CERTIFICATE

*This is to certify that Mr./Ms. ……. **Hemil…Chovatiya............*** *with enrolment no.* ***..........200303108003.................*** *has successfully completed* ***his***/*her laboratory experiments in the* ***....Design and Analysis of Algorithms....*** *from the department of* ***........Information Technology(5ITA1)….….........*** *during the academic year* ***........2022-2023.........***

Date of Submission: ....

Staff In charge: ....

Head of Department: ....

PRACTICAL-1

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

1) Bubble Sorting:-

Algorithm:

1. begin BubbleSort(arr)
2. for all array elements
3. if arr[i] > arr[i+1]
4. swap(arr[i], arr[i+1])
5. end if
6. end for
7. return arr
8. end BubbleSort

Code:-

```
#include<stdio.h>
#include<conio.h>
int main(){
int n, temp, i, j, number[30];
printf("Enter number of elemnts:");
scanf("%d",&n);
printf("Enter %d numbers: ",n);
for(i=0;i<n;i++)
scanf("%d",&number[i]);
for(i=n-2;i>=0;i--){
for(j=0;j<=i;j++){
if(number[j]>number[j+1]){
temp=number[j];
number[j]=number[j+1];
number[j+1]=temp;
}} }
```

```
printf("Sorted elements: ");
for(i=0;i<n;i++)
printf(" %d",number[i]);
return 0;
}
```

OUTPUT:Best Case:O(n)

```
PS F:\pu_it_pratical\DAA\New folder> cd "f:\pu_it_pratical\DAA\New fol
Enter number of elemnts:4
Enter 4 numbers: 12
13
14
15
Sorted elements:  12 13 14 15
PS F:\pu_it_pratical\DAA\New folder>
```

Avg Case:O(n^2)

```
PS F:\pu_it_pratical\DAA\New folder> cd "f:\pu_it_pratical\
Enter number of elemnts:5
Enter 5 numbers: 11
33
44
77
88
Sorted elements:  11 33 44 77 88
PS F:\pu_it_pratical\DAA\New folder>
```

Wrost Case:O(n^2)

```
PS F:\pu_it_pratical\DAA\New folder> cd "f:\pu_it_pratical\DAA
Enter number of elemnts:5
Enter 5 numbers: 99
77
66
44
33
Sorted elements:  33 44 66 77 99
PS F:\pu_it_pratical\DAA\New folder>
```

2) Selection Sorting:-

Algorithm:

1. SELECTION SORT(arr, n)
2. Step 1: Repeat Steps 2 and 3 for i = 0 to n-1
3. Step 2: CALL SMALLEST(arr, i, n, pos)

4. Step 3: SWAP arr[i] with arr[pos]
5. [END OF LOOP]
6. Step 4: EXIT
7. SMALLEST (arr, i, n, pos)
8. Step 1: [INITIALIZE] SET SMALL = arr[i]
9. Step 2: [INITIALIZE] SET pos = i
10. Step 3: Repeat for j = i+1 to n
11. if (SMALL > arr[j])
12. SET SMALL = arr[j]
13. SET pos = j
14. [END OF if]
15. [END OF LOOP]
16. Step 4: RETURN pos

Code:-

```
#include<stdio.h>
#include<conio.h>
int main()
{
int a[100], n, i, j, position, swap;
printf("enter the number of inputs:");
scanf("%d", &n);
printf("Enter %d Numbers:", n);
for (i = 0; i < n; i++)
scanf("%d", &a[i]);
for(i = 0; i < n - 1; i++)
{
position=i;
for(j = i + 1; j < n; j++)
{
if(a[position] > a[j])
```

```
position=j;
}
if(position != i)
{
swap=a[i];
a[i]=a[position];
a[position]=swap;
}
}
printf("Sorted Array:");
for(i = 0; i < n; i++)
printf("%d\n", a[i]);
return 0;
}
```

OUTPUT: **Best Case:(O(n^2))**

```
enter the number of inputs: 5
Enter 5 elements: 1
2
4
5
8
your Sorted elements are:  1 2 4 5 8
PS F:\pu_it_pratical\DAA\New folder>
```

