PRACTICAL NO : - 01

Aim: - Implementation and Time analysis of Bubble , Selection and Insertion sorting algorithm for best case , average case & worst case.

I.Bubble Sort:

Algorithm:

1) Take array as input
2) bs(array[n])
3) for (i = 0 to i = n-1)
4) for (j = 0 to j = n-1)
5) if(array[j] > array[j+1])
6) Swap(array[j], array[j+1])
7) end of loop and print sorted array

Code:

```
#include<
stdio.h>
int
main(){
int
n,i,j,k;
printf("enter the size of
array:");  scanf("%d",&n);
int arr[n];
printf("enter the array
elements:");
for(i=0;i<n;i++){
scanf("%d",&arr[i]);
}
```

```
for(i=0;i<n-1;i++){
for(j=0;j<n-1-i;j++){
if(arr[j]>arr[j+1]){ k=arr[j];
arr[j]=arr[j+1]; arr[j+1]= k;
}
} }  printf("Sorted
array is:");
for(i=0;i<n;i++){
printf("%d\t",arr[i
]);
}
}
```

Output:

1) Best Case:

C:\Users\Shankar\Desktop\DAA\sorted.exe

```
enter the size of array:5
enter the array elements:9
8
7
4
5
Sorted array is:4       5       7       8       9
--------------------------------
Process exited after 6.128 seconds with return value 0
Press any key to continue . . .
```

Time Complexity: (n^2)

2) AVERAGE CASE:

Time Complexity: $O(n^2)$

C:\Users\Shankar\Desktop\DAA\sorted.exe

```
enter the size of array:5
enter the array elements:55
66
22
11
57
Sorted array is:11      22      55      57      66
--------------------------------
Process exited after 16.54 seconds with return value 0
Press any key to continue . . .
```

3) WORST CASE:

Time Complexity: $O(n^2)$

C:\Users\Shankar\Desktop\DAA\sorted.exe

```
enter the size of array:6
enter the array elements:56
89
23
41
47
25
Sorted array is:23      25      41      47      56      89
--------------------------------
Process exited after 15.81 seconds with return value 0
Press any key to continue . . .
```

II Insertion sort:

Algorithm:

- Pick first element and store it in some variable “key”.
- Now compare all array elements with key value.
- if the element of array is smaller than the key value put it to the left side of array; ✞ Else put greater element of array in right side of key value in the array.
- repeat until the array get sorted.
- end of loop.

INPUT:

```
#include<stdio.h> int main(){
int a,i,j,key;
printf("enter the size of array:");  scanf("%d",&a);
int arr[a];  printf("enter the array
elements:");
for(i=0; i<a;i++){
scanf("%d",&arr[i]);
}
for(i=1;i<=a-1;i++){ key
= arr[i]; j= i-1;
while(j>=0 &&
key<=arr[j]){     arr[j+1]
= arr[j];     j--;
} arr[j+1] = key;
}
printf("sorted array is:");
for(i=0;i<a;i++){
printf("%d\t",arr[i]);
}
```

}

OUTPUT:

Best Case:

Time Complexity: O(n)

```
C:\Users\Shankar\Desktop\DAA\ins.exe
enter the size of array:6
enter the array elements:25
41
45
74
85
54
sorted array is:25      41      45      54      74      85
--------------------------------
Process exited after 8.867 seconds with return value 0
Press any key to continue . . .
```

Average Case:

Time Complexity: $O(n^2)$

```
C:\Users\Shankar\Desktop\DAA\ins.exe
enter the size of array:3
enter the array elements:-88
-4
1
sorted array is:-88     -4      1
--------------------------------
Process exited after 7.587 seconds with return value 0
Press any key to continue . . .
```

Worst Case:

Time Complexity: O(n^2)

C:\Users\Shankar\Desktop\DAA\ins.exe

```
enter the size of array:2
enter the array elements:-1
1
sorted array is:-1      1
--------------------------------
Process exited after 17.3 seconds with return value 0
Press any key to continue . . .
```

III.Selection Sort:

Algorithm:

- Select the first element of array also store in a “min” variable and assume that the left side of selected array is sorted and right side of array is unsorted.
- Start comparing all elements of array with the “min” value.
- if element is smaller than the min value and put it to the left of min value in the sorted side of array; else put it to the right side of min value in sorted array, form unsorted array. update the “min” value after getting smaller value than the current min value. Repeat the it until gets sorted array end of loop.

