**Practical 1**

**Aim:** Write a program to sort given elements of an array in ascending order using bubble sort. Analyze the time complexity for best, average and worst case.

#include<stdio.h>

#include<conio.h>

int main()

{

int n, j, i, swap;

printf("Enter number of elements\n");

scanf("%d", &n);

int array[n];

printf("Enter %d integers\n", n);

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

{

scanf("%d", &array[i]);

}

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

{

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

{

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

{

swap = array[j];

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

array[j+1] = swap;

}

}

}

printf("Sorted list in ascending order:\n");

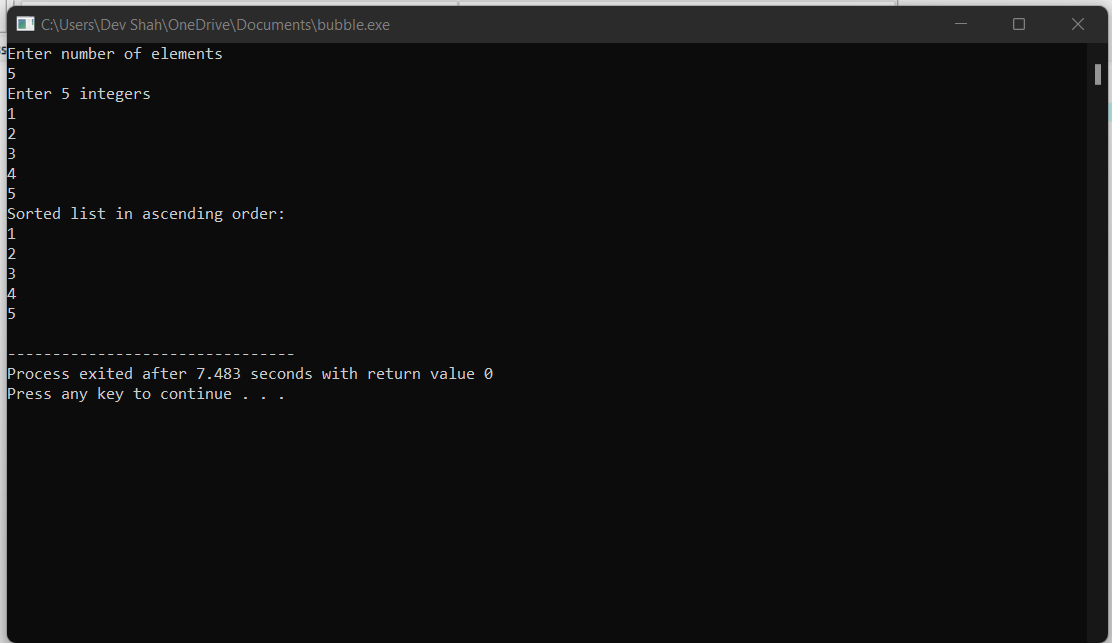
for (i = 0; i < n; i++)

printf("%d\n", array[i]);