Avg Case:O(n^2)

```
PS F:\pu_it_pratical\DAA\New folder> cd "f:\pu_it_pratical\DAA\New f
enter the number of inputs: 5
Enter 5 elements: 11
12
49
18
12
your Sorted elements are:  11 12 12 18 49
PS F:\pu_it_pratical\DAA\New folder>
```

Wrost Case:O(n^2)

```
PROBLEMS   OUTPUT   TERMINAL   JUPYTER   DEBUG CONSOLE

PS F:\pu_it_pratical\DAA\New folder> cd "f:\pu_it_prati
enter the number of inputs: 5
Enter 5 elements: 99
88
44
33
22
your Sorted elements are:  22 33 44 88 99
PS F:\pu_it_pratical\DAA\New folder>
```

3) Insertion Sorting:-

Algorithm:

Step 1 - If the element is the first element, assume that it is already sorted. Return 1.
Step2 - Pick the next element, and store it separately in a key.
Step3 - Now, compare the key with all elements in the sorted array.
Step 4 - If the element in the sorted array is smaller than the current element, then move to the next element. Else, shift greater elements in the array towards the right.
Step 5 - Insert the value.
Step 6 - Repeat until the array is sorted.

Code:-

```
#include <math.h>
#include <stdio.h>
int main(){
int i, j, count, temp, number[25];
  printf("numbers of input: ");
  scanf("%d",&count);
  printf("Enter %d the numbers: ", count);
  for(i=0;i<count;i++)
    scanf("%d",&number[i]);

  for(i=1;i<count;i++)
  {
    temp=number[i];
    j=i-1;
    while((temp<number[j])&&(j>=0))
```

```
    {
      number[j+1]=number[j];
      j=j-1;
    }
    number[j+1]=temp;
  }
  printf("your insertion sorting is: ");
  for(i=0;i<count;i++)
    printf(" %d",number[i]);
  return 0;
}
```

OUTPUT:Best Case:O(n)

```
o insertion } ; if ($?) { .\insertion }
numbers of input: 4
Enter 4 the numbers: 1
2
3
4
your insertion sorting is:  1 2 3 4
PS F:\pu_it_pratical\DAA\New folder>
```

Avg Case:O(n^2)

```
{ gcc insertion.c -o insertion } ; if ($?) { .\insertion }
numbers of input: 5
Enter 5 the numbers: 9
1
4
2
10
your insertion sorting is:  1 2 4 9 10
```

Wrost Case:O(n^2)

```
numbers of input: 5
Enter 5 the numbers: 99
88
77
66
11
your insertion sorting is:  11 66 77 88 99
```

Practical 2:

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

Algorithm:

1. HeapSort(arr)
2. BuildMaxHeap(arr)
3. for i = length(arr) to 2
4. swap arr[1] with arr[i]
5. heap_size[arr] = heap_size[arr] ? 1
6. MaxHeapify(arr,1)
7. End

1. BuildMaxHeap(arr)
2. heap_size(arr) = length(arr)
3. for i = length(arr)/2 to 1
4. MaxHeapify(arr,i)
5. End

1. MaxHeapify(arr,i)
2. L = left(i)
3. R = right(i)
4. if L ? heap_size[arr] and arr[L] > arr[i]
5. largest = L
6. else
7. largest = i
8. if R ? heap_size[arr] and arr[R] > arr[largest]
9. largest = R
10. if largest != i
11. swap arr[i] with arr[largest]
12. MaxHeapify(arr,largest)
13. End

Code:

```
#include <stdio.h>
void swap(int *a, int *b)
{    int temp = *a;
```

```
*a = *b;
*b = temp;  }
void heapify(int arr[], int n, int i)
{    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;
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);
} }
int main()
{
int n,i;
printf("Enter Array size: ");
scanf("%d",&n);
int arr[n];
for(i=0;i<n;i++)
{       printf("Enter Element %d: ",i+1);
```

```
scanf("%d",&arr[i]);     }
heapSort(arr, n);
printf("\nSorted Heap array:\n");
i=0;
for(i=0;i<n;i++)
{
printf("%d\t",arr[i]);
}  }
```

<u>OUTPUT:</u> Best Case: nlog(n)

```
PS F:\pu_it_pratical\DAA\New folder> c
Enter Array size: 5
Enter Element 1: 1
Enter Element 1: 1
Enter Element 2: 2
Enter Element 3: 3
Enter Element 4: 4
Enter Element 5: 5

Sorted Heap array:
1       2       3       4       5
PS F:\pu_it_pratical\DAA>
```

