2η Άσκηση στις Δομές Δεδομένων

Θέμα : Ψάξιμο σε Δομές Δεδομένων

Ομάδα :
Τζένος Δημήτρης AM : 2192
Δαούσης Δημήτρης AM : 2096
Πέρδικα Πολυτίμη AM : XXXX

Γενικά

1. Τα προγράμματα αυτής της άσκησης έχουν υλοποιηθεί σε C. Στην δισκέτα υπάρχουν τόσο τα source αρχεία όσο και τα εκτελέσιμα (για περιβάλλον Windows 98).
2. Για την άσκηση 2 εδώ δίνουμε μόνο τις συναρτήσεις οι οποίες υλοποιούν τον ζητούμενο αλγόριθμο. Αυτές για να "τρέξουν" μπορούν να χρησιμοποιήσουν την παρακάτω main() - πρότυπο. Υποθέτουμε ότι οι υπόλοιπες συναρτήσεις που χρησιμοποιούνται λειτουργούν κανονικά (στα πηγαία αρχεία στην δισκέτα τα προγράμματα είναι έτοιμα προς μετάφραση - εκτέλεση, έχουν στα αρχεία υλοποιημένες όλες τις συναρτήσεις που χρειάζονται).

```
#include <stdio.h>
#include <limits.h>
#include <stdlib.h>

int find(int *,int,int,int);
void swap(int *,int,int);
int counter;

void main(int argc, char *argv[])
{
      extern int counter;
      int k,NumOfEls,x;
      int v[2003];
      FILE *fp;

      if(argc==1) {
            printf("(%s). Syntax :\n",argv[0]);
            printf("     %s input_file element\n",argv[0]);
            printf("input_file : file to be searched\n");
            printf("element    : the element to be searched\n");
            return;
      }
    if(argc>3) {
            printf("Too many arguments. Exiting ...\n");
            return;
      }
      fp=fopen(*++argv,"r");
      v[0]=INT_MIN;
      NumOfEls=1;
      for(k=10;k!=EOF;){
            k=fscanf(fp,"%d ",&v[NumOfEls]);
          if(k!=0 && k!=EOF) NumOfEls++;
      }
      fclose(fp);
      k=atoi(*++argv);
      v[NumOfEls]=INT_MAX;
      x=function(v,k,1,NumOfEls-1);
      printf("The element is %d\n",x);
      printf("Number of comparisons :%d\n",counter);
}
```

Άσκηση 1η

Ζητείται να υλοποιηθεί σε C μια μέθοδος χειρισμού τυχαίων κομβοπροσανατολισμένων δυϊκών δένδρων ψαξίματος. Με άλλα λόγια πρέπει να υλοποιήσουμε τις βασικές πράξεις για αυτά τα δένδρα (Ψάξιμο, Ένθεση και Απόσβεση). Για να το κάνουμε αυτό θα χρησιμοποιήσουμε λίστες από structures. Κάθε στοιχείο του δένδρου θα αποτελεί και μια structure με στοιχεία :

1. Δείκτη στον πατέρα του
2. Δείκτη στον αριστερό υιό του
3. Δείκτη στον δεξί υιό του
4. Την τιμή του
5. Έναν τυχαίο αριθμό από 0 μέχρι 1, την προτεραιότητά του.