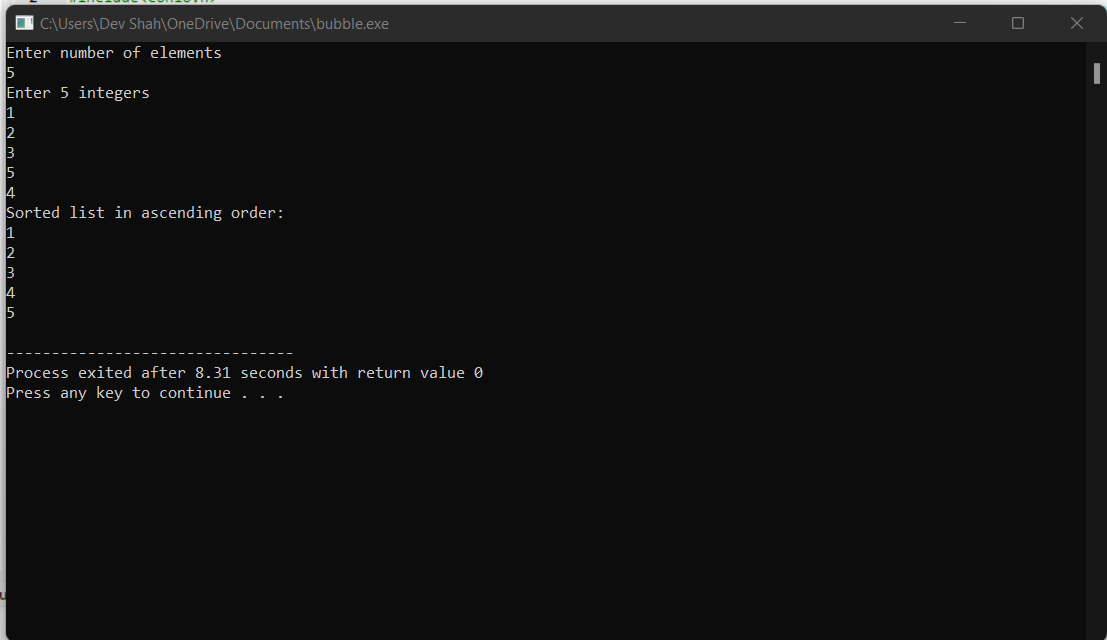
return 0;

}

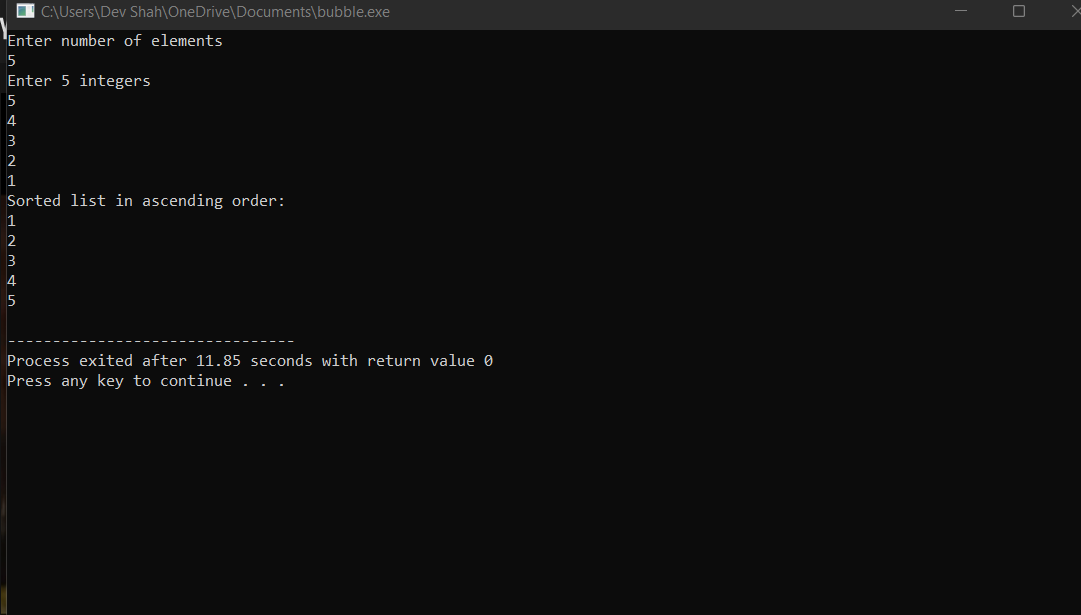
**Output:**

****

**Best case**

****

**Average case**

****

**Worst case**

**Practical 2**

Write a program to sort given elements of an array in ascending order using selection sort. Analyze the time complexity for best, average and worst case.