Avg Case: nlog(n)

```
PS C:\Users\raj> cd "f:\pu_it_pratical\DAA\" ; if ($?) { gcc hea
Enter Array size: 5
Enter Element 1: 12
Enter Element 2: 1
Enter Element 3: 9
Enter Element 4: 4
Enter Element 5: 7

Sorted Heap array:
1       4       7       9       12
PS F:\pu_it_pratical\DAA>
```

Wrost case: nlog(n)

```
PS F:\pu_it_pratical\DAA> cd "f:\pu_it_pratical\DAA\" ; if ($
Enter Array size: 5
Enter Element 1: 99
Enter Element 2: 77
Enter Element 3: 66
Enter Element 4: 55
Enter Element 5: 44

Sorted Heap array:
44      55      66      77      99
PS F:\pu_it_pratical\DAA>
```

PRACTICAL 3:

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

Algorithm:

1. MERGE_SORT(arr, beg, end)
2. if beg < end
3. set mid = (beg + end)/2
4. MERGE_SORT(arr, beg, mid)
5. MERGE_SORT(arr, mid + 1, end)
6. MERGE (arr, beg, mid, end)
7. end of if
8. END MERGE_SORT

Code:

```
#include <stdio.h>
#include <stdlib.h>
void merge(int arr[], int l, int m, int r)
{
        int i, j, k;
        int n1 = m - l + 1;
        int n2 = r - m;
        int L[n1], R[n2];
        for (i = 0; i < n1; i++)
                L[i] = arr[l + i];
        for (j = 0; j < n2; j++)
                R[j] = arr[m + 1 + j];
        i = 0;
        j = 0;
        k = l;
        while (i < n1 && j < n2) {
                if (L[i] <= R[j])
                {       arr[k] = L[i];
                        i++;            }
                else {  arr[k] = R[j];
                        j++;            }
                k++;    }
        while (i < n1) {
                arr[k] = L[i];
                i++;
```

```
        k++;    }
while (j < n2)
{       arr[k] = R[j];
        j++;
        k++;    }
}
void mergeSort(int arr[], int l, int r)
{
        if (l < r)
        {
                int m = l + (r - l) / 2;
                mergeSort(arr, l, m);
                mergeSort(arr, m + 1, r);
                merge(arr, l, m, r);
        }   }
int main()
{       int u,i=0;
        printf("Enter Element size: ");
        scanf("%d",&u);
        int A[u];
        for(i=0;i<u;i++)
         {       printf("Enter Element: ");
                 scanf("%d",&A[i]);             }
        mergeSort(A, 0, u-1);
        printf("\nSorted array is \n");
  i=0;
         for(i=0;i<u;i++)
        {      printf("%d\t",A[i]);        }
  return 0;
}
```

Output:

Best Case Complexity: O(n*logn)

```
PS C:\Users\raj> cd "f:\pu_it_pratical\DAA\" ; if ($?) { gc
Enter Element size: 4
Enter Element: 1
Enter Element: 2
Enter Element: 3
Enter Element: 4

Sorted array is
1       2       3       4
PS F:\pu_it_pratical\DAA>
```

Average Case Complexity: O(n*logn):

```
PS C:\Users\raj> cd "f:\pu_it_pratical
Enter Element size: 4
Enter Element: 1
Enter Element: 2
Enter Element: 4
Enter Element: 3

Sorted array is
1       2       3       4
PS F:\pu_it_pratical\DAA>
```

Worst Case Complexity: O(n*logn):

```
PS F:\pu_it_pratical\DAA> cd "f:\pu_it_pr
Enter Element size: 5
Enter Element: 9
Enter Element: 6
Enter Element: 5
Enter Element: 4
Enter Element: 2

Sorted array is
2       4       5       6       9
PS F:\pu_it_pratical\DAA>
```

PRACTICAL 4:

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

Algorithm:

Pivot value is Front in quick sort

```
QUICKSORT (array A, start, end)
{
 if (start < end)
 {
 p = partition(A, start, end)
 QUICKSORT (A, start, p - 1)
 QUICKSORT (A, p + 1, end)
 }
}
PARTITION (array A, start, end)
{
 pivot ? A[end]
 i ? start-1
 for j ? start to end -1 {
 do if (A[j] < pivot) {
 then i = i + 1
 swap A[i] with A[j]
  }}
 swap A[i+1] with A[end]
 return i+1
}
```