Code:

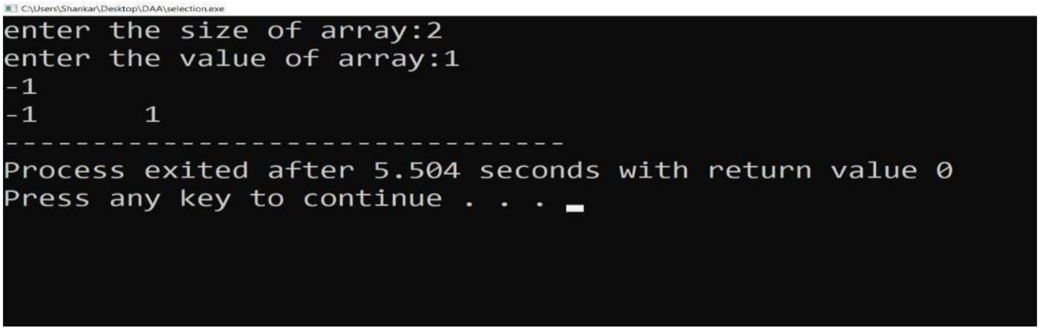
```
#include <stdio.h>

int main()

{int n, i, j, min, t;

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

scanf("%d",&n);

int arr[n];

printf("enter the value of array:", n);
```

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

{scanf("%d", &arr[i]);}

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

{min=i;

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

{if (arr[min] > arr[j])

min=j;

}

if

(min!=i)

{t= arr[i];arr[i] = arr[min];arr[min] = t;

}

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

printf("%d\t", arr[i]);  return 0;

}
```

Output:

Best Case:

Time Complexity: O(n^2)



Average Case:

Time Complexity: O(n^2)

C:\Users\Shankar\Desktop\DAA\selection.exe

```
enter the size of array:4
enter the value of array:0
4
-1
5
-1      0       4       5
--------------------------------
Process exited after 14.03 seconds with return value 0
Press any key to continue . . .
```

Worst Case:

Time Complexity: O(n^2)

C:\Users\Shankar\Desktop\DAA\selection.exe