Η ρίζα έχει πατέρα μια "υποθετική" structure που λέγεται *head*. Στο στοιχείο που είναι κάθε φορά η ρίζα δείχνει ένας δείκτης που τον λέμε *root*. Στην αρχή αυτός ο δείκτης δεν δείχνει πουθενά (έχει τιμή NULL).
Όταν ψάχνουμε το δένδρο για ένα στοιχείο, τότε ξεκινούμε από τη ρίζα και κατεβαίνουμε το δένδρο μέχρι να βρούμε εσωτερικό κόμβο ή φύλλο που να περιέχει το στοιχείο που αναζητούμε. Αν δεν το βρούμε, τότε το ψάξιμο είναι ανεπιτυχές και σταματά στο αμέσως μεγαλύτερο ή μικρότερο στοιχείο στο δένδρο από αυτό που ζητούμε.
Κάθε φορά που θέλουμε να εισάγουμε ένα νέο στοιχείο πρώτα ψάχνουμε στο δένδρο για το στοιχείο. Αν αυτό υπάρχει, τότε δεν κάνουμε τίποτα. Αν όχι, τότε χρησιμοποιείται μια συνάρτηση που δεσμεύει χώρο στη μνήμη για μια νέα καταχώρηση και στη συνέχεια κάνουμε το νέο στοιχείο υιό τού στοιχείου στο οποίο σταμάτησε το ψάξιμο. (Αν το στοιχείο που θα εισάγουμε είναι μικρότερο, τότε το κάνουμε αριστερό υιό, διαφορετικά το κάνουμε δεξί υιό). Μετά, επιλέγεται η προτεραιότητα και ανάλογα με αυτή κάνουμε δεξιές/αριστερές περιστροφές στο δένδρο μέχρις ότου πάρουμε τέτοια διάταξη ώστε τα στοιχεία με μικρότερη αριθμητικά προτεραιότητα να βρίσκονται ψηλότερα στο δέντρο.
Για να σβήσουμε ένα στοιχείο, πρώτα του δίνουμε άπειρη προτεραιότητα και το "κατεβάζουμε" σε ένα φύλλο με τις ανάλογες αριστερές/δεξιές περιστροφές. Μετά το σβήνουμε.
Το δένδρο παρουσιάζεται με τη βοήθεια της συνάρτησης *symord* που υλοποιεί την ομώνυμη διαδικασία παρουσίασης δυϊκών δένδρων. Επίσης, κάθε φορά δηλώνεται ποιο στοιχείο βρίσκεται στη ρίζα. Το ύψος του δένδρου μετριέται με την συνάρτηση measure.