#include <stdio.h>

void selection(int arr[], int n)

{

int i, j, small;

for (i = 0; i < n-1; i++) // One by one move boundary of unsorted subarray

{

small = i; //minimum element in unsorted array

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

if (arr[j] < arr[small])

small = j;

// Swap the minimum element with the first element

int temp = arr[small];

arr[small] = arr[i];

arr[i] = temp;

}

}

void printArr(int a[], int n) /\* function to print the array \*/

{

int i;

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

printf("%d ", a[i]);

}

int main()

{

int a[] = { 12, 31, 25, 8, 32, 17 };

int n = sizeof(a) / sizeof(a[0]);

printf("Before sorting array elements are - \n");

printArr(a, n);

selection(a, n);

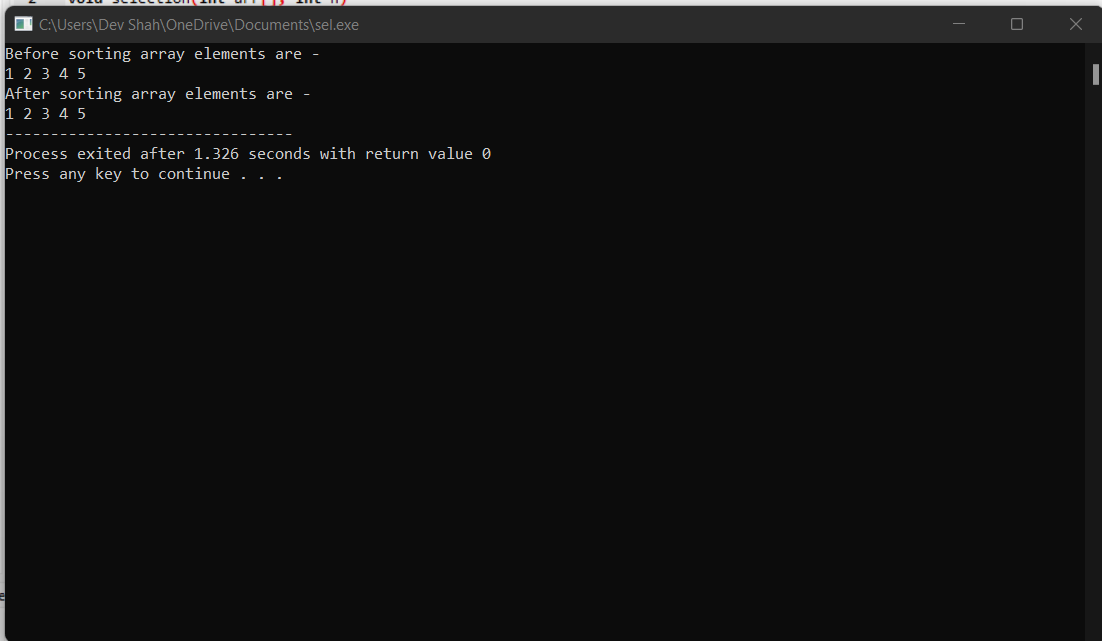
printf("\nAfter sorting array elements are - \n");

printArr(a, n);

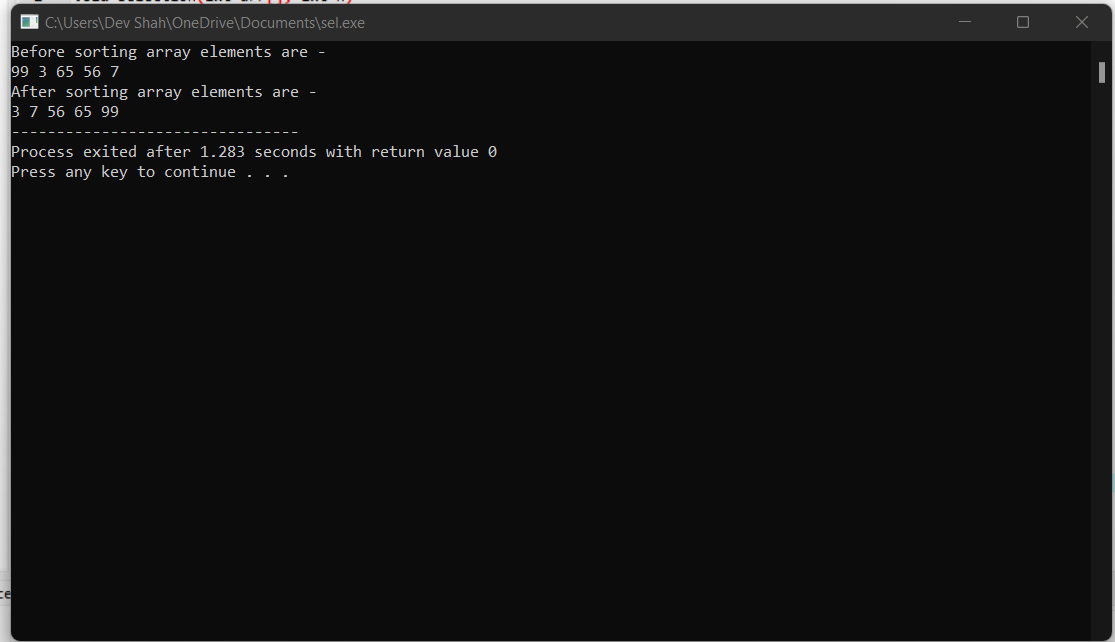
return 0;