```
enter the size of array:2
enter the value of array:0
-8888
-8888   0
--------------------------------
Process exited after 7.717 seconds with return value 0
Press any key to continue . . .
```

PRACTICAL NO :- 2

AIM :- Implementation and Time analysis of Max-Heap sort algorithm.

Algorithm :

HeapSort(arr) BuildMaxHea
p(arr) for i = length(arr) to
2 swap arr[1] with arr[i]
heap_size[arr] = heap_size[arr]
? 1 MaxHeapify(arr,1)
End

BuildMaxHeap(arr)

BuildMaxHeap(arr) heap_size(arr)

= length(arr) for i = length(arr)/2

to 1

MaxHeapify(arr,i)

End

MaxHeapi

fy(arr,i)

MaxHeapif

y(arr,i)

L = left(i)
R = right(i) if L ? heap_size[arr] and arr[L] >

arr[i] largest = L else largest = i if R ?

heap_size[arr] and arr[R] > arr[largest]

largest = R if largest != i

swap arr[i] with arr[largest]

MaxHeapify(arr,lar gest)

End

Code :-

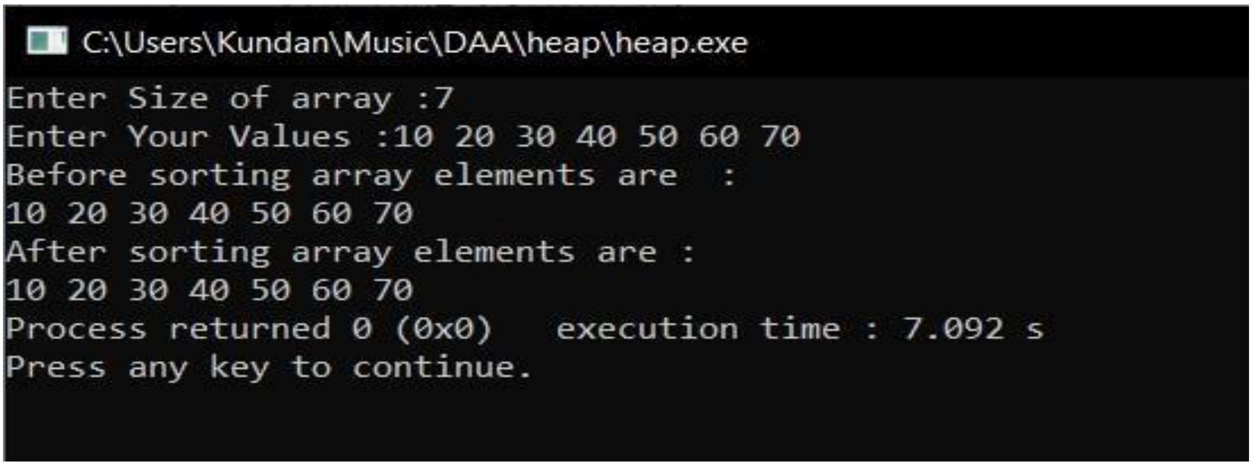
```
#include <stdio.h>
void heapify(int a[], int n, int i)
{
   int largest = i; int left = 2 * i + 1; int right = 2 *
   i + 2; if (left < n && a[left] > a[largest])
{
      largest = left;
}
                      if (right < n && a[right] > a[largest])
{
      largest = right;
}
   if (largest != i)
{
      int temp = a[i]; a[i] = a[largest];
      a[largest] = temp; heapify(a, n,
      largest);
   }
}
void heapSort(int a[], int n)
{
   for (int i = n / 2 - 1; i >= 0; i--)
{
      heapify(a, n, i);
}
   for (int i = n - 1; i >= 0; i--)
{
```

```
        int temp = a[0]; a[0] = a[i];
        a[i] = temp; heapify(a, i, 0);
    }
}
void printArr(int arr[], int n)
{
    for (int i = 0; i < n; ++i)
{
        printf("%d", arr[i]); printf(" ");}
}
int main()
{
    int k,i; printf("Enter Size of array :");
    scanf("%d",&k); printf("Enter Your Values
    :"); int a[k];
for(i=0;i<k;i++)
     { scanf("%d",&a[i]);
}
    int n = sizeof(a) / sizeof(a[0]); printf("Before sorting array
    elements are : \n"); printArr(a, n); heapSort(a, n);
    printf("\nAfter sorting array elements are : \n"); printArr(a,
    n); return 0;
}
```

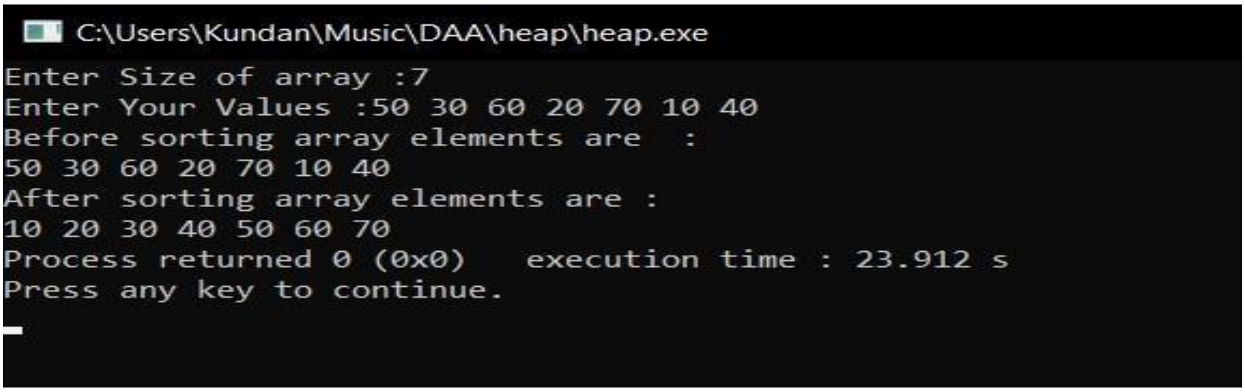
OUTPUT :-

Space Complexity = O(1)

BEST CASE :- Time Complexity = O(n logn)



AVERAGE CASE :- Time Complexity = O(n logn)



WORST CASE :- Time Complexity = O(n logn)

