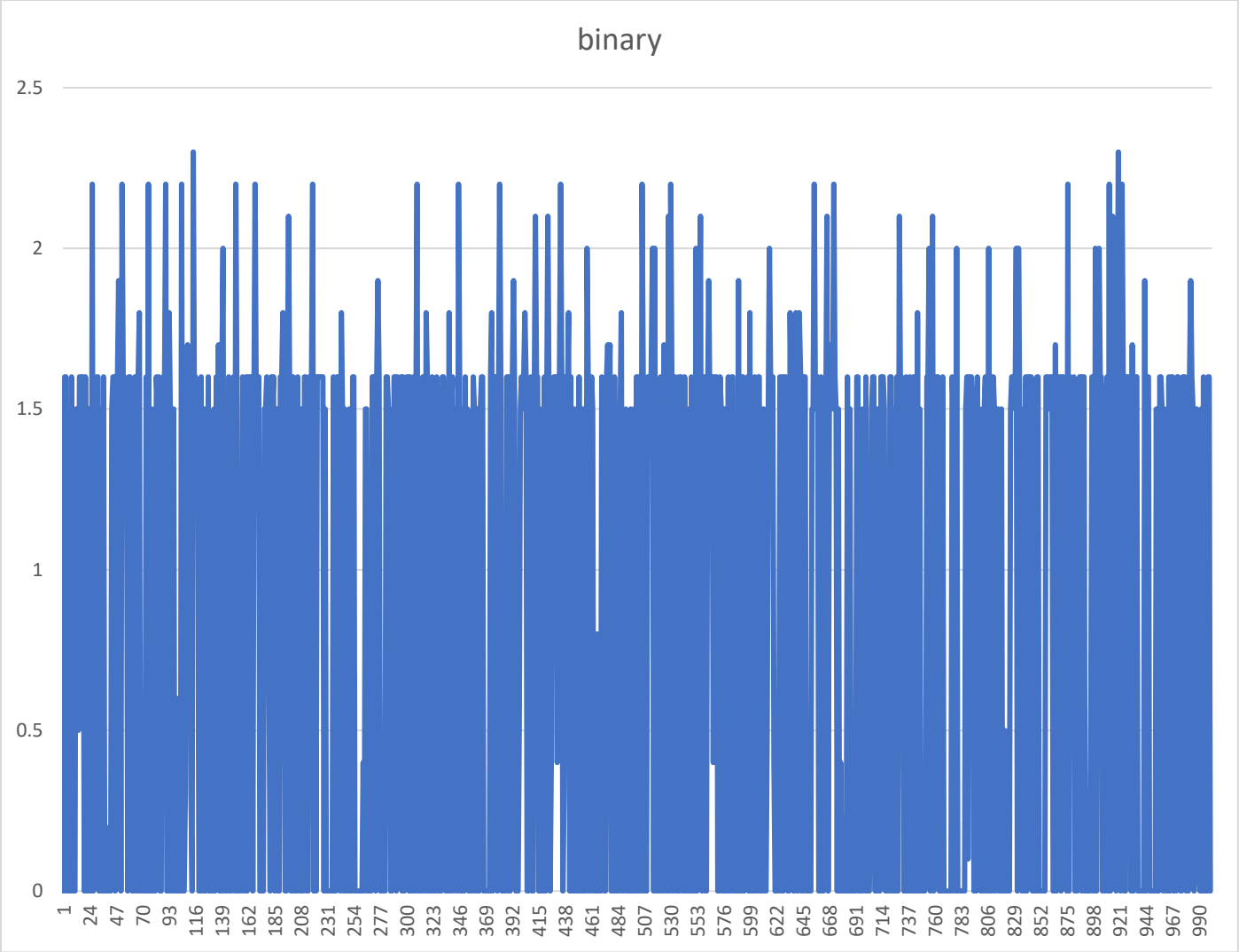
Με τον ακέραιο how_many_numbers = 1000 δηλώνω ότι θα γίνει αναζήτηση 1000 αριθμών σε κάθε μια από τις 4 αναζητήσεις. Επίσης δηλώνω τον ακέραιο random_number = rand() %500000 +1 ώστε να οι 100 αριθμοί να είναι μεταξύ του 1 και του 500000. Στις Linear search και Interpolation search ψάχνω τον ίδιο αριθμό για 100 φορές, στην Binary search ψάχνω τον ίδιο αριθμό για 10000 φορές και στην Red Black search ψάχνω τον ίδιο αριθμό για 100000 φορές. Ο ίδιος αριθμός αναζητείται κάθε φορά και στις 4 αναζητήσεις.

