void insertionSort( int \*vector, int vectorSize){

int i;

for( i = 1 ; i < vectorSize ; i ++){

int aux = vector[i];

int j;

for(j = i - 1; j >= 0 && vector[j] > aux ; j--){

vector[j +1] = vector[j];

}

vector[j+1] = aux;

}

void selectionSort ( int \*vector, unsigned int vectorSize){

int fst;

for(fst = 0; fst < vectorSize -1; fst++){

int smlr = fst;

int i;

for(i = fst + 1; i < vectorSize; i ++){

if(vector[smlr] > vector[i]){

smlr = i;

}

}

if(smlr != fst){

int aux = vector[fst];

vector[fst] = vector[smlr];

vector[smlr] = aux;

}

}

}

void criarheap(int \*vet, int i, int f){

int aux = vet[i];

int j=i\*2+1;

while(j<=f){

if(j<f){

if(vet[j] < vet[j+1]){

j=j+1;

}

}

if(aux<vet[j]){

vet[i] = vet[j];

i=j;

j=2\*i+1;

}

else{

j=f+1;

}

}

vet[i] = aux;

}

void heapsort(int \*vet, int n){

int i, aux;

for(i = (n-1)/2 ; i >= 0 ; i--){

criarheap(vet,i,n-1);

}

for(i = n-1 ; i>= 1; i--){

aux = vet[0];

vet[0] = vet[i];

vet[i] = aux;

criarheap(vet,0,i-1);

}

}

void intercalar(int p,int q,int r,int \*v){

int i=p,j=q,k=0,\*w;

w=(int\*)malloc(sizeof(int)\*(r-p));

while(i<q&&j<r){

if(v[i]<=v[j]){

w[k++]= v[i++];

}

else{

w[k++]= v[j++];

}

}

while(i<q){

w[k++]= v[i++];

}

while(j<r){

w[k++]= v[j++];

}

for(i=p;i<r;i++){

v[i]=w[i-p];

}

free(w);

}

void mergesort(int p,int r,int \*v){

int q;

if(p<r-1){

q=(p+r)/2;

mergesort(p,q,v);

mergesort(q,r,v);

intercalar(p,q,r,v);

}

}

void quicksort(int \*v, int p, int t){

int x, down, up, aux;

if(p>=t){

return;

}

x=v[p];

up=t;

down=p;

while(down<up){

while(v[down]<=x && down<t){

down++;

}

while(v[up]>x){

up--;

}

if(down<up){

aux=v[down];

v[down]=v[up];

v[up]=aux;

}

}

v[p]=v[up];

v[up]=x;

quicksort(v, 1, up-1);

quicksort(v, up+1,t);

}

typedef struct{

int key;

}element;

typedef struct{

element \* vector;

unsigned int vectorsize;

unsigned int listsize;

}List;

element \*binarysearchekey(element \*vector, int first ,int last, int skey){

if(last<first){

return NULL;

}

if (last== first){

if (vector[first].key == skey){

return &(vector[first]);

}

else{

return NULL;

}

}

int m = (last + first)\*0.5;

if (vector[m].key == skey){

return &(vector[m]);

}

if(skey < vector[m].key){

return binarysearchekey(vector, first,m - 1, skey);

}

else{

return binarysearchekey(vector , m , last , skey);

}

}

element \*binarysearche( List \* list, int searchekey){

if(list == NULL){

return NULL;

}

return binarysearchekey( list->vector, 0 ,list -> listsize - 1, searchekey );

}