PRACHI AGGARWAL

BSC(H) COMPUTER SCIENCE, 2ND YEAR,

4TH SEMESTER

UNIVERSITY ROLLNO-> 21033570042

COLLEGE ROLLNO-> 21570015

UPC -> 32341401

SUBMITTED TO-> DR. ANSHULA MAM

PRACTICAL FILE

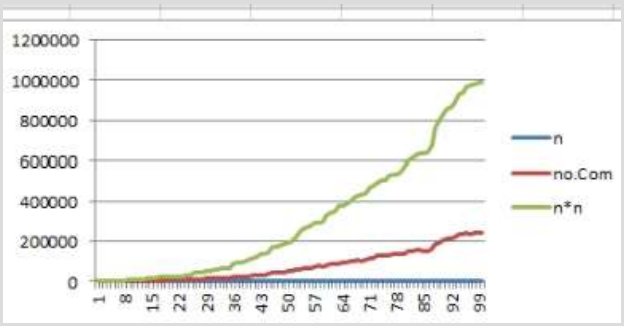
For the algorithms at S.No 1 to 3 test run the algorithm on 100 different inputs of sizes varying from 30 to 1000. Count the number of comparisons and draw the graph. Compare it with a graph of nlogn.

Q1) i. Implement Insertion Sort (The program should report the number of comparisons).

```
#include <iostream>
using namespace std;
int insertion(int arr[], int n)
{
    int comp = 0;
    int i, j, key;
    for (i = 1; i < n; i++)
    {
        j = i - 1;
        key = arr[i];
        comp++;
        while (j >= 0 && key <= arr[j])
        {
            arr[j + 1] = arr[j];
            j = j - 1;
            comp++;
        }
        arr[j + 1] = key;
    }
    return comp;
}
void print(int arr[], int n)
{
    cout << "Sorted array is: " << endl;
    for (int i = 0; i < n; i++)
    {
        cout << arr[i] << " ";
    }
    cout << endl;
}
int main()
{
    int *arr;
    int size;
    cout << "Enter the size of the array" << endl;
    cin >> size;
    cout << "Enter the unsorted array" << endl;
    arr = new int[size];
    for (int i = 0; i < size; i++)
    {
        cin >> arr[i];
    }
    int comparisons = insertion(arr, size);
    print(arr, size);
    cout << "Number of comparisons happen are; " << comparisons << endl;
    return 0;
}
```

```
Enter the size of the array
71
Enter the unsorted array
21 30 65 2 45 1 0 96 -8 75 19 91 3 6 8 45 9 6 78 45 32 19 15 57 75 46 62 26 35 59 85 47
45 8 25 52 330 5 55 8 29 24 23 54 32 93 25 87 24 89 23 21 8 720 51 21 86 33 54 11 8
9 78 91 19 36 67 76 28 84
23
Sorted array is:
-8 0 1 2 3 5 6 6 8 8 8 8 8 9 9 11 15 19 19 19 21 21 21 23 23 23 24 24 25 25 26 28 29 30
32 32 33 35 36 45 45 45 45 46 47 51 52 54 54 55 57 59 62 65 67 75 75 76 78 78 84 85 86 8
7 89 91 91 93 96 330 720
Number of comparisons happen are; 1162
```

```
Enter the size of the array
100
Enter the unsorted array
12 32 56 98 75 41 15 36 96 54 78 87 12 23 32 36 56 59 51 24 75 54 31 16 34 96 54 26 27 6
4 12 33 66 55 22 11 44 77 99 88 54 24 15 13 10 20 97 30 65 90 20 70 46 31 15 18 17 96 95
 30 28 12 87 45 96 92 94 16 78 24 256 02 20 59 95
14 75 98 36 21 45 84 256 10 79 85 86 34 56 60 79 90 14 73 91 45 26 21 70
5
Sorted array is:
2 5 10 10 11 12 12 12 12 13 14 14 15 15 15 16 16 17 18 20 20 20 21 21 22 23 24 24 24 26
26 27 28 30 30 31 31 32 32 33 34 34 36 36 36 41 44 45 45 45 46 51 54 54 54 54 55 56 56 5
6 59 59 60 64 65 66 70 70 73 75 75 75 77 78 78 79 79 84 85 86 87 87 88 90 90 91 92 94 95
 95 96 96 96 96 97 98 98 99 256 256
Number of comparisons happen are; 2489
```