Code(Pivot Value from Start):

```
#include<stdio.h>
void quicksort(int num[],int front,int l)
{  int i,j,pivot,temp;
   if(front<l)
   {  pivot=front;
      i=front;
      j=l;
      while(i<j)
      {
         while(num[i]<=num[pivot]&&i<l)
         i++;
         while(num[j]>num[pivot])
         j--;
         if(i<j)
```

```
            {  temp=num[i];
               num[i]=num[j];
               num[j]=temp;
            }        }
        temp=num[pivot];
        num[pivot]=num[j];
        num[j]=temp;
        quicksort(num,front,j-1);
        quicksort(num,j+1,l);
    }
}
int main()
{
    int i,count;
    printf("Enter element Size:");
    scanf("%d",&count);
    int num[count];
    printf("Enter %d elements:",count);
    for (i=0;i<count;i++)
    { scanf("%d",&num[i]); }
    quicksort(num,0,count-1);
    printf("Sorted elements:");
    for(i=0;i<count;i++)
    { printf("%d\t",num[i]); }
    return 0;

}
```

Code(Pivot Value from end):

```
#include <stdio.h>
void swap(int *a, int *b)
{    int temp = *a;
    *a = *b;
    *b = temp;    }
int partition(int a[], int start, int end)
{   int pivot = a[end];
    int i = (start - 1);
    for (int j = start; j <= end - 1; j++)
    {      if (a[j] < pivot)
           {              i++;
                          int t = a[i];
                          a[i] = a[j];
                          a[j] = t;          }      }
    int t = a[i+1];
```

```
    a[i+1] = a[end];
    a[end] = t;
    return (i + 1);
}
void quicksort(int a[], int start, int end)
{      if (start < end)
       {              int p = partition(a, start, end);
                      quicksort(a, start, p - 1);
                      quicksort(a, p + 1, end);           }     }
int main()
{   int u,i=0;
    printf("Enter Element size: ");
    scanf("%d",&u);
    int A[u];
    for(i=0;i<u;i++)
    {       printf("Enter Element: ");
            scanf("%d",&A[i]);               }
     quicksort(A, 0, u - 1);
    printf("\nSorted array:\n");
    i=0;
     for(i=0;i<u;i++)
    {       printf("%d\t",A[i]);       }
     return 0;        }
```

Output:
Front:
Best Case Complexity: O(n*logn)

```
PS F:\pu_it_pratical\DAA> cd "f:\pu_it_pratical\DAA\" ; if ($?) { gcc quick_sort_fr
Enter element Size:9
Enter 9 elements:13
19
17
15
12
16
18
4
11
Sorted elements:4       11      12      13      15      16      17      18      19
PS F:\pu_it_pratical\DAA>
```

Average Case Complexity: O(n*logn)

```
PS F:\pu_it_pratical\DAA> cd "f:\pu_it_pratical\DAA\" ; if
Enter element Size:4
Enter 4 elements:1
3
2
5
Sorted elements:1       2       3       5
```

Worst Case Complexity: O(n^2):

```
Enter Element size: 4
Enter Element: 1
Enter Element: 2
Enter Element: 5
Enter Element: 4

Sorted array:
1       2       4       5
PS F:\pu_it_pratical\DAA>
```

End:

Best Case Complexity: O(n*logn)

```
Enter Element size: 9
Enter Element: 19
Enter Element: 17
Enter Element: 15
Enter Element: 12
Enter Element: 16
Enter Element: 18
Enter Element: 4
Enter Element: 11
Enter Element: 13

Sorted array:
4       11      12      13      15      16      17      18      19
```

Average Case Complexity: O(n*logn)

```
Enter Element size: 4
Enter Element: 1
Enter Element: 2
Enter Element: 5
Enter Element: 4

Sorted array:
1       2       4       5
```

Worst Case Complexity: O(n^2):

```
Enter Element size: 5
Enter Element: 9
Enter Element: 7
Enter Element: 6
Enter Element: 5
Enter Element: 1

Sorted array:
1       5       6       7       9
```

PRACTICAL 5:

AIM: Write a program to solve fractional knapsack problem.