}

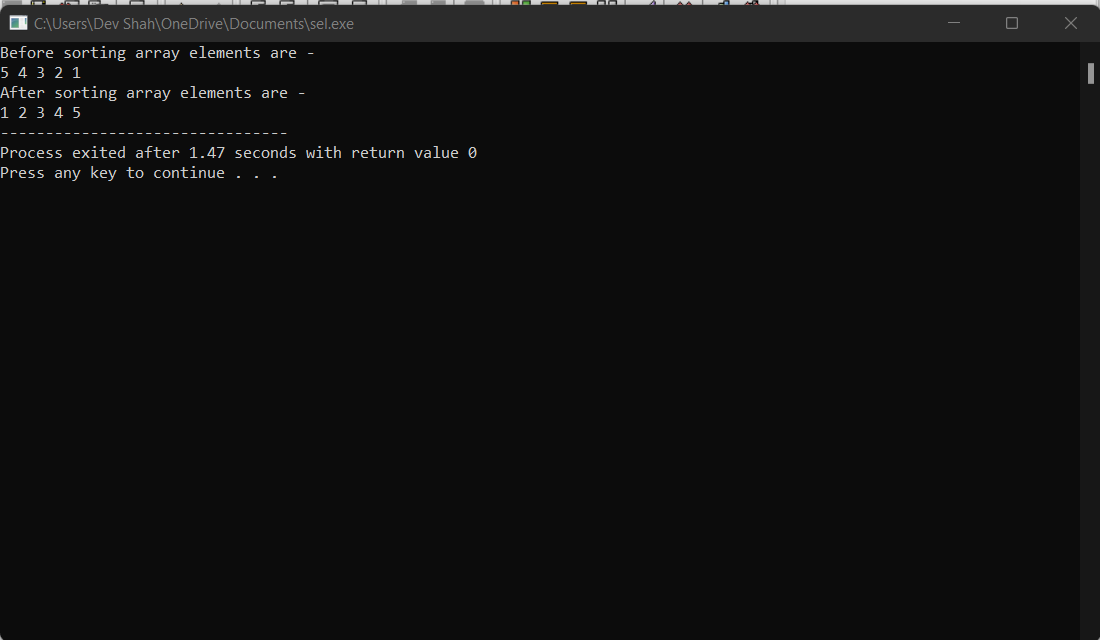
**Output:**

****

**Best case**

****

**Average case**

****

**Worst case**

**Practical 3**

**#include <stdio.h>**

**// Function to swap the the position of two elements**

**void swap(int \*a, int \*b) {**

**int temp = \*a;**

**\*a = \*b;**

**\*b = temp;**

**}**

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

**// Find largest among root, left child and right child**

**int largest = i;**

**int left = 2 \* i + 1;**

**int right = 2 \* i + 2;**

**if (left < n && arr[left] > arr[largest])**

**largest = left;**

**if (right < n && arr[right] > arr[largest])**

**largest = right;**

**// Swap and continue heapifying if root is not largest**

**if (largest != i) {**

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

**heapify(arr, n, largest);**

**}**

**}**

**// Main function to do heap sort**

**void heapSort(int arr[], int n) {**

**// Build max heap**

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

**heapify(arr, n, i);**

**// Heap sort**

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

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

**// Heapify root element to get highest element at root again**

**heapify(arr, i, 0);**

**}**

**}**

**// Print an array**

**void printArray(int arr[], int n) {**

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