```
C:\Users\Kundan\Music\DAA\heap\heap.exe
Enter Size of array :7
Enter Your Values :70 60 50 40 30 20 10
Before sorting array elements are  :
70 60 50 40 30 20 10
After sorting array elements are :
10 20 30 40 50 60 70
Process returned 0 (0x0)   execution time : 24.505 s
Press any key to continue.
```

PRACTICAL -3

AIM: Implementation and Time Analysis Of Merge Sort Algorithms For Best Case, Average Case &Worst Case Using Divide And Conquer.

Algorithm:

```
  MergeSort (array, lb, ub)
{

if (lb < ub)
    {
      int mid = (lb + ub) / 2;

MergeSort (array, lb, mid);
      MergeSort (array, mid + 1,
ub);
      c_merge (array, lb, mid,
ub);
    }

}
c_merge (array, lb, mid, ub)
{
  i = lb;
  j = mid + 1;
  k = 0;
  while (i <= mid && j <= ub)
    if (array[i] <= array[j])
    b[k++] = array[i++];



    else
      b[k++] = array[j++];
```

```
While (i <= mid) b[k++] =
array[i++];

While (j <= ub)
 b[k++] = array[j++];

For (i = lb, j = 0;
i <= ub;
i++, j++) array[i] = b[i];
```

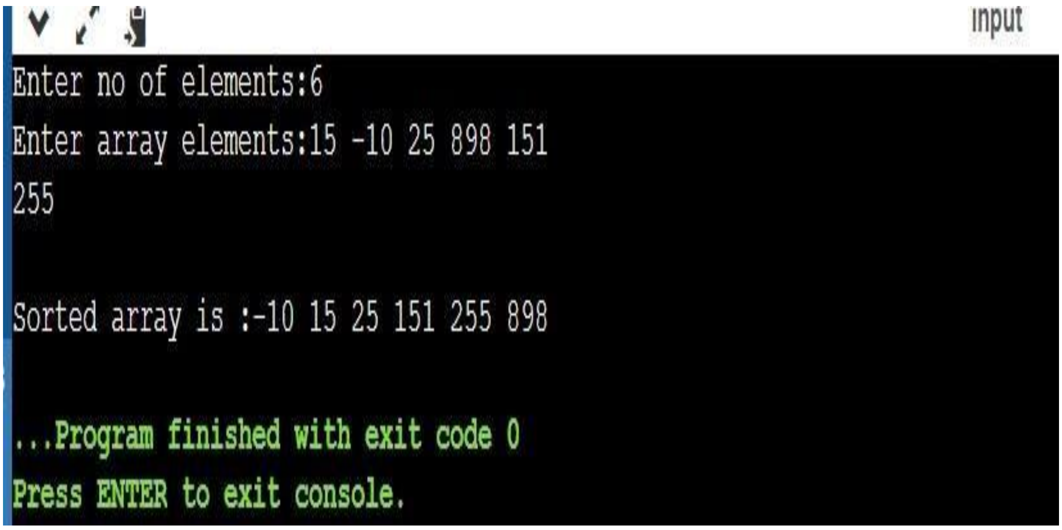
CODE:

```
#include<stdio.h> int main()
{
  int n, i;
  printf ("Enter no of
elements:");
  scanf ("%d", &n);
  int a[n];
  printf ("Enter array elements:");
  for (i = 0; i < n; i++)
    {
      scanf ("%d", &a[i]);

}
  mergesort (a, 0, n - 1);
  printf ("\nSorted array is :");
  for (i = 0; i < n; i++)
    {
      printf ("%d ", a[i]);