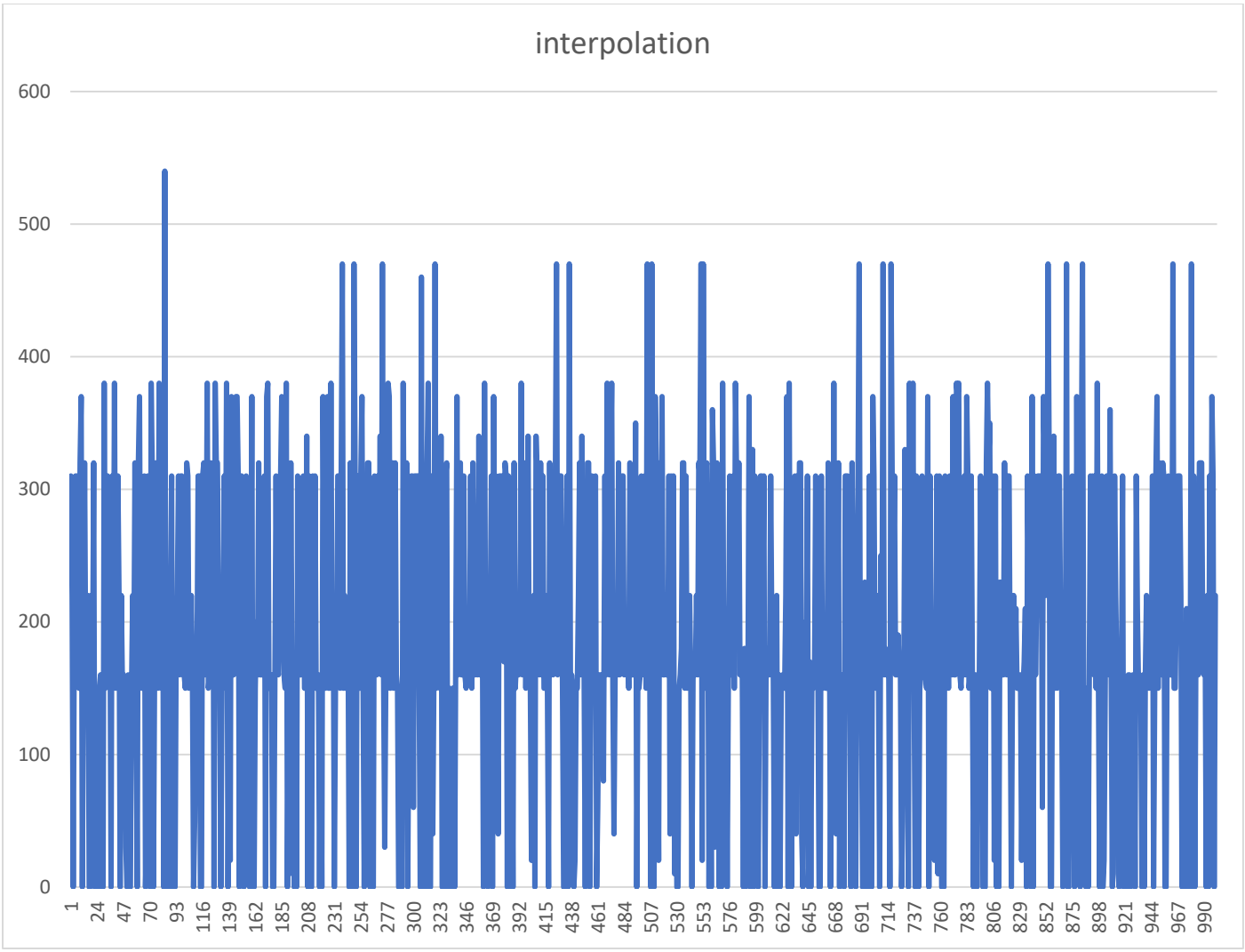
Για να μετρήσω τους χρόνους κάνω comments τα μηνύματα που βγαίνουν από κάθε αναζήτηση για το αν βρέθηκε ή όχι το στοιχείο.