Όλα αυτά υλοποιούνται σε C με το παρακάτω πρόγραμμα :

```
#include <stdio.h>
#include <stdlib.h>
#define frand() ((double) rand() / (RAND_MAX+1.0))

int found;


typedef struct Node_{
      struct Node_ *parent;
      struct Node_ *l_son;
      struct Node_ *r_son;
      int value;
      double possibility;
} Node;


Node *root,*head;
Node *NodeAlloc(void);
int measure(Node *);
double random_(void);
void balance(Node *s,int i);
void rotright(Node *y, Node *x);
void rotleft(Node *x, Node *y);
void symord(Node *v);


/* psaxnei to stoixeio sto dendro kai epistrefei ton
   komvo ston opoio bre8ike eite ton "plisiestero" os pros
   tin timi komvo. Episis 8etei tin metavliti found se 1 an vre8ike
   autousio to stoixeio i se 0 an den vre8ike */
Node *search(int x){
      Node *current;
      extern found;
      extern Node *root;
      found=0;
      current=root;
      while(current!=NULL&&current->value!=x){
            if(current->value > x )
                  if(current->l_son==NULL){
                        break;
                  }
                  else
                        current=current->l_son;
            else
                  if(current->r_son == NULL){
                        break;
                  }
                  else
                        current=current->r_son;
      }
      if(current==root && root==NULL) return current;
      if(current->value == x){
            found=1;
            return current;
      }
      else return current;
}
```

```
/*8etei to stoixeio x sto dendro an auto den yparxei. Den kanei
  tipota an to stoixeio eite yparxei eite den arkei i mnimi */
void insert(int x){
      extern Node *root,*head;
      Node *p;
      Node *n;
      extern found;

      p=search(x);
      if(found){
            printf("%d : This element already exists. It will not be
added in the tree\n",x);
            return;
      }
      if((n=NodeAlloc())==NULL){
            printf("Insert(%d) : Not enough memory\n",x);
            return;
      }
      n->parent=p;
      n->l_son=NULL;
      n->r_son=NULL;
      n->value=x;
      n->possibility=random_();
      if(root==NULL){
            root=n;
            root->parent=head;
            head->l_son=root;
      }
      if(n!=root){
            if(x < p->value) p->l_son=n;
            else
                  p->r_son=n;
            balance(n,0);
      }
}


/* epistrefei tyxaio ari8m0 apo 0 mexri 1 poy xrisimopoieitai
   gia epilogi proteraiotitas */
double random_(void){
      srand((unsigned) rand());
      return frand();
}


/*Desmevei xwro gia neo komvo */
Node *NodeAlloc(void){
      return (Node *) malloc(sizeof(Node));
}


/*Diagrafei to stoixeio x apo to dendro */
void del(int x){
      Node *p;
      extern int found;

      p=search(x);
      if(!found){
            printf("delete : value %d does not exist in the
tree\n",x);
            return;
```

```
      }
      p->possibility=2;
      if(p->l_son!=NULL)
            balance(p->l_son,1);
      else if(p->r_son!=NULL)
            balance(p->r_son,1);
      if(p->parent->l_son==p){
            p->parent->l_son=NULL;
            free(p);
      }
      else {
            p->parent->r_son=NULL;
            free(p);
      }
}


/* Apofasizei an prepei na ginoun peristrofes, kai an nai ti eidous*/
void balance(Node *s,int i){
      extern Node *head,k;
      Node *l;

    l=s->parent;
      if(s->parent->possibility > s->possibility){
            if(s==s->parent->l_son){
                  rotright(s,s->parent);
            }
            else{
                  rotleft(s,s->parent);
            }
          switch(i){
               case 0 : if(s->parent!=head)
                                    balance(s,0);
                          break;
               case 1 : if(l->l_son!=NULL)
                                    balance(l->l_son,1);
                         else if(l->r_son!=NULL)
                                    balance(l->r_son,1);
                               break;
            }
      }

}
/* Ektelei deksies peristrofes */
void rotright(Node *y, Node *x){
      extern Node *root,*head;

      if(x->parent->l_son==x)
                x->parent->l_son=y;
          else
                x->parent->r_son=y;
      y->parent=x->parent;
      x->parent=y;
      x->l_son=y->r_son;
      if(y->r_son!=NULL){
            y->r_son->parent=x;
      }
      y->r_son=x;
    root=head->l_son;
}
/* Ektelei aristeres peristrofes */
```

```
void rotleft(Node *x, Node *y){
      extern Node *root,*head;

      if(y->parent->l_son==y)
          y->parent->l_son=x;
      else
          y->parent->r_son=x;
      x->parent=y->parent;
      y->parent=x;
      y->r_son=x->l_son;
      if(x->l_son!=NULL){
            x->l_son->parent=y;
      }
      x->l_son=y;
    root=head->l_son;
}

void main(){
      int k,i;
      char c[10],fil[12];
      Node *l;
      FILE *fp;
      extern Node *root,*head;
    root=NULL;

      head=NodeAlloc();
      head->parent=NULL;
      head->r_son=NULL;
      head->l_son=root;
      while(1){
            printf("What do you want me to do (if=insert from
file,\n");
            printf("i=insert from keybord,df=delete from file,
f=delete,\n");
            printf("s=search, q=quit) ?\n");
          scanf("%s",c);
          switch(c[0]){
                case 'i' : if(c[1]=='f'){
                                    printf("Input file:");
                                    scanf("%s",fil);
                                    fp=fopen(fil,"r");
                                    while((i=fscanf(fp,"%d
",&k))!=0 && i!=EOF)
                                          insert(k);
                                  fclose(fp);
                                    break;
                                }
                               else if(c[1]==0){
                                    printf("What ?\n");
                                    scanf("%d",&k);
                                    insert(k);
                                    break;
                                }
                  case 'd' : if(c[1]=='f'){
                                    printf("Input file:");
                                    scanf("%s",fil);
                                    fp=fopen(fil,"r");

while((i=fscanf(fp,"%d",&k))!=0 && i!=EOF)
                                          del(k);
                                  fclose(fp);
```

```
                                        break;
                                    }
                                else if(c[1]==0){
                                        printf("Which ?\n");
                                        scanf("%d",&k);
                                        del(k);
                                        break;
                                    }
                    case 's' : printf("Who ?\n");
                                    scanf("%d",&k);
                                    l=search(k);
                                    if(found)
                                        printf("%d : Found\n",l-
>value);
                                    else
                                        printf("%d : Not Found. Next
element :%d\n",k,l->value);
                                    break;
                    case 'q' : break;
                    default  : break;
              }
              if(c[0]=='q') break;
              symord(root);
              printf("The height is :%d\n",measure(root));
            printf("The root is %d\n",root->value);
       }
}

void symord(Node *v){
       if(v->l_son==NULL && v->r_son==NULL)
             printf("%d : %g\n",v->value,v->possibility);
       else{
             if(v->l_son!=NULL){
                 symord(v->l_son);
             }
             printf("%d : %g\n",v->value,v->possibility);
             if(v->r_son!=NULL){
                 symord(v->r_son);
             }
       }
}

/* Metraei to ypsos toy dendroy */
int measure(Node *v){
       int k;
       int r;

       if(v!=NULL){
             if(v->l_son==NULL && v->r_son==NULL){
                   return 0;
             }
             else{
                   k=measure(v->l_son)+1;
                   r=measure(v->r_son)+1;
                 if(k>r) return k;
                   else return r;
             }
       }
     return 0;
}
```

