**INSERTION SORT**

# 1.1 OBJECTIVE:

Write a C Program to implement Insertion Sort.

# 1.2 RESOURCES:

Dev C++

# 1.3 PROGRAM LOGIC:

Insertion sort is a simple sorting algorithm that builds the final sorted array (or list) one item at a time. It is much less efficient on large lists than more advanced algorithms such as quicksort, heapsort, or merge sort.

**Insertion Sort Algorithm**

**Insertion\_Sort(A)**

1. for j=2 to length[A]
2. key=A[j]
3. // Insert A[j] into sorted sequence A[1..j-1]
4. i=j-1
5. whilei>0 and A[i]>key
6. A[i+1]=A[i]
7. i=i-1
8. A[i+1]=key

# 1.4 PROCEDURE:

1. Create: Open Dev C++, write a program after that save the program with .c extension.
2. Compile: Alt + F9
3. Execute: Ctrl + F10

# 1.5 SOURCE CODE:

#include<stdio.h> #include<conio.h> void INSERTION(int a[],int n); //Function declaration void main() void main()

{ int i,a[10],n;

printf("\nProgram for INSERTION SORT\n");

printf("Enter no. of elements:\n");//Input section

scanf("%d",&n); for(i=0;i<n;i++)

{

printf("Enter element%d: ",i); scanf("%d",&a[i]);

}

INSERTION(a,n);//Calling function in main printf("Sorted array:\n");//Output section for(i=0;i<n;i++)

{

printf("%6d",a[i]);

} getch(); }

void INSERTION(int a[],int n)//Function definition

{ int i,j,key;

for(j=1;j<n;j++) // Sorting section

{ key=a[j]; i=j-1;

while((i>=0)&&(a[i]>key)) // Comparison

{

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

} a[i+1]=key;

}

}

**1.6 INPUT/ OUTPUT**



**SHELL SORT**

# 2.1OBJECTIVE:

Write a C Program to implement Shell Sort.

# 2.2RESOURCES:

Dev C++

# 2.3PROGRAM LOGIC:

Quick Sort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. Also known as partition-exchange sort. It works by selecting a 'pivot' element from the array and partitioning the other elements into two sub-arrays, according to whether they are less than or greater than the pivot. The subarrays are then sorted recursively.

**Quick Sort Algorithm**

**QuickSort(A,p,r)**

1.if p<r

1. q=partition(A,p,r)
2. QuickSort(A,p,q-1)
3. QuickSort(A.q+1,r)

**partition(A,p,r)**

1.x=A[r]

2.i=p-1

3.for j=p to r-1

1. if A[j]<=x
2. i=i+1
3. exchange A[i] with A[j]
4. exchange A[i+1] with A[r]

8.return i+1

# 2.4PROCEDURE:

1. Create: Open Dev C++, write a program after that save the program with .c extension.
2. Compile: Alt + F9
3. Execute: Ctrl + F10

# 2.5 SOURCE CODE:

#include <iostream>

using namespace std;

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

int incr=n/2;

while(incr>0){

for(int i=incr; i<n;i++){

int j=i-incr;

while(j>0){

if(a[j]>a[j+incr]){

int temp;

temp=a[j];

a[j]=a[j+incr];

a[j+incr]=temp;

j=j-incr;

}

else{

j=0;

}

}

}

incr=incr/2;

}

}

int main()

{

int n;

cout<<"Enter the elements"<<endl;

cin>>n;

int a[n];

cout<<"Enter elements :"<<endl;

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

cin>>a[i];

}

shell\_sort(a,n);

cout<<"Sorted Elements :"<<endl;