**printf("%d ", arr[i]);**

**printf("\n");**

**}**

**// Driver code**

**int main() {**

**int arr[] = {1, 12, 9, 5, 6, 10};**

**int n = sizeof(arr) / sizeof(arr[0]);**

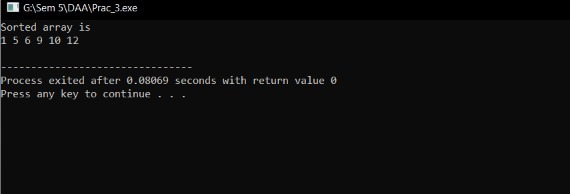
**heapSort(arr, n);**

**printf("Sorted array is \n");**

**printArray(arr, n);**

**}**

**Output:**

****

**Worst case**

**Practical 4**

Write a program to sort given elements of an array in ascending order using insertion sort. Analyze the time complexity for best, average and worst case.

#include <stdio.h>

void insert(int a[], int n)

{

int i, j, temp;

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

temp = a[i];

j = i - 1;

while(j>=0 && temp <= a[j])

{

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

j = j-1;

}

a[j+1] = temp;

}

}

void printArr(int a[], int n)

{

int i;

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

printf("%d ", a[i]);

}

int main()

{

int a[] = { 12, 31, 25, 8, 32, 17 };

int n = sizeof(a) / sizeof(a[0]);

printf("Before sorting array elements are - \n");

printArr(a, n);

insert(a, n);

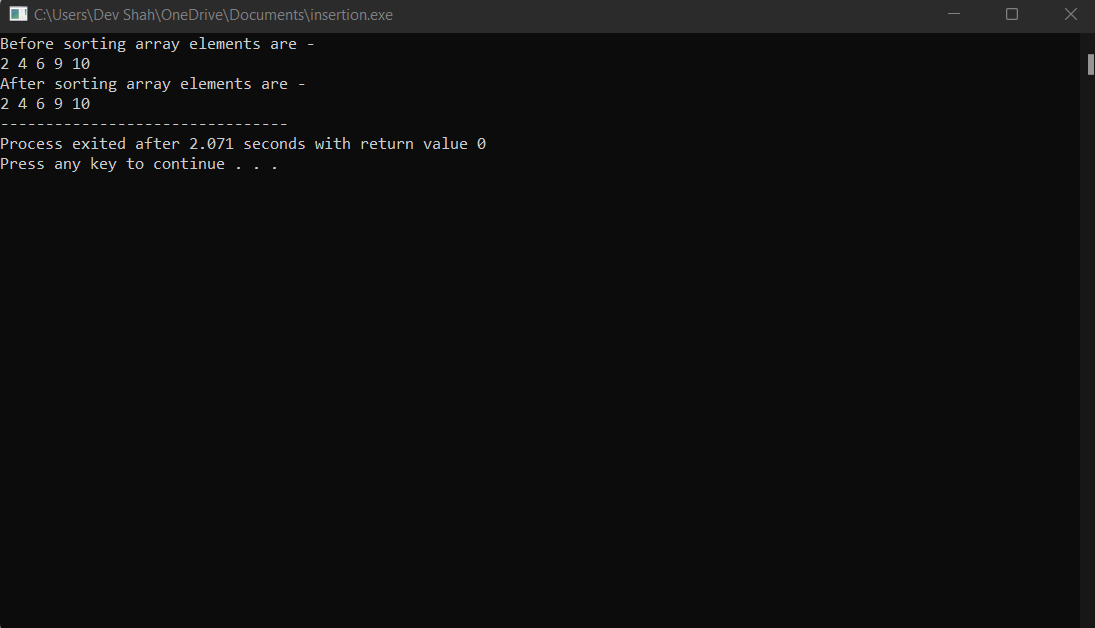
printf("\nAfter sorting array elements are - \n");