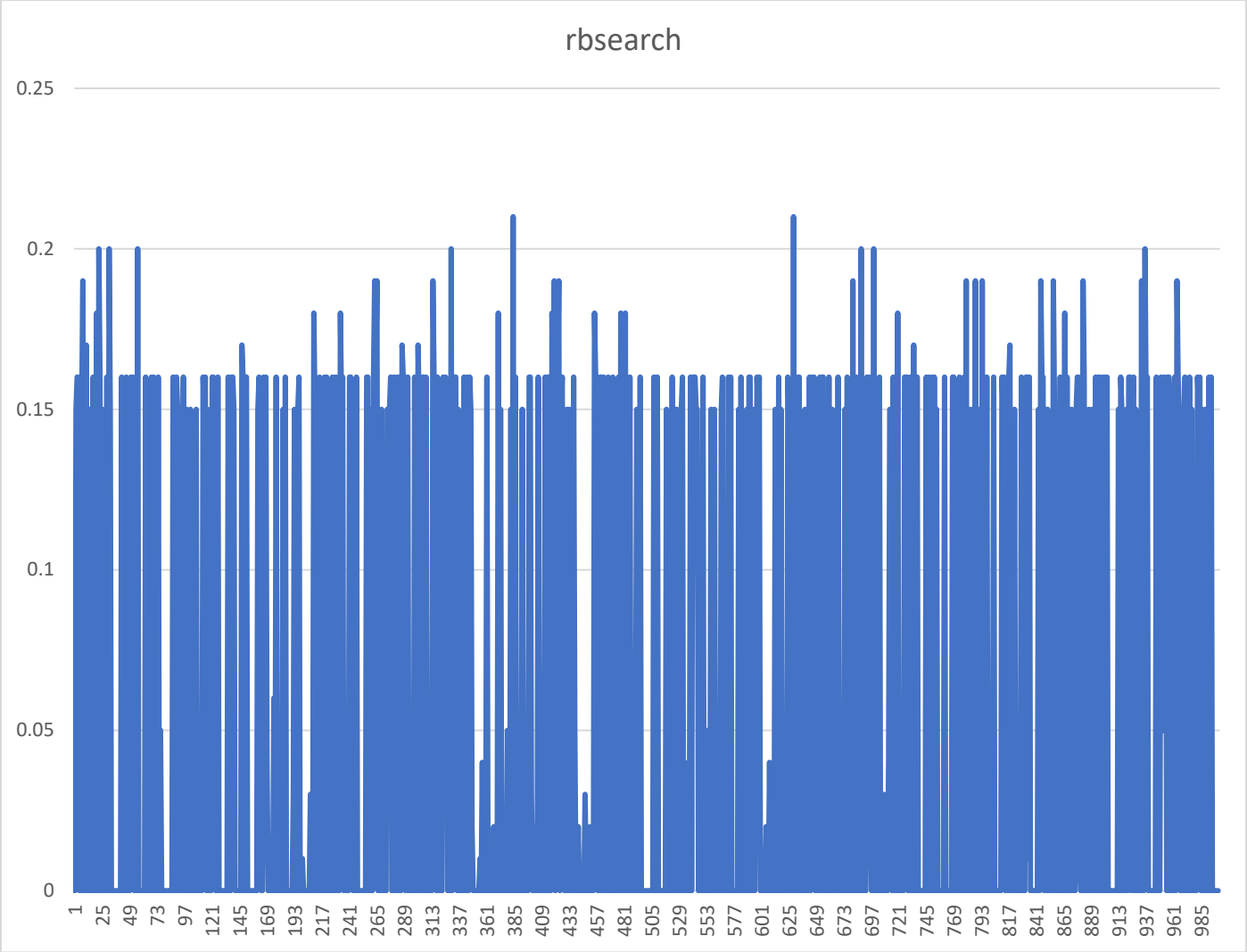
Παρακάτω παρουσιάζονται τα διαγράμματα κάθε αναζήτησης :