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

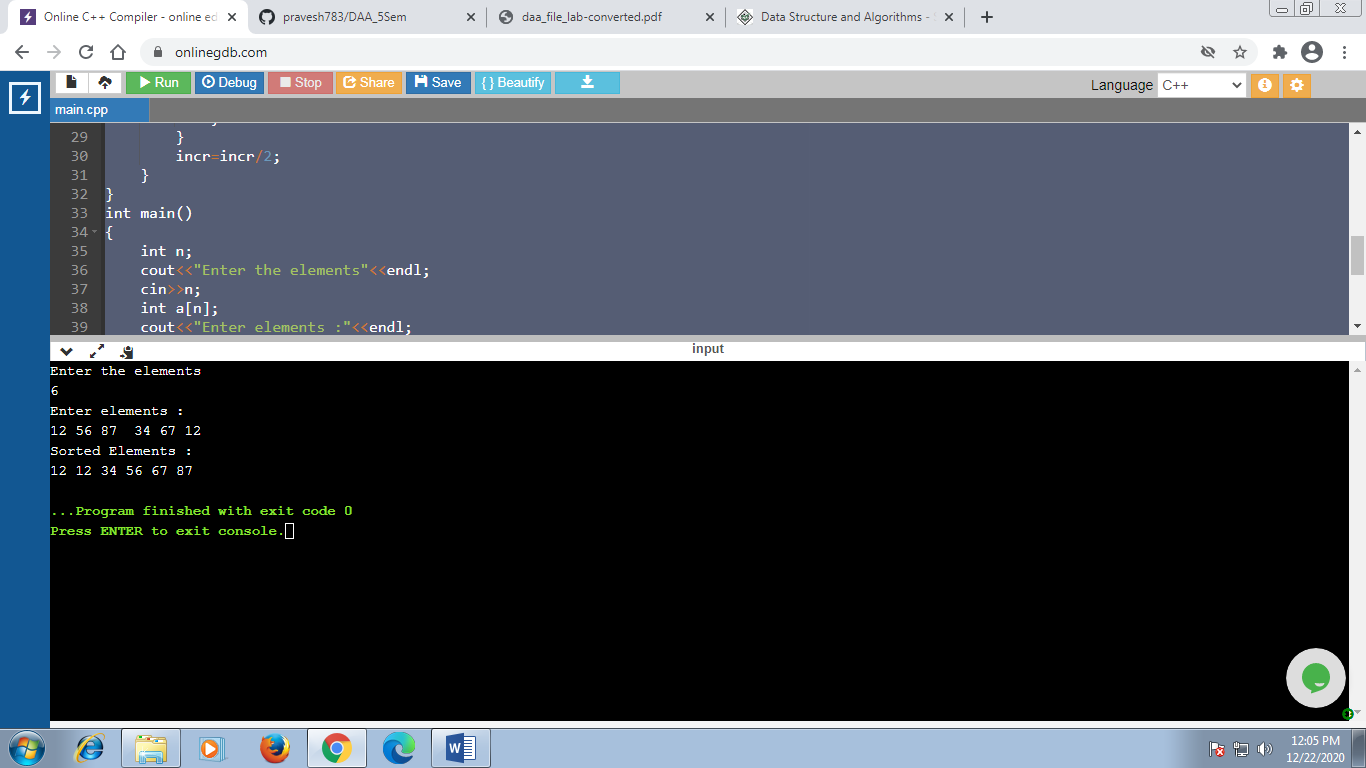
cout<<a[i]<<" ";

}

return 0;

}

**2.6INPUT/ OUTPUT**



# MERGE SORT

# 3.1OBJECTIVE:

Write a C Program to implement Merge Sort.

# 3.2RESOURCES:

Dev C++

# 3.3PROGRAM LOGIC:

Merge sort is one of the most efficient sorting algorithms. It works on the principle of Divide and Conquer. Merge sort repeatedly breaks down a list into several sublists until each sublist consists of a single element and merging those sublists in a manner that results into a sorted list.

**MergeSort Algorithm**

**MergeSort(A,p,r)**

1. if p<r
2. then q=|\_(p+r)/2
3. MergeSort(A,p,q)
4. MergeSort(A,q+1,r)
5. Merge(A,p,q,r)

**Merge(A,p,q,r)**

1.n1=q-p+1

2.n2=r-q

3.create L[1…..n1+1] and R[1…….n2+1]

4.for i =1 to n1

5. L[i]=A[p+i-1]

6.for j=1 to n2

1. R[j]=A[q+j]
2. L[n1+1]= ∞
3. R[n2+1]= ∞

10.i=1

11.j=1

12.for k=p to r

1. if L[i]<=R[j]
2. A[k]=L[i]
3. i=i+1
4. else A[k]=R[j]
5. j=j+1

# 3.4PROCEDURE:

1.Create: Open Dev C++, write a program after that save the program with .cpp extension.

2.Compile: Alt + F9

3.Execute: Ctrl + F10

# 3.5SOURCE CODE:

#include<bits/stdc++.h>

#include <iostream>

using namespace std;

int Merge(int A[],int p,int q,int r);

int Merge(int A[],int p,int q,int r){

int n1,n2; n1=q-p+1; n2=r-q; int L[n1+1],R[n2+1]; for(int i=1;i<=n1;i++){

L[i]=A[p+i-1];

}

for(int j=1;j<=n2;j++){

R[j]=A[q+j];

}

L[n1+1]=INT\_MAX; R[n2+1]=INT\_MAX;

int i=1; int j=1; for(int k=p;k<=r;k++){ if(L[i]<=R[j]){ A[k]=L[i]; i++; } else{ A[k]=R[j]; j++;

}

}

return 0;

