CODE1:

#include <iostream>

using namespace std;

/\* run this program using the console pauser or add your own getch, system("pause") or input loop \*/

class bin{

public :

void search(int x,int arr[],int size)

{

int low ,high,mid;

low=0;

high=size-1;

while(low<=high)

{

mid=(high+low)/2;

if(arr[mid]==x)

{

cout<<"Yes it is present at index : "<<mid<<endl;

break;

}

if(arr[mid]<x)

{

low=mid+1;

}

if(arr[mid]>x)

{

high=mid-1;

}

}

}

void find(int arr[],int i,int x)

{

for(int j=0;j<i;j++)

{

if(arr[j]==x)

{

cout<<"Element is present at index "<<j<<endl;

}

}

}

};

int main(int argc, char\*\* argv) {

int arr[10]={10,20,30,40,50,60,70,80,90,100};

int i=sizeof(arr)/sizeof(int);

bin b;

int k,element;

cout<<"Enter your Choice:\n1.Binary search \n2.Linear search"<<endl;

cin>>k;

cout<<"Enter element to search :"<<endl;

cin>>element;

switch(k)

{

case 1:

b.search(element,arr,i);

break;

case 2:

b.find(arr,i,element);

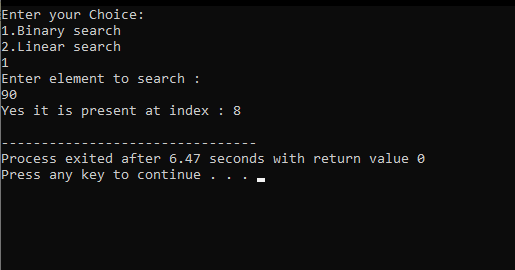
break;

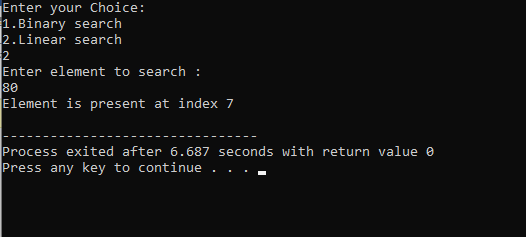
}

return 0;

}

Output:





CODE2:

#include <iostream>

using namespace std;

/\* run this program using the console pauser or add your own getch, system("pause") or input loop \*/

class sorting

{

public:

void bubble\_sort(int arr[],int n)

{

int i, j;

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

// Last i elements are already in place

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

if (arr[j] > arr[j+1])

swapsel(&arr[j], &arr[j+1]);

}

void insert\_sort(int arr[],int n)

{

int i, key, j;

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

{

key = arr[i];

j = i - 1;

/\* Move elements of arr[0..i-1], that are

greater than key, to one position ahead

of their current position \*/

while (j >= 0 && arr[j] > key)

{

arr[j + 1] = arr[j];

j = j - 1;

}

arr[j + 1] = key;

}

}

void swapsel(int \*xp, int \*yp)

{

int temp = \*xp;

\*xp = \*yp;

\*yp = temp;

}

void selection\_sort(int arr[],int n)

{int i, j, min\_idx;

// One by one move boundary of unsorted subarray

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

{

// Find the minimum element in unsorted array

min\_idx = i;

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

if (arr[j] < arr[min\_idx])

min\_idx = j;

// Swap the found minimum element with the first element

swapsel(&arr[min\_idx], &arr[i]);

}

}

void merge\_sort(int a[],int n)

{

int temp[n];

int beg=0;

int end=n-1;

int mid=(beg+end)/2;

int i=0;

int j=mid+1;

int index=0;

while(i<=mid && j<=end)

{

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

{

temp[index]=a[i];

i=i+1;

}

else

{

temp[index]=a[j];

int j=j+1;

}

index++;

}

if(i>mid)

{

while(j<=end)

{

temp[index]=a[j];

j=j+1;

index=index+1;

}

}

else

{

while(i<=mid)

{

temp[index]=a[i];

index=index+1;

i=i+1;

}

}

for(int i=0;i<n;i++)

{

a[i]=temp[i];

}

}

void merge(int array[], int const left, int const mid, int const right)

{

int const subArrayOne = mid - left + 1;

int const subArrayTwo = right - mid;

int \*leftArray = new int[subArrayOne];

int \*rightArray = new int[subArrayTwo];

for (int i = 0; i < subArrayOne; i++)

{

leftArray[i] = array[left + i];

