EXPERIMENT-7

Program:

WAP to implement insertion sort using c/c++ and write the complexity.

Pseudo code:

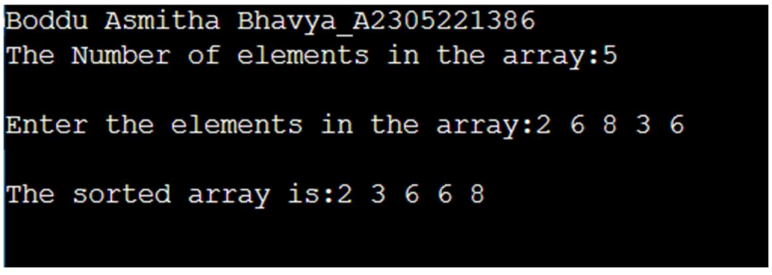
```
InsertionSort(A)
  for j ← 2 to length[A]
    do key ← A[j]
    i ← j - 1
    while i > 0 and A[i] > key
      do A[i + 1] ← A[i]
      i ← i - 1
    A[i + 1] ← key
```

Input:

```
#include <math.h>
#include <stdio.h>
#define N 100
void insertionSort(int arr[], int n)
{
        int i, key, j;
        for (i = 1; i < n; i++)
        {
                key = arr[i];
                j = i - 1;
                while (j >= 0 && arr[j] > key)
                {
                        arr[j + 1] = arr[j];
                        j = j - 1;
                }
                arr[j + 1] = key;
        }
}
```

```
void printArray(int arr[], int n)
{
        int i;
        for (i = 0; i < n; i++)
                printf("%d ", arr[i]);
        printf("\n");
}
int main()
{
        printf("Boddu Asmitha Bhavya_A2305221386");
   int a[N],i,n;
   printf("\nThe Number of elements in the array:");
   scanf("%d",&n);
   printf("\nEnter the elements in the array:");
   for(i=0;i<n;i++)
   {
      scanf("%d",&a[i]);
   }
   insertionSort(a,n);
   printf("\nThe sorted array is:");
   printArray(a,n);
   return 0;
}
```

Output:



Complexity:

Time complexity is:O(N^2)

EXPERIMENT-8

Program:

WAP to implement Fractional Knapsack problem using c/c++ and write the complexity.

Pseudo code:

GREEDY_FRACTIONAL_KNAPSACK (X, V, W, M)

1. for i <- 1 to size (V)

2. calculate cost [i] <- V [i] / W [i]

3. Sort-Descending (cost)

4. i ← 1

5. while (i <= size (V))

6. if W [i] <= M

7. M ← M – W [i]

8. total ← total + V [i];

9. if W [i] > M

10. i ← i+1

Input:

```
#include <stdio.h>
void main()
{
   int capacity, no_items, cur_weight, item;
   int used[10];
   float total_profit;
   int i;
   int weight[10];
   int value[10];
   printf("Boddu Asmitha Bhavya_A2305221386");
   printf("\nEnter the capacity of knapsack:");
   scanf("%d", &capacity);
   printf("\nEnter the number of items:");
   scanf("%d", &no_items);
   printf("\nEnter the weight and value of %d item:", no_items);
```

```
   for (i = 0; i<no_items; i++)
   {
      printf("\nWeight[%d]:", i);
      scanf("%d", &weight[i]);
      printf("\nValue[%d]:", i);
      scanf("%d", &value[i]);
   }
   for (i = 0; i < no_items; ++i)
   used[i]= 0;
   cur_weight = capacity;
   while (cur_weight > 0)
   {
      item = -1;
      for (i = 0; i < no_items; ++i)
         if ((used[i] == 0) &&
            ((item == -1) || ((float) value[i] / weight[i] > (float) value[item] / weight[item])))
            item = i;

      used[item] = 1;
      cur_weight -= weight[item];
      total_profit += value[item];
      if (cur_weight >= 0)
         printf("\nAdded object %d (%d Rs., %dKg) completely in the bag. Space left: %d.", item + 1,
value[item], weight[item], cur_weight);
      else
      {
         int item_percent = (int) ((1 + (float) cur_weight / weight[item]) * 100);
         printf("\nAdded %d%% (%d Rs., %dKg) of object %d in the bag.", item_percent, value[item],
weight[item], item + 1);
         total_profit -= value[item];
         total_profit += (1 + (float)cur_weight / weight[item]) * value[item];
      }
```

```
    }
    printf("\nFilled the bag with objects worth %.2f Rs.", total_profit);
}
```

Output:

```
Boddu Asmitha Bhavya_A2305221386
Enter the capacity of knapsack:50

Enter the number of items:3

Enter the weight and value of 3 item:
Weight[0]:20

Value[0]:30

Weight[1]:40

Value[1]:50

Weight[2]:10

Value[2]:20

Added object 3 (20 Rs., 10Kg) completely in the bag. Space left: 40.
Added object 1 (30 Rs., 20Kg) completely in the bag. Space left: 20.
Added 50% (50 Rs., 40Kg) of object 2 in the bag.
Filled the bag with objects worth 75.00 Rs.
```

Complexity:

Time Complexity is: O(N * logN)

EXPERIMENT-9

Program:

WAP to implement the Kruskal's Algorithm using c/c++.