Όπου ο κάθετος άξονας είναι ο χρόνος και ο οριζόντιος σε ποια επανάληψη εμφανίστηκε αυτός ο χρόνος


linear
2000
1800
1600
1400
1200
1000
800
600
400
200
0
1
25
49
73
97
121
145
169
193
217
241
265
289
313
337
361
385
409
433
457
481
505
529
553
577
601
625
649
673
697
721
745
769
793
817
841
865
889
913
937
961
985


binary
2.5
2
1.5
1
0.5
0
1
24
47
70
93
116
139
162
185
208
231
254
277
300
323
346
369
392
415
438
461
484
507
530
553
576
599
622
645
668
691
714
737
760
783
806
829
852
875
898
921
944
967
990


interpolation
600
500
400
300
200
100
0
1
24
47
70
93
116
139
162
185
208
231
254
277
300
323
346
369
392
415
438
461
484
507
530
553
576
599
622
645
668
691
714
737
760
783
806
829
852
875
898
921
944
967
990



Η γραμμική αναζήτηση έχει χρόνο O(n)

Η δυαδική αναζήτηση στην χειρότερη περίπτωση έχει χρόνο O(nlogn).

Η αναζήτηση παρεμβολής στην χειρότερη περίπτωση έχει χρόνο O(n)

Η αναζήτηση στο red black tree στην χειρότερη περίπτωση είναι O(logn)

Για να βρεθούν αυτοί οι χρόνοι υλοποίησα τις συναρτήσεις printcsv , printtxt για να περαστούν σε αρχείο csv και txt αντίστοιχα.

void **printcsv**(double time[][4],int times)

{

int ans;

cout << "are you sure you would like to overwrite the csv file?(1 for yes)\n";

cin >> ans;

string words[]={"linear","binary","interpolation","rbsearch"};

```
if (ans==1)
{
ofstream myfil;
myfil.open ("output.csv");
for(int k=0; k<4; k++){
 myfil<<words[k]<<";";
for(int l=1;l<times+1;l++)
myfil << time[l-1][k]<< ";";
myfil << "\n";
}
myfil.close();
}
}
void printtxt(double time[][4],int times)
{
int ans;
double sum[]={0.0, 0.0, 0.0, 0.0};
double worst[] ={0.0, 0.0, 0.0, 0.0};
cout << "are you sure you would like to overwrite the txt file?(1 for yes)\n";
cin >> ans;
string words[]={"linear","binary","interpolation","rbsearch"," 1000 milliseconds","1000 milliseconds", "1000 milliseconds","1000 milliseconds"};
if (ans==1)
{
ofstream myfil;
myfil.open ("output.txt");
for(int k=0; k<4; k++){
myfil<<words[k]<<"  ";
for(int l=1;l<times+1;l++){
```

```
if(worst[k]<time[l-1][k])
{worst[k]=time[l-1][k];}
sum[k]=sum[k]+time[l-1][k];}
myfil<<" The worst time is "<<worst[k]<<" and the average time is
"<<sum[k]/(double)times<<" milliseconds\n";
}
myfil.close();
}
}
```

Στην main τοποθέτησα τον κώδικα :

```
{
 int how_many_numbers=1000; //for the time
int start_s,stop_s=0;

double time[how_many_numbers][4];
 for (int p=0; p<how_many_numbers; p ++){
 for (int mode=0; mode<4; mode++ )
 { time[p][mode]=0.0;}}

 for (int p=0; p<how_many_numbers; p ++)
 {
 int random_number = rand() %500000 +1  ;

 int mode =0 ;
 start_s=clock();
 for(int d=0; d<100; d++)
 {
 Linearsearch(random_number,vec);
 }
 stop_s=clock();
```

```
time[p][mode]=((stop_s-start_s)/ (float)(CLOCKS_PER_SEC)* 1000.0)*10.0; //msec
mode++;

start_s=clock();
for(int d=0; d<10000; d++)
{
Binarysearch(vec,random_number,0,(int)vec.size()-1);
}
stop_s=clock();
time[p][mode]=((stop_s-start_s)/ (float)(CLOCKS_PER_SEC)* 1000.0)/10.0;//msec
mode++;

start_s=clock();
for(int d=0; d<100; d++)
{
Interpolationsearch(random_number,vec);
}
stop_s=clock();
time[p][mode]=((stop_s-start_s)/ (float)(CLOCKS_PER_SEC)* 1000.0)*10.0;//msec
mode++;

start_s=clock();
for(int d=0; d<100000; d++)
{
rbsearch(root,random_number,leaf);
}
stop_s=clock();
time[p][mode]=((stop_s-start_s)/ (float)(CLOCKS_PER_SEC) * 1000.0)/100.0;//msec
}
```

```
printcsv(time,how_many_numbers);
printtxt(time,how_many_numbers);
        }
```