Χρησιμοποιήσαμε το πρόγραμμα αυτό για να σχηματίσουμε δένδρο για τα στοιχεία ενός αρχείου File με 128 στοιχεία. Μετά μετρήσαμε το ύψος του δένδρου και το βρήκαμε ίσο με **13.** Μετά αφαιρούμε 40 στοιχεία από το δένδρο και ξαναμετρούμε το ύψος. Τώρα το βρίσκουμε ίσο με **9.**
Παρατηρούμε ότι μετά από τις διαγραφές των στοιχείων άλλαξε πάρα πολύ η δομή του δένδρου. Επίσης βλέπουμε ότι οι τιμές που έχουμε για τις προτεραιότητες παίζουν πρωταρχικό ρόλο στις περιστροφές και την κατασκευή του δένδρου γενικότερα.

Άσκηση 2η

Στην άσκηση αυτή πρέπει να υλοποιήσουμε τις διαδικασίες Find και Select των για arrays των 2000 αριθμών με τυχαία σειρά οι οποίοι είναι ανά 2 διαφορετικοί μεταξύ τους. Αυτό σε C γίνεται με τα παρακάτω προγράμματα - συναρτήσεις :

```
int find(int *M,int i,int l,int r){
      int s,k,Num,j;
      extern int counter;

      j=l;
      k=r+1;
      s=M[l];
      counter++;
      while(j<k){
            do{
                  j++;
                  counter++;
            } while(!(M[j]>=s));
            do{
                  k--;
                  counter++;
      } while(!(M[k]<=s));
            counter++;
            if(k>j) swap(M,k,j);
            counter++;
      }
      Num=k-l;
      counter++;
      if(Num<i-1)
            return find(M,i-Num-1,k+1,r);
    counter++;
      if(Num==i-1)
            return s;
      counter++;
      if(Num>i-1)
            return find(M,i,l+1,k-1);
      return 0;
}


int select(int *v,int i,int l,int r){
      int n,j,m,max,k;
      int Medians[405];
```

```
	extern int counter;

	n=r-l+1;
	counter++;
	if(n<100){
		mergesort(v,l,r);
		return v[l+i-1];
	}
	max=(int) floor(n/5.0);
	counter++;
	for(k=0;k<max;k++){
		counter++;
		mergesort(v,l+k*5,l+(k+1)*5-1);
		Medians[k]=v[l+k*5+3];
	}
	counter++;
	if(l+max*5-1<r){
		mergesort(v,l+max*5,r);
		Medians[max]=v[r-(int) ceil((r-l-max*5+1)/2.0)];
	}
	m=select(Medians,(int) ceil(n/10.0),0,max);
	j=l-1;
	k=r+1;
	counter++;
	while(j<k){
		do{ counter++;
			j++;
		} while(v[j]<m && j<r);
		do{ counter++;
		    k--;
		} while(v[k]>m && k>l);
		counter++;
	    if(k>j) swap(v,k,j);
	counter++;
}
counter++;
if(k-l+1>=i)
	return select(v,i,l,k);
else
	return select(v,i-k+l-1,k+1,r);
```

Στις δύο παραπάνω συναρτήσεις η συνάρτηση mergesort(A,p,r) είναι η συνάρτηση με την οποία υλοποιείται ο αλγόριθμος ταξινόμησης mergesort. Επίσης, η συνάρτηση swap(A,p,r) εκτελεί εναλλαγή των στοιχείων p και r του array A.