}
  return
0;
```

```
}
void
mergesort (int a[], int lb, int ub)
{
  int mid;
  if (lb < ub)
    {
      mid = (lb + ub) / 2;
      mergesort (a, lb, mid);
      mergesort (a, mid + 1, ub);
      merge (a, lb, mid, mid + 1, ub);
    }
}
void
merge (int a[], int lb, int mid, int ub)
{
  int b[ub + 1];
  int i, j, k;
  i = lb;
  j = mid + 1;
  k = 0;
  while (i <= mid && j <= ub)
    {

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

b[k++] = a[i++];
}
      else
{
b[k++] = a[j++];
```

```
}
}
  while (i <= mid)
    {
      b[k++] = a[i++];
    }
  while (j <= ub)
    {
      b[k++] = a[j++];
    }

for (i = lb, j = 0; i <= ub; i++, j++)
    {
      a[i] = b[j];
    }

}
```

OUTPUT:



TIME COMPLEXITY:

- BEST CASE: O(n logn)
- AVERAGE CASE: O(n logn)
- WORST CASE: O(n logn)

PRACTICAL -4

AIM: Implementation and Time analysis of Quick Sort algorithms for Best case, Average case &Worst- case using Divide and Conquer.

Algorithm:

quicksort (A, low, high) begin
Declare array A[N]
to be sorted
low = 1 st element;
high = last element;
pivot if (low < high)
begin pivot = partition (A, low, high);
quicksort (A, low, pivot - 1) quicksort (A, pivot + 1, high)
End end

CODE:

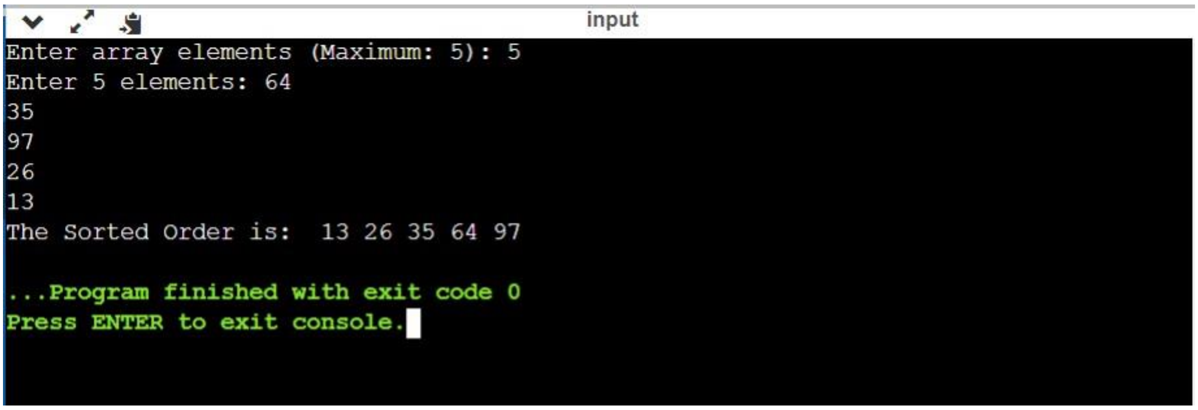
```
#include<stdio.h>

void

quicksort (int number[5], int first, int last)

{

  int i, j, pivot, temp;

  if (first < last)

    {

      pivot = first;

      i = first;

      j = last;

      while (i < j)
```

```
    {
      while (number[i] <= number[pivot] && i < last)
        i++;
      while (number[j] > number[pivot])
        j--;
if(i < j)
        {
          temp = number[i];
          number[i] = number[j];
          number[j] = temp;
        }
    }
   temp = number[pivot];
   number[pivot] = number[j];
   number[j] = temp;
   quicksort (number, first, j - 1);
   quicksort (number, j + 1, last);
  }
}
int
main ()
{
  int i, count, number[5];
```

```
  printf ("Enter some elements (Max: 5): ");
  scanf ("%d", &count);
  printf ("Enter %d elements: ", count);
  for (i = 0; i < count; i++)
    scanf ("%d", &number[i]);
  quicksort (number, 0, count - 1);
  printf ("The Sorted Order is: ");
  for (i = 0; i < count; i++)
    printf (" %d", number[i]);

  return 0;
}
```

Output:

Worst case: Time Complexity = O(n^2)