Μέρος Ε

Για την υλοποίηση του μέρους Ε δημιούργησα την συνάρτηση Load_Words () η οποία "φορτώνει" τις λέξεις, οι οποίες αποτελούνται από χαρακτήρες του λατινικού αλφαβήτου σε έναν δυναμικό πίνακα(vector) , τον οποίο έχω ονομάσει words_vector. Επίσης εμφανίζεται ανάλογο μήνυμα για το αν διαβάστηκε ή όχι το αρχείο.

Για την υλοποίηση αυτού του μέρους δημιούργησα header file <<Trie.h>>.Ουσιαστικά κάθε κόμβος του δέντρου είναι ένα πίνακας από δείκτες με κάθε θέση του πίνακα να αποτελεί ένα γράμμα της αλφαβήτου.

Για τη αναζήτηση λέξης χρησιμοποιείται η συνάρτηση trie_search() η οποία ξεκινάει διατρέχοντας τον πίνακα της αλφαβήτου του πρώτου κόμβου, δηλαδή τον array_of_letters, ψάχνοντας το πρώτο γράμμα της λέξης και όταν το βρει πηγαίνει στον κόμβο που δείχνει ο δείκτης και συνεχίζει για τα επόμενα γράμματα. Αν κάποια θέση του πίνακα δεέχνει σε κενό (NULL) σημαίνει ότι δεν υπάρχει αυτή η λέξη. Επίσης αν η μεταβλητή last_letter είναι ψευδής στο τελευταίο γράμμα της λέξης που ψάχνουμε τότε σημαίνει ότι η λέξη δεν υπάρχει ακόμη και αν υπάρχουν τα γράμματα της. Ενώ αν η μεταβλητή last_letter είναι αληθής στο τελευταίο γράμμα της λέξης που ψάχνουμε τότε η λέξη υπάρχει. Εμφανίζεται ανάλογο μήνυμα με βάση αν βρέθηκε η λέξη ή όχι.

Για τη εισαγωγή λέξης χρησιμοποιείται η συνάρτηση trie_insert() η οποία ξεκινάει από την ρίζα του δένδρου και ελέγχει αν το πρώτο γράμμα της λέξης που ψάχνουμε δείχνει σε κάποιον άλλον κόμβο , αν δείχνει συνεχίζει στον παρακάτω μέχρι να βρει κελί του πίνακα που να δείχνει σε κενό και το βάζει να δείχνει στον επόμενο γράμμα της λέξης του επόμενου κόμβου. Στο τελευταίο γράμμα της λέξης θέτει την μεταβλητή last_letter του κόμβου αυτού αληθή.

Για τη διαγραφή λέξης δεν μπόρεσα να υλοποιήσω συνάρτηση αλλά ουσιαστικά αρχικά θα έλεγχε αν υπάρχει η λέξη που θέλουμε να διαγράψουμε. Στην συνέχεια θα γινόταν έλεγχος για το αν υπάρχει λέξη που χρησιμοποιεί τα ίδια γράμματα με την λέξη που θέλουμε να διαγράψουμε. Αν υπάρχει και η λέξη που θέλουμε να διαγράψουμε έχει περισσότερα γράμματα τότε διαγράφουμε τους κόμβους τους παρακάτω κόμβους από την λέξη που έχει αντίστοιχα γράμματα με αυτήν που θέλουμε να διαγράψουμε. Διαφορετικά αν η λέξη

που θέλουμε να διαγράψουμε έχει λιγότερα γράμματα τότε αλλάζουμε την μεταβλητή last_letter, για το τέλος της λέξης που θέλουμε να διαγράψουμε σε ψευδή.