Υλοποιήσαμε τις παραπάνω συναρτήσεις σε προγράμματα (όπως αναφέρεται στην ενότητα ***Γενικά)*** και τα εκτελέσαμε για ένα αρχείο 2000 στοιχείων ψάχνοντας για $i \in \{1, 2000, 1500, 700\}$. Πήραμε τα παρακάτω αποτελέσματα :

	1	**700**	**1500**	**2000**
Find	*3306*	*8521*	*10128*	*8888*
Select	*12543*	*10581*	*7526*	*6026*

Από αυτά τα αποτελέσματα επιβεβαιώνεται ότι οι δύο αλγόριθμοι έχουν γραμμικό χρόνο. Επίσης φαίνεται ότι οι δύο υλοποιήσεις έχουν περίπου την ίδια πολυπλοκότητα, αν και εμείς ξέρουμε ότι σε κάποια χειρότερη περίπτωση ο

αλγόριθμος *Select* θα τελειώσει γρηγορότερα από τον *Find.*

Αν και οι αλγόριθμοι Quicksort και Find βασίζονται αμφότεροι στην Αρχή Divide & Conquer, παρόλα αυτά ο Find έχει μέσο χρόνο O(n) ενώ ο Quicksort έχει μέσο χρόνο O(nlogn). Αυτό οφείλεται στο εξής :
Πρόκειται για δύο διαφορετικές εφαρμογές της μεθόδου "Divide & Conquer". Ο Quicksort χρησιμοποιεί την Αρχή αυτή για να διαιρέσει το αρχικό πρόβλημα σε δύο μικρότερα υποπροβλήματα, αλλά στη συνέχεια οφείλουν να λυθούν και τα 2 αυτά νέα υποπροβλήματα προκειμένου να φτάσουμε στη λύση του αρχικού προβλήματος. Αντίθετα, ο Find διαιρεί το αρχικό πρόβλημα σε 2 υποπροβλήματα, αλλά από αυτά επιλέγει και λύνει **μονάχα** το ένα κάθε φορά, αποφεύγοντας έτσι να κάνει περιττές πράξεις.
Άρα, για τον αλγόριθμο Quicksort, εφόσον απαιτείται η λύση και των δύο υποπροβλημάτων η αναδρομική εξίσωση θα έχει τη μορφή :

$$T(i) = 2T(i-1) + E(i),$$

όπου i είναι ο αριθμός του υποπροβλήματος που έχω να λύσω με i=1 στο τελευταίο υποπρόβλημα που έχω να λύσω και i=logn στο αρχικό που πρόβλημα. (Αφού διαιρώντας το αρχικό μου πρόβλημα σε 2 υποπροβλήματα, μπορώ να έχω μέχρι logn υποπροβλήματα). Επίσης, E(i) καλούμε τον χρόνο που χρειάζεται για να βρεθεί η λύση του αρχικού προβλήματος από τα δύο υποπροβλήματα.
Για τον Find, τα πράγματα είναι διαφορετικά.. Λύνοντας μόνο το ένα υποπρόβλημα κάθε φορά, παίρνω μια σχέση της μορφής :

$$T(i) = T(i-1) + E(i),$$

όπου τα σύμβολα έχουν την ίδια σημασία όπως προηγούμενα. Είναι εμφανές ότι ο Quicksort πρέπει να κάνει περισσότερα βήματα για να λύσει ένα πρόβλημα μεγέθους n απ' ότι ο Find. Μάλιστα στη μέση περίπτωση όπου τα δύο υποπροβλήματα έχουν περίπου το ίδιο μέγεθος, φαίνεται ότι ο Quicksort πρέπει να κάνει logn φορές τα βήματα του Find.

Ο αλγόριθμος Find μπορεί με μικρές αλλαγές να χρησιμοποιηθεί για να λύσει και άλλα προβλήματα. Υποθέτουμε ότι έχουμε ένα array A[1..n] που περιέχει όλους τους ακέραιους από 0 μέχρι n εκτός από έναν. Υποθέτουμε ότι δεν μπορούμε να προσπελάσουμε έναν αριθμό ολόκληρο με μια πράξη, αλλά μόνο το j-οστό bit του αριθμού κάθε φορά. Χρησιμοποιώντας μια παραλλαγή του Find μπορούμε να βρούμε σε γραμμικό χρόνο ποιος είναι ο αριθμός που λείπει. Για ευκολία, και χωρίς να βλάπτουμε την γενικότητα, θεωρούμε ότι το n είναι δύναμη του 2, δηλαδή $n=2^p$.