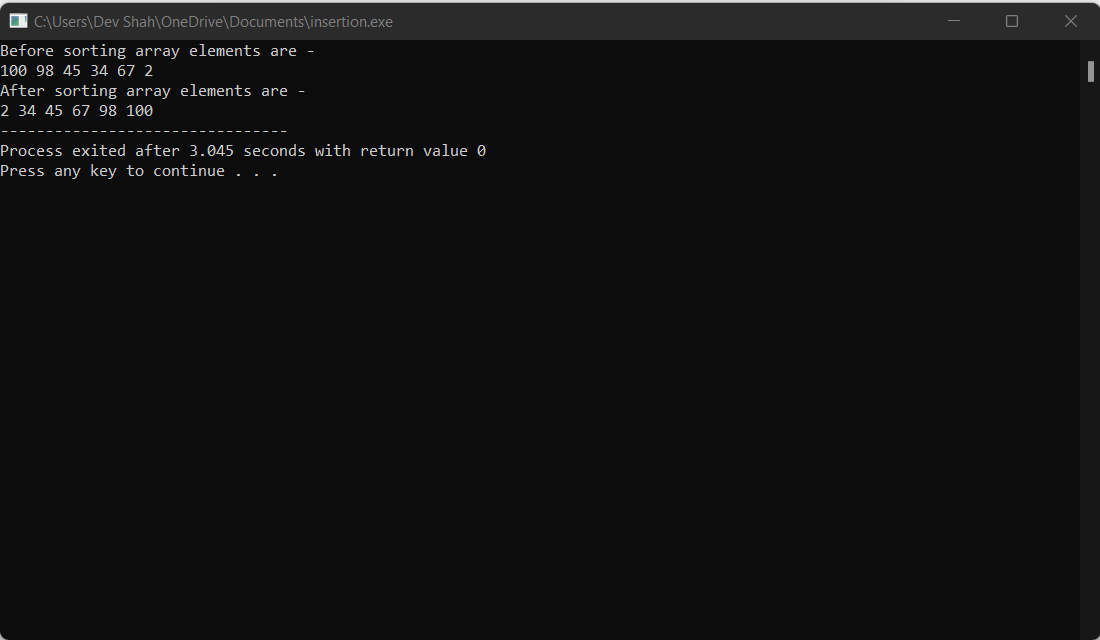
printArr(a, n);

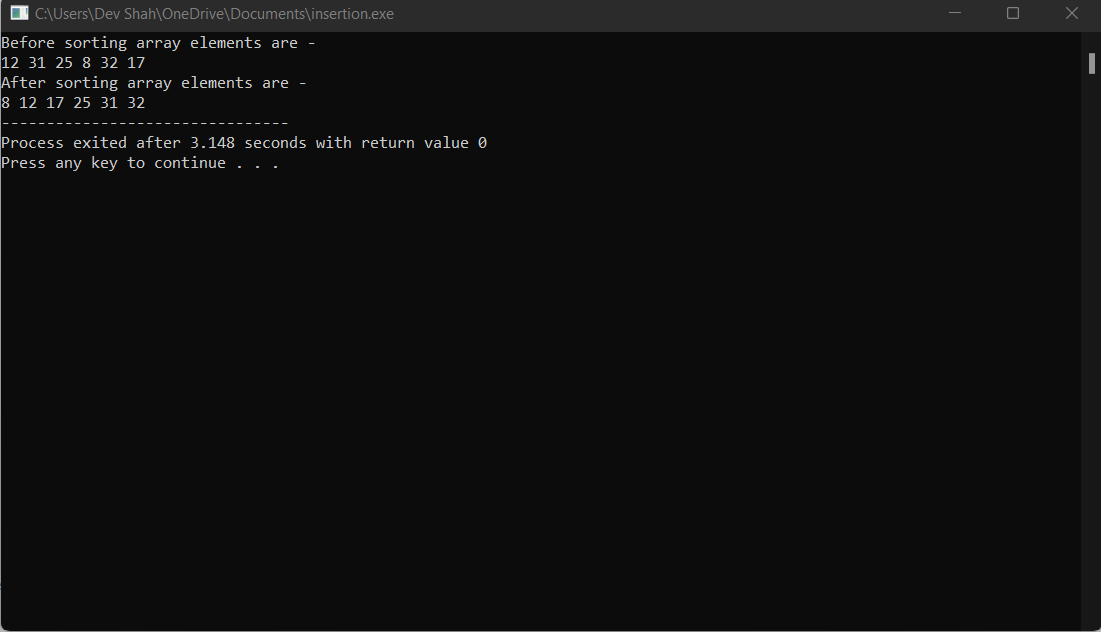
return 0;