Pseudo code:

KRUSKAL (G):

A = Ø

For each vertex v ∈ G.V:

  MAKE-SET(v)

For each edge (u, v) ∈ G.E ordered by increasing order by weight (u, v):

  if FIND-SET(u) ≠ FIND-SET(v):

    A = A ∪ { (u, v)}

    UNION(u, v)

return A

Input:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_VERTICES 100
#define MAX_EDGES 1000
int find(int subsets[], int i) {
  if (subsets[i] != i)
    subsets[i] = find(subsets, subsets[i]);
  return subsets[i];
}
void unionSets(int subsets[], int x, int y) {
  int xroot = find(subsets, x);
  int yroot = find(subsets, y);

  subsets[xroot] = yroot;
}
int compare(const void* a, const void* b) {
  return (*(int*)a - *(int*)b);
```

```
}
void kruskalMST(int V, int E, int edges[][3]) {
    qsort(edges, E, sizeof(edges[0]), compare);

    int subsets[MAX_VERTICES];
    for (int v = 0; v < V; v++) {
        subsets[v] = v;
    }

    printf("\nMinimum Spanning Tree:\n");
    int totalCost = 0;
    for (int i = 0, e = 0; e < V - 1 && i < E; i++) {
        int src = edges[i][0];
        int dest = edges[i][1];
        int x = find(subsets, src);
        int y = find(subsets, dest);

        if (x != y) {
            unionSets(subsets, x, y);
            printf("%d - %d : %d\n", src, dest, edges[i][2]);
            totalCost += edges[i][2];
            e++;
        }
    }

    printf("\nTotal cost of Minimum Spanning Tree: %d\n", totalCost);
}

int main() {
    int V, E;
    printf("Boddu Asmitha Bhavya_A2305221386");
```

```
    printf("\nEnter the number of vertices and edges: ");
    scanf("%d %d", &V, &E);

    int edges[MAX_EDGES][3];
    for (int i = 0; i < E; i++) {
        printf("\nEnter edge %d (source destination weight): ", i + 1);
        scanf("%d %d %d", &edges[i][0], &edges[i][1], &edges[i][2]);
    }

    kruskalMST(V, E, edges);

    return 0;
}
```

Output:

```
Boddu Asmitha Bhavya_A2305221386
Enter the number of vertices and edges: 6 10

Enter edge 1 (source destination weight): 1 2 5

Enter edge 2 (source destination weight): 2 6 3

Enter edge 3 (source destination weight): 6 5 4

Enter edge 4 (source destination weight): 5 3 3

Enter edge 5 (source destination weight): 3 1 4

Enter edge 6 (source destination weight): 1 5 2

Enter edge 7 (source destination weight): 4 1 6

Enter edge 8 (source destination weight): 4 2 2

Enter edge 9 (source destination weight): 4 6 2

Enter edge 10 (source destination weight): 4 5 1

Minimum Spanning Tree:
1 - 2 : 5
1 - 5 : 2
2 - 6 : 3
3 - 1 : 4
4 - 1 : 6

Total cost of Minimum Spanning Tree: 20
```

Complexity:

Time Complexity is O(E * logE) or O(E * logV)

EXPERIMENT-10

Program:

WAP to implement the Prism's algorithm using c/c++ and write the complexity.

Pseudo code:

Prim(G, w, s)

1. T = ∅; U = { s };

2. while (U ≠ V)

3. let (u, v) be the lowest cost edge such that u ∈ U and v ∈ V - U;

4. T = T ∪ { (u, v) };

5. U = U ∪ { v };

6. return T;

Input:

```
#include <stdio.h>
#include <stdbool.h>
#define MAX_VERTICES 10
int graph[MAX_VERTICES][MAX_VERTICES];
int parent[MAX_VERTICES];
int key[MAX_VERTICES];
bool mstSet[MAX_VERTICES];
int findMinKeyVertex(int vertices) {
   int minKey = __INT_MAX__;
   int minIndex = -1;
   for (int v = 0; v < vertices; v++) {
      if (!mstSet[v] && key[v] < minKey) {
         minKey = key[v];
         minIndex = v;
      }
   }
   return minIndex;
}
```

```
void printMST(int vertices) {
    printf("\nEdge \tWeight\n");
    for (int v = 1; v < vertices; v++) {
        printf("%d - %d\t%d\n", parent[v], v, graph[parent[v]][v]);
    }
}
void primMST(int vertices) {
    for (int v = 0; v < vertices; v++) {
        key[v] = __INT_MAX__;
        mstSet[v] = false;
    }
  key[0] = 0;
    parent[0] = -1;
    for (int count = 0; count < vertices - 1; count++) {
        int u = findMinKeyVertex(vertices);
        mstSet[u] = true;
        for (int v = 0; v < vertices; v++) {
            if (graph[u][v] && !mstSet[v] && graph[u][v] < key[v]) {
                parent[v] = u;
                key[v] = graph[u][v];
            }
        }
    }
    printMST(vertices);
}
int main() {
    int vertices;
    printf("Boddu Asmitha Bhavya_A2305221386");
    printf("\nEnter the number of vertices: ");
    scanf("%d", &vertices);
```

```
    printf("\nEnter the adjacency matrix:\n");
    for (int i = 0; i < vertices; i++) {
        for (int j = 0; j < vertices; j++) {
            scanf("%d", &graph[i][j]);
        }
    }
  primMST(vertices);


    return 0;
}
```

Output:

```
Boddu Asmitha Bhavya_A2305221386
Enter the number of vertices: 4

Enter the adjacency matrix:
0 2 5 0
2 0 5 3
0 8 2 6
4 8 0 1

Edge    Weight
0 - 1   2
0 - 2   5
1 - 3   3
```

Complexity:

Time Complexity is O(E*log(E)) where E is the number of edges