Έτσι, σκεπτόμαστε ως εξής :

Ξεκινούμε από το MSB των αριθμών.

1. Τους χωρίζουμε σε ομάδες M_0 και M_1 ανάλογα αν το bit υπό εξέταση είναι 0 ή 1.
2. Αν $|M_0| > |M_1|$ σημαίνει ότι ο αριθμός που λείπει έχει 1 στη θέση bit που εξετάζουμε, άρα βάζω 1 στο bit της θέσης αυτής στον αριθμό που θα επιστρέψει ο αλγόριθμος και θα είναι αυτός που λείπει. Διαφορετικά, ο αριθμός που λείπει έχει 0 στο bit αυτής της θέσης και άρα βάζω 0 στο αποτέλεσμα στη συγκεκριμένη θέση bit.
3. Αν δεν βρίσκομαι ήδη στο LSB των αριθμών μου, παίρνω το επόμενο bit και πηγαίνω στο βήμα 1 εφαρμόζοντας ξανά τον αλγόριθμο για το υποσύνολο M_x με τα λιγότερα στοιχεία.

Έστω, λοιπόν ότι έχω τους αριθμούς

1. 000
2. 001
3. 010
4. 011
5. 100
6. 101
7. 110
8. 111

και έστω ότι στο array [1..7] έχω τα νούμερα

1. 000
2. 001
3. 011
4. 100
5. 101
6. 110
7. 111

δηλαδή λείπει ο αριθμός 010.
Εφαρμόζω τον αλγόριθμο και ξεκινώ από το 1^o bit. Βλέπω ότι οι αριθμοί με 0 στο bit υπό εξέταση είναι 3 (οι 000, 001 και 011) ενώ οι αριθμοί με 1 είναι 4 (οι 100, 101, 110 και 111). Άρα το $|M_0|=3$ και $|M_1|=4$ και ο αριθμός που λείπει έχει **0 στο 1^o bit.**
Εφόσον δεν βρίσκομαι στο τελευταίο bit συνεχίζω. Εξετάζω τώρα το 2^o bit εφαρμόζοντας τον ίδιο αλγόριθμο μόνο στην ομάδα M_0.
Βλέπω ότι οι αριθμοί με 0 στο 2^o bit είναι 2 (οι 000 και 001) ενώ με 1 στο 2^o bit είναι μόνο ο 011. Άρα τώρα $|M_1|=1$ και $|M_0|=2$. Άρα ο αριθμός που λείπει έχει **1 στο 2^o bit.**
Εφόσον δεν βρίσκομαι στο τελευταίο bit συνεχίζω. Εξετάζω τώρα το 3^o bit εφαρμόζοντας τον ίδιο αλγόριθμο μόνο στην ομάδα M_1. Βλέπω ότι 1 στο 3^o bit έχει ο αριθμός 011 ενώ 0 στο 3^o bit δεν έχει κανένας αριθμός. Άρα $|M_0|=0$ και $|M_1|=1$.
Επομένως, ο αριθμός που λείπει έχει **0 στο 3^o bit.** Βρίσκομαι στο τελευταίο bit των αριθμών μου, άρα σταματώ εδώ.
Έτσι έχω βρει ότι ο αριθμός που λείπει είναι ο : **010** που είναι σωστό.

Αν οι αριθμοί μου έχουν k bit και έχω n-1 αριθμούς στο διάστημα [0, n-1] τότε :

1. Στο 1^o bit κάνω n-1 συγκρίσεις.
2. Στο 2^o bit κάνω $\frac{n}{2}-1$ συγκρίσεις.
3. Στο 3^o bit κάνω $\frac{n}{4}-1$ συγκρίσεις
4. Στο 4^o bit κάνω $\frac{n}{8}-1$ συγκρίσεις

.....
.....

k. Στο k-οστό bit κάνω $\frac{n}{2^{k-1}}$ συγκρίσεις.