}

int mergeSort(int A[],int p,int r){

if (p<r){ int q; q=(p+r)/2; mergeSort(A,p,q); mergeSort(A,q+1,r); Merge(A,p,q,r);

}

return 0;}

int main()

{

int n;

cout<<"Enter the lenghth"<<endl; cin>>n; int A[n+1];

cout<<"Enter the element"<<endl;

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

cin>>A[i];

}

mergeSort(A,1,n);

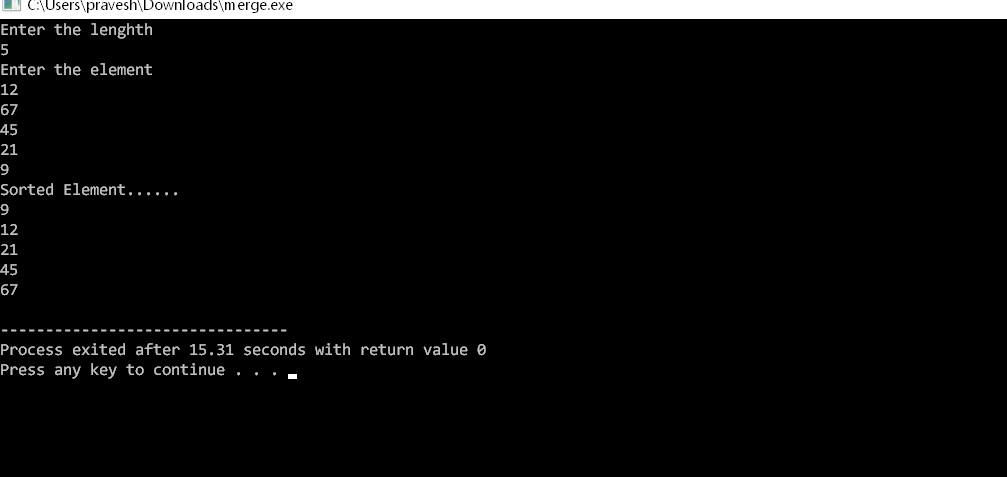
cout<<"Sorted Element......"<<endl; for(int i=1;i<=n;i++){ cout<<A[i]<<endl;

}

return 0;

}

**3.6INPUT/ OUTPUT**



# COUNTING SORT

# 4.1OBJECTIVE:

Write a C Program to implement Counting Sort.

# 4.2RESOURCES:

Dev C++

# 4.3PROGRAM LOGIC:

[Counting sort](http://en.wikipedia.org/wiki/Counting_sort) is a sorting technique based on keys between a specific range. It works by counting the number of objects having distinct key values (kind of hashing). Then doing some arithmetic to calculate the position of each object in the output sequence.

**CountingSort Algorithm**

**Counting\_Sort(A)**

1.Create C[0…….k] be a new array

1. for i=0 to k
2. C[i]=0

4.for j =1 to A.length

5. C[A[j]]=C[A[j]]+1

6.//C[i] now contains the number of element equal to i

7.for i=1 to k

8. C[i]=C[i]+C[i-1]

9.//C[i] now contains the number of element less than or equal to i

10.for j=A.length downto 1

1. B[C[A[j]]]=A[j]
2. C[A[j]]=C[A[j]]-1

# 4.4PROCEDURE:

1.Create: Open Dev C++, write a program after that save the program with .c extension.

2.Compile: Alt + F9

3.Execute: Ctrl + F10

# 4.5SOURCE CODE:

#include<stdio.h>

#include<conio.h>

int k=0; // for storing the maximum element of input array

/\* Method to sort the array \*/

void count\_sort(int A[],int B[],int n)

{

int count[k+1],t,i;

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

{

//Initialize array count

count[i] = 0;

}

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

{

// count the occurrence of elements u of A

// & increment count[u] where u=A[i]

t = A[i];

count[t]++;

}

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

{

// Updating elements of count array

count[i] = count[i]+count[i-1];

}

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

{

// Add elements of array A to array B t = A[i];

B[count[t]] = t;

// Decrement the count value by 1 count[t]=count[t]-1;

}

} int main() { int n,i; int A[20],B[20]; printf("Counting Sort\n");

printf("Enter the no of elements\n");

scanf("%d",&n);

// A is the input array and will store elements entered by the user

// B is the output array having the sorted elements

printf("Enter the array elements:\n "); for(i=0;i<n;i++)

{

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

if(A[i]>k)

{

// k will have the maximum element of A[]

k = A[i];

}

}

count\_sort(A,B,n);

// Printing the elements of array B printf("Sorted element ....\n");

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

{

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

}

return 0;

}

**4.6INPUT/ OUTPUT**