```
input
Enter array elements (Maximum: 5): 5
Enter 5 elements: 64
35
97
26
13
The Sorted Order is:  13 26 35 64 97

...Program finished with exit code 0
Press ENTER to exit console.
```

Average case: Time Complexity = O(nlog(n))

```
Enter some elements (Max: 5): 5
Enter 5 elements: 26
35
13
97
64
The Sorted Order is:  13 26 35 64 97

...Program finished with exit code 0
Press ENTER to exit console.
```

Best case: Time Complexity = O(nlog(n))

```
Enter some elements (Max: 5): 5
Enter 5 elements: 26
13
97
35
64
The Sorted Order is:  13 26 35 64 97

...Program finished with exit code 0
Press ENTER to exit console.
```

Space Complexity = O(nlog(n))

PRACTICAL -5

AIM: Write a program to solve fractional knapsack problem.

Algorithm:-

int
knapSack (int W, int w[], int v[], int n)
int i, wt;
int K[n + 1][W + 1] for i = 0
to n for wt
= 0 to W if (i == 0 or wt == 0)
Do K[i][wt] = 0
else
if (w[i - 1] <= wt)

Compute:K[i][wt] = max (v[i - 1] + K[i - 1][wt - w[i - 1]], K[i - 1][wt])
else
K[i][wt] = K[i - 1][wt] return K[n][W] End

Program:

```
#include <stdio.h>

int n = 5;

  int c[10] = { 12, 1, 2, 1, 4 };

  int v[10] = { 4, 2, 2, 1, 10 };

  int W = 15;

  void

  simple_fill ()

  { int cur_w;

    float tot_v;
```

```
int i, maxi;

int used[10];

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

  used[i] = 0;

cur_w =  W;

while (cur_w > 0)

  {

    maxi = -1;

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

       if ((used[i] == 0) &&

           ((maxi == -1)

            || ((float) v[i] / c[i] > (float) v[maxi] / c[maxi])))

         maxi = i;

    used[maxi] = 1;

    cur_w -= c[maxi];

    tot_v += v[maxi];

    if (cur_w >= 0)

       printf

         ("Added object %d (%d$, %dKg) completely in the bag. Space left: %d.\n",

          maxi + 1, v[maxi], c[maxi], cur_w);

    else

       {

         printf ("Added %d%% (%d$, %dKg) of object %d in the bag.\n",
```

```
                    (int) ((1 + (float) cur_w / c[maxi]) * 100), v[maxi],
                    c[maxi], maxi + 1);
            tot_v -= v[maxi];
            tot_v += (1 + (float) cur_w / c[maxi]) * v[maxi];
          }
    }
  printf ("Filled the bag with objects worth %.2f$.\n", tot_v);
}
int
main (int argc, char *argv[])
{
  simple_fill ();
  return 0;
}
```

OUTPUT:

Time complexity of the fractional knapsack problem: O(NlogN)

Space Complexity: O(n)

```
Added object 5 (10$, 4Kg) completely in the bag. Space left: 11.
Added object 2 (2$, 1Kg) completely in the bag. Space left: 10.
Added object 3 (2$, 2Kg) completely in the bag. Space left: 8.
Added object 4 (1$, 1Kg) completely in the bag. Space left: 7.
Added 58% (4$, 12Kg) of object 1 in the bag.
Filled the bag with objects worth 17.33$.