Άρα όλες οι συγκρίσεις είναι :

$$T(n) = \sum_{i=1}^{i=k} \frac{n}{2^{i-1}} = \alpha \bullet n = O(n)$$

δηλαδή ο αλγόριθμος που διατυπώσαμε πριν **έχει γραμμική πολυπλοκότητα.**

Άσκηση 3η

Στην άσκηση αυτή πρέπει να υλοποιήσουμε τις διαδικασίες (i) Binary Search, (ii) binary interpolation Search και (iii) alternate. Στη συνέχεια εφαρμόζουμε τον κάθε αλγόριθμο σε ένα file, που ονομάζεται Search_Test και περιέχει 2000 στοιχεία σε αύξουσα σειρά, τον κάθε αλγόριθμο ψάχνοντας για τα στοιχεία 7, 25, 30, 68, 100, 155, 330, 667, 1001, 1330, 1700, 1821, 1917, 1973.

Οι αλγόριθμοι αυτοί υλοποιημένοι σε C είναι οι εξής :

A]Binary Search

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <math.h>

int binsearch(int *,int,int,int);
int counter;

void main(int argc, char *argv[])
{
      extern int counter;
      int k,NumOfEls,x;
      int v[2003];
```

```
      FILE *fp;

      if(argc==1) {
            printf("Select. Syntax :\n");
            printf("     select input_file x\n");
            printf("input_file : file to be searched\n");
            printf("x : Find the xth smaller element in the
input_file\n");
            return;
      }
    if(argc>3) {
            printf("Too many arguments. Exiting ...\n");
          return;
      }
    fp=fopen(*++argv,"r");
      v[0]=INT_MIN;
      NumOfEls=1;
      for(k=10;k!=EOF;){
            k=fscanf(fp,"%d ",&v[NumOfEls]);
          if(k!=0 && k!=EOF) NumOfEls++;
      }
      fclose(fp);
      v[NumOfEls]=INT_MAX;
      x=binsearch(v,atoi(*++argv),1,NumOfEls-1);
      if(x==-1){
            printf("Element not found.\n");
            return;
      }
      printf("Found. The element is %d in the row\n",x);
      printf("Number of Comparisons :%d\n",counter);
}

int binsearch(int *v,int x,int l,int r){
      int next,find;
    extern int counter;

      next=(int) floor((l+r)/2);
      find=0;
      counter++;
      while(x!=v[next] && r>l){
            counter++;
            if(x<v[next]) r=next-1;
            else l=next+1;
            next=(int) floor((l+r)/2);
            counter++;
            if(x==v[next]){
                  find=1;
                  break;
            }
      counter++;
      }
      if(find==1) return next;
      else return -1;
}
```

B] Binary Interpolation Search

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <math.h>

int BIsearch(int *,int,int,int);
int counter;

void main(int argc, char *argv[])
{
      extern int counter;
      int k,NumOfEls,x;
      int v[2003];
      FILE *fp;

      if(argc==1) {
            printf("Select. Syntax :\n");
            printf("     select input_file x\n");
            printf("input_file : file to be searched\n");
            printf("x : Find the xth smaller element in the
input_file\n");
            return;
      }
    if(argc>3) {
            printf("Too many arguments. Exiting ...\n");
          return;
      }
    fp=fopen(*++argv,"r");
      v[0]=INT_MIN;
      NumOfEls=1;
      for(k=10;k!=EOF;){
            k=fscanf(fp,"%d ",&v[NumOfEls]);
          if(k!=0 && k!=EOF) NumOfEls++;
      }
      fclose(fp);
      v[NumOfEls]=INT_MAX;
      k=atoi(*++argv);
      if(k<v[NumOfEls-1] && k>v[1]){
            x=BIsearch(v,k,1,NumOfEls-1);
      }
      else x=0;
      if(x==0){
            printf("Element not found.\n");
            return;
      }
      printf("The element is %d in the row\n",x);
      printf("Number of Comparisons :%d\n",counter);
}

int BIsearch(int *v,int y,int l,int r){
      int next,find,i,step;
      extern int counter;

      v[l-1]=0;
      v[r+1]=1;
      next=(int) ceil( (y-v[l])*(r-l) / (v[r]-v[l]) ) + l;
      find=0;
      step=(int) sqrt(r-l+1);
```

```
      counter++;
      if(y==v[next])
            return next;
      counter++;
      if(r-l==1&&v[next-1]!=y) return 0;
      counter++;
      if(y>v[next]){
            i=1;
            counter++;
            while(y>v[next+i*step]){
                  i++;
                  counter++;
            }
            return BIsearch(v,y,next+(i-1)*step,next+i*step);
      }
      else{
            i=1;
            counter++;
            while(y<v[next-i*step]){
                  i++;
                  counter++;
            }
            return BIsearch(v,y,next-i*step,next-(i-1)*step);
      }
    return 0;
}
```