Algorithm:

Algorithm: Greedy-Fractional-Knapsack (w[1..n], p[1..n], W)

for i = 1 to n
do x[i] = 0
weight = 0
for i = 1 to n
if weight + w[i] ≤ W then
x[i] = 1
weight = weight + w[i]
else
x[i] = (W - weight) / w[i]
weight = W
break
return x

Code:

```
#include <stdio.h>
int n = 5; /* The number of objects */
int c[10] = {12, 1, 2, 1, 4}; /* c[i] is the *COST* of the ith object; i.e. what
            YOU PAY to take the object */
int v[10] = {4, 2, 2, 1, 10}; /* v[i] is the *VALUE* of the ith object; i.e.
            what YOU GET for taking the object */
int W = 15; /* The maximum weight you can take */
void simple_fill() {
   int cur_w;
   float tot_v;
   int i, maxi;
   int used[10];

   for (i = 0; i < n; ++i)
      used[i] = 0; /* I have not used the ith object yet */

   cur_w = W;
   while (cur_w > 0) { /* while there's still room*/
      /* Find the best object */
      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; /* mark the maxi-th object as used */
        cur_w -= c[maxi]; /* with the object in the bag, I can carry less */
        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:

```
PS F:\pu_it_pratical\DAA> cd "f:\pu_it_pratical\DAA\" ; if ($?) { gcc fractional_knapsa
Enter the capacity of knapsack:
50
Enter the number of items:
3
Enter the weight and value of 3 item:
Weight[0]:      10
Value[0]:       60
Weight[1]:      20
Value[1]:       100
Weight[2]:      30
Value[2]:       120
Added object 1 (60 Rs., 10Kg) completely in the bag. Space left: 40.
Added object 2 (100 Rs., 20Kg) completely in the bag. Space left: 20.
Added 66% (120 Rs., 30Kg) of object 3 in the bag.
Filled the bag with objects worth 240.00 Rs.
PS F:\pu_it_pratical\DAA>
```

PRACTICAL 6:

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

Algorithm:

Steps for finding MST using Kruskal's Algorithm:

1. Arrange the edge of G in order of increasing weight.
2. Starting only with the vertices of G and proceeding sequentially add each edge which does not result in a cycle, until (n - 1) edges are used.
3. EXIT.

MST- KRUSKAL (G, w)

1. A ← ∅
2. for each vertex v ∈ V [G]
3. do MAKE - SET (v)
4. sort the edges of E into non decreasing order by weight w
5. for each edge (u, v) ∈ E, taken in non decreasing order by weight
6. do if FIND-SET (μ) ≠ if FIND-SET (v)
7. then A ← A ∪ {(u, v)}
8. UNION (u, v)
9. return A

Time Analysis:

Analysis: Where E is the number of edges in the graph and V is the number of vertices, Kruskal's Algorithm can be shown to run in O (E log E) time, or simply, O (E log V) time, all with simple data structures. These running times are equivalent because:

- **E** is at most **V^2** and **log V^2= 2 x log V is O (log V).**
- If we ignore isolated vertices, which will each their components of the minimum spanning tree, V ≤ 2 E, so log V is O (log E).

Thus the total time is

1. **O (E log E) = O (E log V).**

Code:

```
#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;
}
```

Output:

```
PS F:\pu_it_pratical\DAA> cd "f:\pu_it_pratical\DAA\" ; if ($?) { gcc kruskal_MST.c -o

        Implementation of Kruskal's Algorithm

Enter the no. of vertices:6

Enter the cost adjacency matrix:
0 3 1 6 0 0
3 0 5 0 3 0
1 5 0 5 6 4
6 0 5 0 0 2
0 3 6 0 0 6
0 0 4 2 6 0
The edges of Minimum Cost Spanning Tree are
1 edge (1,3) =1
2 edge (4,6) =2
3 edge (1,2) =3
4 edge (2,5) =3
5 edge (3,6) =4

        Minimum cost = 13
```

PRACTICAL 7:

AIM: Implementation and Time analysis of Prim's Minimum spanning Tree algorithms.

Algorithm:

```
Step 1 −     for (int v = 0; v < V; v++)
        if (mstSet[v] == false && key[v] < min)
            min = key[v], min_index = v;
Step 2 − for (int i = 0; i < V; i++)
        key[i] = INT_MAX, mstSet[i] = false;
Step 3 −    for (int count = 0; count < V - 1; count++) {
        int u = minKey(key, mstSet);
Step 4 −        for (int v = 0; v < V; v++)
            if (graph[u][v] && mstSet[v] == false
```

Code:

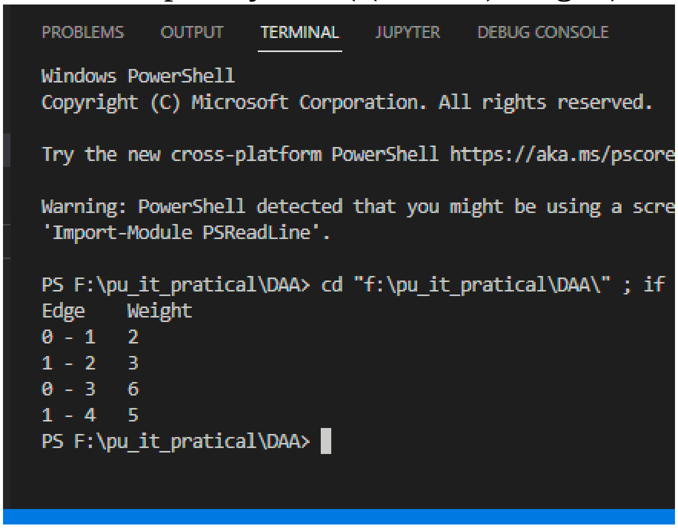
```
#include <limits.h>
#include <stdbool.h>
#include <stdio.h>
#define V 5
int minKey(int key[], bool mstSet[])
{
  int min = INT_MAX, min_index;
    for (int v = 0; v < V; v++)
        if (mstSet[v] == false && key[v] < min)
            min = key[v], min_index = v;

    return min_index;
}
int printMST(int parent[], int graph[V][V])
{
    printf("Edge \tWeight\n");
    for (int i = 1; i < V; i++)
        printf("%d - %d \t%d \n", parent[i], i,
            graph[i][parent[i]]);
}
void primMST(int graph[V][V])
{
    int parent[V];
    int key[V];
    bool mstSet[V];
    for (int i = 0; i < V; i++)
```

```
        key[i] = INT_MAX, mstSet[i] = false;
    key[0] = 0;
    parent[0] = -1; // First node is always root of MST
    for (int count = 0; count < V - 1; count++) {
        int u = minKey(key, mstSet);
        mstSet[u] = true;
        for (int v = 0; v < V; v++)
            if (graph[u][v] && mstSet[v] == false
                && graph[u][v] < key[v])
                parent[v] = u, key[v] = graph[u][v];
    }
    printMST(parent, graph);
}
int main()
{   int graph[V][V] = { { 0, 2, 0, 6, 0 },
                        { 2, 0, 3, 8, 5 },
                        { 0, 3, 0, 0, 7 },
                        { 6, 8, 0, 0, 9 },
                        { 0, 5, 7, 9, 0 } };
    primMST(graph);
    return 0;
}
```

Output:

Time Complexity = O ((V + E) l o g V)



PRACTICAL 8:

AIM: Write a program to solve 0-1 knapsack problem.

Algorithm:

Step 1 − if (i==0 || w==0)
K[i][w] = 0;
Step 2 − K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]], K[i-1][w]);
Step 3 − K[i][w] = K[i-1][w];
Step 4 − int main()
int i, n, val[20], wt[20], W;

Code:

```
#include<stdio.h>
int max(int a, int b) { return (a > b)? a : b; }
int knapSack(int W, int wt[], int val[], int n)
{
   int i, w;
   int K[n+1][W+1];
   for (i = 0; i <= n; i++)
   {
       for (w = 0; w <= W; w++)
       {
           if (i==0 || w==0)
               K[i][w] = 0;
           else if (wt[i-1] <= w)
                 K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]],  K[i-1][w]);
           else
                 K[i][w] = K[i-1][w];
       }
   }
   return K[n][W];
}
int main()
{
    int i, n, val[20], wt[20], W;

    printf("Enter number of items:");
    scanf("%d", &n);

    printf("Enter value and weight of items:\n");
    for(i = 0;i < n; ++i){
     scanf("%d%d", &val[i], &wt[i]);
```