...Program finished with exit code 0
Press ENTER to exit console.
```

Practical-6

Aim:- Implementation and Time analysis of Krushkal's Minimum spanning Tree algorithms.

Algorithm for Krushkal:-
MST-KRUSHKAL(G,W)
A=FI
For each vertex v in V[G]
do Make-Set(v)
sort the edges of E in no decreasing
order by weight W
do if Find-Set(n) not equal to Find-Set(V)
then A = AU{(u,v)}
Union(W,V)
Return A

Program:-

```
 #include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int i, j, k, a, b, u, v, n, ne = 1;

int min, mincost = 0, cost[9][9], parent[9];

int find (int);

int uni (int, int);

void
main ()
{
  printf ("\n\tImplementation of Kruskal's algorithm\n");

  printf ("\nEnter the no. of vertices:");

  scanf ("%d", &n);

  printf ("\nEnter the cost adjacency matrix:\n");

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

    {

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

```
        {
          scanf ("%d", &cost[i][j]);
          if (cost[i][j] == 0)
            cost[i][j] = 999;
        }
  }
printf ("The edges of Minimum Cost Spanning Tree are\n");
while (ne < n)
  {
    for (i = 1, min = 999; i <= n; i++)
        {
          for (j = 1; j <= n; j++)
            {
              if (cost[i][j] < min)
                  {
                    min = cost[i][j];
                    a = u = i;
                    b = v = j;
                  }
            }
        }
    u = find (u);
    v = find (v);
```

```
    if (uni (u, v))
        {
          printf ("%d edge (%d,%d) =%d\n", ne++, a, b, min);
          mincost += min;
        }
    cost[a][b] = cost[b][a] = 999;
  }
 printf ("\n\tMinimum cost = %d\n", mincost);
 getch ();
}

int
find (int i)
{
 while (parent[i])
  i = parent[i];
 return i;
}

int
uni (int i, int j)
{
 if (i != j)
  {
    parent[j] = i;
    return 1;
```

```
    }

  return 0;

}
```

Time Complexity:-
Best Case → f(n) = O(NlogE)
Average Case → f(n) =
Worst Case → f(n) = O(ElogE)

```
        Implementation of Kruskal's algorithm

Enter the no. of vertices:4

Enter the cost adjacency matrix:
10
1
2
3
1
10
4
5
2
4
10
7
3
5
7
10
The edges of Minimum Cost Spanning Tree are
1 edge (1,2) =1
2 edge (1,3) =2
3 edge (1,4) =3

        Minimum cost = 6


...Program finished with exit code 255
Press ENTER to exit console.
```

Practical-7

Aim:- Implementation and Time analysis of Prim's Minimum spanning Tree algorithms

Algorithm:-

```
Prim(g,w,r)
 For each u in V[g]
    do key [u] = infinite
        PI[r] = NIL
        key[r] = 0
       Q = V[g]
      While Q is not equal to FI
        do u= EXTRACT-MIN[g]
         for each V in adj[u]
         do if V in Q & w(u,v) Key[v]
            then PI[v]=u
       Key[V] = w(u,v)
```

Program:-

```
#include<stdio.h>

#include<conio.h>

int a, b, u, v, n, i, j, ne = 1;

int visited[10] = { 0 }, min, mincost = 0, cost[10][10];

void
main ()
{

  printf ("\nEnter the number of nodes:");

  scanf ("%d", &n);

  printf ("\nEnter the adjacency matrix:\n");

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

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

      {

          scanf ("%d", &cost[i][j]);
```

```
        if (cost[i][j] == 0)
          cost[i][j] = 999;
    }
visited[1] = 1;
printf ("\n");
while (ne < n)
 {
   for (i = 1, min = 999; i <= n; i++)
        for (j = 1; j <= n; j++)
          if (cost[i][j] < min)
            if (visited[i] != 0)
              {
                     min = cost[i][j];
                     a = u = i;
                     b = v = j;
              }
   if (visited[u] == 0 || visited[v] == 0)
        {
          printf ("\n Edge %d:(%d %d) cost:%d", ne++, a, b, min);
          mincost += min;
          visited[b] = 1;
        }
   cost[a][b] = cost[b][a] = 999;
```

```
  }

 printf ("\n Minimun cost=%d", mincost);

 getch ();

}
```

Time Complexity:-
Best Case → f(n) =
Average Case → f(n) =
Worst Case → f(n) =

```
Enter the number of nodes:3

Enter the adjacency matrix:
9
2
1
2
9
5
1
5
9


 Edge 1:(1 3) cost:1
 Edge 2:(1 2) cost:2
 Minimun cost=3

...Program finished with exit code 255
Press ENTER to exit console.
```