Γ] Alternate

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <math.h>

int binsearch(int *,int,int,int);
int BIsearch(int *,int,int,int);
int counter;

void main(int argc, char *argv[])
{
      extern int counter;
      int k,NumOfEls,x;
      int v[2003];
      FILE *fp;

      if(argc==1) {
            printf("Select. Syntax :\n");
            printf("     select input_file x\n");
            printf("input_file : file to be searched\n");
            printf("x : Find the xth smaller element in the
input_file\n");
            return;
      }
    if(argc>3) {
            printf("Too many arguments. Exiting ...\n");
```

```
            return;
          }
        fp=fopen(*++argv,"r");
          v[0]=INT_MIN;
          NumOfEls=1;
          for(k=10;k!=EOF;){
                k=fscanf(fp,"%d ",&v[NumOfEls]);
            if(k!=0 && k!=EOF) NumOfEls++;
          }
          fclose(fp);
          v[NumOfEls]=INT_MAX;
          k=atoi(*++argv);
          if(k<=v[NumOfEls-1]&&k>=v[1]){
                x=binsearch(v,k,1,NumOfEls-1);
          }
          else x=-1;
          if(x==-1){
                printf("Element not found.\n");
                return;
          }
          printf("Found. The element is %d in the row\n",x);
          printf("Number of Comparisons :%d\n",counter);
}

int binsearch(int *v,int x,int l,int r){
          int next,find;
        extern int counter;

          next=(int) floor((l+r)/2);
          find=0;
          counter++;
          if(x!=v[next]){
                if(x<v[next]) r=next-1;
            else
                  if(x>v[next])
                        l=next+1;
        }
          else{
                find=1;
          }
          if(find==1) return next;
          else{
                if(l<r) return BIsearch(v,x,l,r);
                else return -1;
          }
}




int BIsearch(int *v,int x,int l,int r){
          int next,find,i,step;
          extern int counter;

          v[l-1]=0;
          v[r+1]=1;
          next=(int) ceil( (x-v[l])*(r-l) / (v[r]-v[l]) ) + l;
          find=0;
          step=(int) sqrt(r-l+1);
          counter++;
          if(x==v[next])
```

```
            return next;
      counter++;
      if(r-l==1&&v[next-1]!=x) return -1;
      counter++;
      if(x>v[next]){
            i=1;
            counter++;
            while(x>v[next+i*step]){
                  i++;
                  counter++;
            }
            return binsearch(v,x,next+(i-1)*step,next+i*step);
      }
      else{
            i=1;
            counter++;
            while(x<v[next-i*step]){
                  i++;
                  counter++;
            }
            return binsearch(v,x,next-i*step,next-(i-1)*step);
      }
}
```

Τρέξαμε αυτά τα προγράμματα για ένα file 2000 στοιχείων ψάχνοντας για τα στοιχεία

7, 25, 30, 68, 100, 155, 330, 667, 1001, 1330, 1700, 1821, 1917, 1973. Κάθε φορά μετρούσαμε τον απαιτούμενο αριθμό πράξεων. Τα αποτελέσματα που πήραμε φαίνονται στον παρακάτω πίνακα :
Binary Search
B.I.Search
Alternate
7
24
13
11 **25** *30* *17* *7* **30** *27* *5* *7* **68** *24* *5* *7* **100** *30* *9* *11* **155** *3*
0 *13* *12* **330** *30* *9* *12* **667** *30* *17* *12* **1001** *30* *14* *13* **1330** *15*

14 7 **1700** *30 11 13* **1821** *30 11 8* **1917** *18 11 8* **1973** *30 7 12*