```
    }
     printf("Enter size of knapsack:");
    scanf("%d", &W);
    printf("%d", knapSack(W, wt, val, n));
    return 0;
}
```

Output:

```
PS F:\pu_it_pratical\DAA> cd "f:\pu_i
Enter number of items:4
Enter value and weight of items:
13 42
51 15
42 25
16 18
Enter size of knapsack:80
109
PS F:\pu_it_pratical\DAA>
```

PRACTICAL 9:

AIM: Implementation and Time analysis of Depth First Search (DFS) Graph Traversal and Breadth First Traversal (BFS) Graph Traversal.

1.BFS:

Algorithm:

Step 1 − for(i=0;i<n;i++) visited[i]=0;
DFS(0);
Step 2 − visited[i]=1; for(j=0;j<n;j++)
Step 3 − if(!visited[j]&&G[i][j]==1) DFS(j);

Code:

```
#include<stdio.h>
void DFS(int);
int G[10][10],visited[10],n;
void main()
{
int i,j;
printf("Enter number of vertices:");
scanf("%d",&n);
printf("\nEnter adjecency matrix of the graph:");
for(i=0;i<n;i++) for(j=0;j<n;j++)
scanf("%d",&G[i][j]);
for(i=0;i<n;i++) visited[i]=0;
DFS(0);
}
void DFS(int i)
{
int j;
printf("\n%d",i);
visited[i]=1; for(j=0;j<n;j++)
if(!visited[j]&&G[i][j]==1) DFS(j);
}
```

Output:

```
PS F:\pu_it_pratical\DAA> cd "f:\pu_it_pratical\DAA\" ; if ($?)
Enter number of vertices:5

Enter adjecency matrix of the graph:1 0 1 1 0
1 0 0 0 1
0 1 1 1 0
1 0 1 0 1
0 1 1 0 1

0
2
1
4
3
PS F:\pu_it_pratical\DAA>
```

2.DFS

Algorithm:

Step 1 − for(i=0;i<n;i++)
  visited[i]=0; DFS(0);
Step 2 − void DFS(int i)
  node *p;
  visited[i]=1; while(p!=NULL)
Step 3 − i=p->vertex;
  if(!visited[i])
  DFS(i);
  p=p->next;
Step 4 − node *p,*q;
  q=(node*)malloc(sizeof(node)); q->vertex=vj;

Code:

```
#include<stdio.h>
#include<stdlib.h>
typedef struct node
{
struct node *next;
int vertex;
}node;
node *G[20];
int visited[20];
int n;
void read_graph();
void insert(int,int);
void DFS(int);
void main()
{       int i;
read_graph();
for(i=0;i<n;i++)
```

```
visited[i]=0; DFS(0);
}
void DFS(int i)
{       node *p;
        printf("\n%d",i); p=G[i];
        visited[i]=1; while(p!=NULL)
{       i=p->vertex;
        if(!visited[i])
        DFS(i);
        p=p->next;                  }}
void read_graph()
{       int i,vi,vj,no_of_edges; printf("Enter number of vertices:");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
        G[i]=NULL;
        printf("Enter number of edges:"); scanf("%d",&no_of_edges);
        for(i=0;i<no_of_edges;i++)
        {
        printf("Enter an edge(u,v):");
        scanf("%d%d",&vi,&vj); insert(vi,vj);
}}}
void insert(int vi,int vj)
{      node *p,*q;
        q=(node*)malloc(sizeof(node)); q->vertex=vj;
        q->next=NULL;
        if(G[vi]==NULL)
        G[vi]=q; else
        {p=G[vi];
        while(p->next!=NULL) p=p->next;
        p->next=q;
          }}
```

Output:

```
PS F:\pu_it_pratical\DAA> cd "f:\pu_it_pratical\DAA\
Enter number of vertices:4
Enter number of edges:6
Enter an edge(u,v):2 4
Enter an edge(u,v):1 4
Enter an edge(u,v):2 5
Enter an edge(u,v):0 5
Enter an edge(u,v):5 0
Enter an edge(u,v):4 6

0
5
PS F:\pu_it_pratical\DAA>
```