}

**Output:**

****

**Best caseAverage case**

****

**Worst case**

**Practical 5**

#include <stdio.h>

void Merge(int \* , int , int , int );

void MergeSort(int \*array, int left, int right){

int middle = (left+right)/2;

if(left<right){

//Sorting the left part

MergeSort(array, left, middle);

//Sorting the right part

MergeSort(array, middle + 1, right);

// Merge the two parts

Merge(array, left, middle, right);

}

}

void Merge(int \*array, int left, int middle, int right){

int tmp[right - left + 1];

int pos = 0, leftposition = left, rightposition = middle + 1;

while (leftposition <= middle && rightposition <= right){

if (array[leftposition] < array[rightposition]){

tmp[pos++] = array[leftposition++];

}

else{

tmp[pos++] = array[rightposition++];

}

}

while (leftposition <= middle)

tmp[pos++] = array[leftposition++];

while (rightposition <= right)

tmp[pos++] = array[rightposition++];

int i;

for (i = 0; i < pos; i++){

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

}

return;

}

int main(){

int size;

printf("\n enter size of array:");

scanf("%d", &size);

int array[size];

int i, j, k;

printf("\n enter the elements in an array:");

for (i = 0; i < size; i++){

scanf("%d", &array[i]);

}

MergeSort(array, 0, size - 1);//calling sort function

for (i = 0; i< size; i++){

printf("%d ", array[i]);

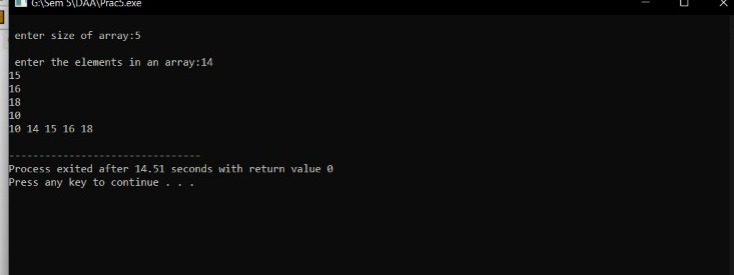
}

printf("\n");

return 0;

}

**Output:**

****

**Worst case**

**Practical 6**

#include<stdio.h>

void quicksort(int [],int,int);

int main(){

int size,i;

printf("Enter size of the array: ");

scanf("%d",&size);

int x[size];

printf("Enter %d elements: ",size);

for(i=0;i<size;i++)

scanf("%d",&x[i]);

quicksort(x,0,size-1);

printf("Sorted elements: ");

for(i=0;i<size;i++)

printf(" %d",x[i]);

return 0;

}

void quicksort(int x[],int first,int last){

int pivot,j,temp,i;

if(first<last){

pivot=first;

i=first;

j=last;

while(i<j){

while(x[i]<=x[pivot]&&i<last)

i++;

while(x[j]>x[pivot])

j--;

if(i<j){

temp=x[i];

x[i]=x[j];

x[j]=temp;

}

}

temp=x[pivot];

x[pivot]=x[j];

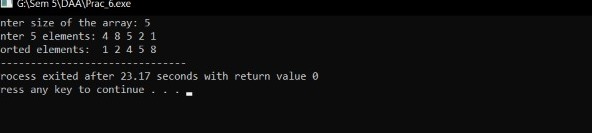
x[j]=temp;

quicksort(x,first,j-1);

quicksort(x,j+1,last);

}

**Output:**



**Worst case**