}

for (int j = 0; j < subArrayTwo; j++)

{

rightArray[j] = array[mid + 1 + j];

}

int indexOfSubArrayOne = 0;

int indexOfSubArrayTwo = 0;

int indexOfMergedArray = left;

while (indexOfSubArrayOne < subArrayOne && indexOfSubArrayTwo < subArrayTwo) {

if (leftArray[indexOfSubArrayOne] <= rightArray[indexOfSubArrayTwo])

{

array[indexOfMergedArray] = leftArray[indexOfSubArrayOne];

indexOfSubArrayOne++;

}

else {

array[indexOfMergedArray] = rightArray[indexOfSubArrayTwo];

indexOfSubArrayTwo++;

}

indexOfMergedArray++;

}

while (indexOfSubArrayOne < subArrayOne) {

array[indexOfMergedArray] = leftArray[indexOfSubArrayOne];

indexOfSubArrayOne++;

indexOfMergedArray++;

}

while (indexOfSubArrayTwo < subArrayTwo) {

array[indexOfMergedArray] = rightArray[indexOfSubArrayTwo];

indexOfSubArrayTwo++;

indexOfMergedArray++;

}

}

void mergeSort(int array[], int const begin, int const end)

{

if (begin >= end)

return; // Returns recursivly

int mid = begin + (end - begin) / 2;

mergeSort(array, begin, mid);

mergeSort(array, mid + 1, end);

merge(array, begin, mid, end);

}

void swappar(int\* a, int\* b)

{

int t = \*a;

\*a = \*b;

\*b = t;

}

int partition (int arr[], int low, int high)

{

int pivot = arr[high]; // pivot

int i = (low - 1); // Index of smaller element and indicates the right position of pivot found so far

for (int j = low; j <= high - 1; j++)

{

if (arr[j] < pivot)

{

i++; // increment index of smaller element

swappar(&arr[i], &arr[j]);

}

}

swappar(&arr[i + 1], &arr[high]);

return (i + 1);

}

void quickSort(int arr[], int low, int high)

{

if (low < high)

{

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

void heapify(int arr[], int n, int i)

{

int largest = i;

int l = 2 \* i + 1;

int r = 2 \* i + 2;

if (l < n && arr[l] > arr[largest])

largest = l;

if (r < n && arr[r] > arr[largest])

largest = r;

if (largest != i) {

swap(arr[i], arr[largest]);

heapify(arr, n, largest);

}

}

void heapSort(int arr[], int n)

{

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

heapify(arr, n, i);

for (int i = n - 1; i > 0; i--)

{

swap(arr[0], arr[i]);

heapify(arr, i, 0);

}

}

void show(int n,int a[])

{

cout<<"{";

for(int i=0;i<n;i++)

{

cout<<a[i]<<" ";

}

cout<<"}"<<endl;

}

};

int main(int argc, char\*\* argv) {

int a[10]={10,50,20,30,70,80,40,90,100,60};

int i=sizeof(a)/sizeof(int);

sorting s;

int k,h;

do{

cout<<"Press 1.Bubble sort 2.Insertion sort 3.Selection sort 4.Merge sort 5.Quick sort 6.Heap sort"<<endl;

cin>>k;

switch(k)

{

case 1:

cout<<"Before sorting :"<<endl;

s.show(i,a);

s.bubble\_sort(a,i);

cout<<"After sorting :"<<endl;

s.show(i,a);

break;

case 2:

cout<<"Before sorting :"<<endl;

s.show(i,a);

s.insert\_sort(a,i);

cout<<"After sorting :"<<endl;

s.show(i,a);

break;

case 3:

cout<<"Before sorting :"<<endl;

s.show(i,a);

s.selection\_sort(a,i);

cout<<"After sorting :"<<endl;

s.show(i,a);

break;

case 4:

cout<<"Before sorting :"<<endl;

s.show(i,a);

s.mergeSort(a,0,i-1);

cout<<"After sorting :"<<endl;

s.show(i,a);

break;

case 5:

cout<<"Before sorting :"<<endl;

s.show(i,a);

s.quickSort(a,0,i-1);

cout<<"After sorting :"<<endl;

s.show(i,a);

break;

case 6:

cout<<"Before sorting :"<<endl;

s.show(i,a);

cout<<"After sorting :"<<endl;

s.heapSort(a,i);

s.show(i,a);

break;

}

cout<<"Press 4 to continue"<<endl;

cin>>h;

}while(h==4);

return 0;

}

Output:

