int cautare\_binara(int x, int a[], int n)

{

int m, stanga, dreapta, gasit;

stanga = 0;

dreapta = n-1;

gasit = 0;

while(stanga <= dreapta && (!gasit)) {

m = (stanga + dreapta)/2;

if(a[m] == x)

gasit = 1;

else

if(a[m] < x)

stanga = m+1;

else

dreapta = m-1;

}

return m;

}

void sortare\_prin\_inserare(int n, int a[])

{

int i,j,key;

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

key = a[i];

j=i-1;

while(j>=0 && a[j]>key) {

a[j+1]=a[j];

j--;

}

a[j+1]=key;

}

}

void sortare\_prin\_insertie\_binara(int n, int a[])

{

int i,j,key,stg,dr,m;

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

key = a[i];

stg=0;

dr=n-1;

while(stg <= dr) {

m = (stg+dr)/2;

if(a[m]<key)

stg=m+1;

else

dr=m-1;

}

for(j=i-1; j>=stg; j--) {

a[j+1]=a[j];

}

a[stg]=key;

}

}

int binary\_search(int a[], int x, int stg, int dr) {

int m;

while(stg <= dr) {

m = (stg+dr)/2;

if(a[m]==x)

return m+1;

else if(x>a[m])

stg=m+1;

else

dr=m-1;

}

return stg;

}

void sortare\_prin\_insertie\_binara(int n, int a[])

{

int i,j,key,stg;

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

j=i-1;

key=a[i];

stg=binary\_search(a,key,0,j);

while(j >= stg) {

a[j+1]=a[j];

j--;

}

a[j+1]=key;

}

}

void selection\_sort(int n, int a[]) {

for(int i=0; i<n-1; i++) {

int iMin = i;

for(int j=i+1; j<n; j++) {

if(a[j] < a[iMin])

iMin = j;

}

int temp = a[i];

a[i]=a[iMin];

a[iMin] = temp;

}

}

void bubblesort(int n, int a[]) {

int i, modificat, temp;

do {

modificat = 0;

for(i=0; i<n-1; i++) {

if(a[i] > a[i+1])

{

temp = a[i];

a[i]=a[i+1];

a[i+1]=temp;

modificat = 1;

}

}

} while(modificat);

}

/\*void shakersort(int n, int a[]) {

int j, sus, jos, ultim, temp;

sus = 1;

jos = n-1;

ultim = n-1;

do{

for(j=jos; j>=sus; j--) {

if(a[j-1] > a[j]) {

temp = a[j-1];

a[j-1] = a[j];

a[j] = temp;

ultim = j;

}

}

sus = ultim + 1;

for(j=sus; j<jos-1; j++) {

temp = a[j-1];

a[j-1] = a[j];

a[j] = temp;

ultim = j;

}

jos = ultim - 1;

}while(!(sus>jos));

}\*/

void shakersort(int n, int a[]) {

int i, start, end, swapped, temp;

start = 0;

end = n-1;

swapped = 1;

while(swapped) {

swapped = 0;

for(i=start; i<end; i++) {

if(a[i]>a[i+1]) {

temp = a[i];

a[i] = a[i+1];

a[i+1] = temp;

swapped = 1;

}

}

if(!swapped)

break;

swapped = 0;

end--;

for(i=end-1; i>=start; i--) {

if(a[i]>a[i+1]) {

temp = a[i];

a[i] = a[i+1];

a[i+1] = temp;

swapped = 1;

}

}

start++;

}

}

int shellsort(int n, int a[]) {

int i, j, temp, gap;

for(gap=n/2; gap>0; n/=2) {

for(i=gap; i<n; i++) {

temp = a[i];

for(j=i; j>=gap && a[j-gap] > temp; j-=gap) {

a[j] = a[j-gap];

}

a[j-gap]=temp;

}

}

return 0;

}

int partition(int a[], int start, int end) {

    int pIndex = start;

    int pivot = a[end];

    for(int i=start; i<end; i++) {

        if(a[i] <= pivot ) {

            int temp = a[pIndex];

            a[pIndex] = a[i];

            a[i] = temp;

            pIndex++;

        }

    }

    int temp = a[pIndex];

    a[pIndex] = a[end];

    a[end] = temp;

    return pIndex;

}

void quicksort(int a[], int start, int end) {

    if(start < end) {

        int pIndex = partition(a,start,end);

        quicksort(a,start, pIndex-1);

        quicksort(a,pIndex+1, end);

    }

}

void heapify(int heap[], int curr, int size) {

    int largest = curr;

    int l = 2\*curr + 1;

    int r= 2\*curr + 2;

    if(l<size && heap[l] > heap[largest]) {

        largest = l;

    }

    if(r<size && heap[r] > heap[largest]) {

        largest = r;

    }

    if(largest != curr) {

        int temp = heap[curr];

        heap[curr] = heap[largest];

        heap[largest] = temp;

        heapify(heap, largest, size);

    }

}

void heapSort(int heap[], int size) {

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

        heapify(heap, i, size);

    }

    for(int i = size-1; i > 0; i--) {

        int temp = heap[0];

        heap[0] = heap[i];

        heap[i] = temp;

        heapify(heap, 0, i);

    }

}