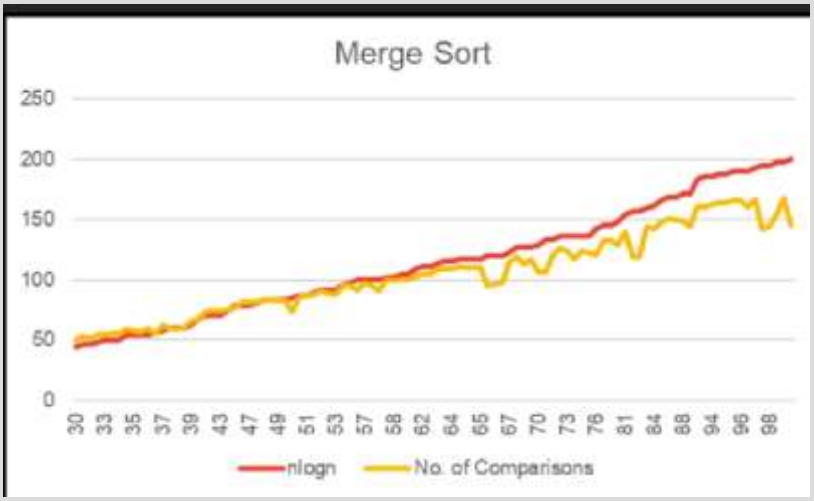
ii. Implement Merge Sort (The program should report the number of comparisons).

```
#include <iostream>
using namespace std;
void mergeSort(int arr[], int low, int high);
void merge(int arr[], int low, int mid, int high);
int count=0;
void printArr(int arr[], int n)
{
    for (int i = 0; i < n; i++)
    {
        cout << arr[i] << ' ';
    }
    cout << endl;
}
int main()
{
    int *arr;

   int size;
   cout << "Enter the size of the array" << endl;
   cin >> size;
   cout << "Enter the unsorted array" << endl;
   arr = new int[size];
   for (int i = 0; i < size; i++)
   {
      cin >> arr[i];
   }
    mergeSort(arr, 0, sizeof(arr) / sizeof(arr[0]) - 1);
    printArr(arr, sizeof(arr) / sizeof(arr[0]) - 1);
    cout<<"Total number of comparisons are: "<<count<<endl;
    return 0;
}

void mergeSort(int arr[], int low, int high)
{
    int mid;
    if (low < high)
    {
        mid = (low + high) / 2;
        mergeSort(arr, low, mid);
```

```
        mergeSort(arr, mid + 1, high);
        merge(arr, low, mid, high);
    }
}

void merge(int arr[], int low, int mid, int high)
{
    int i, j, k;
    int new_arr[50];
    i = low;
    k = low;
    j = mid + 1;
    while (i <= mid && j <= high)
    {
        count++;
        if (arr[i] < arr[j])
        {
            new_arr[k] = arr[i];
            k++;
            i++;

        }
        else
        {
            new_arr[k] = arr[j];
            k++;
            j++;

        }
    }
    while (i <= mid)
    {
        count++;
        new_arr[k] = arr[i];
        k++;
        i++;
    }
    while (j <= high)
    {
        count++;
        new_arr[k] = arr[j];
        k++;
        j++;
    }
    for (i = low; i < k; i++)
    {
        arr[i] = new_arr[i];
    }
}
```

```
Enter the size of the array
5
Enter the unsorted array
3 5 0 -8 6
-8 0 3 5
Total number of comparisons are: 12
```

```
Enter the size of the array
20
Enter the unsorted array
12 3 64 5 0 89 62 21 41 31 96 26 45 80 47 83 39 64 15 27
0 3 5 12 15 21 26 27 31 39 41 45 47 62 64 64 80 83 89
Total number of comparisons are: 88
PS D:\prachi c\algo>
```

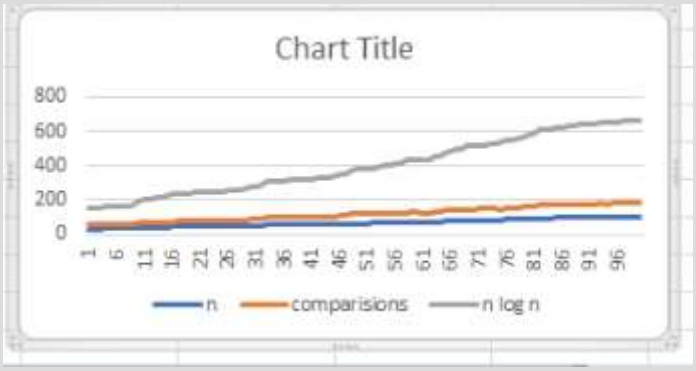


Q2) Implement Heap Sort(The program should report the number of comparisons).

```
#include<iostream>
using namespace std;
int count=0;
void heapify(int arr[], int n, int i){
    int max = i;
    int left= 2*i+1;
    int right = 2*i+2;
    if(left<n && arr[left]>arr[max]){
        max = left;
        count++;
    }
    if(right<n && arr[right]>arr[max]){
        max = right;
        count++;
    }
    if(max!=i){

        int temp = arr[i];
        arr[i] = arr[max];
        arr[max] = temp;

        heapify(arr,n,max);
    }
}

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--){
        int hold = arr[0];
        arr[0]=arr[i];
        arr[i]= hold;

        heapify(arr,i, 0);
    }
}

void printArr(int arr[], int n){
    for(int i=0; i<n; i++){
        cout<<arr[i]<<" ";
    }
    cout<<endl;
}

int main(){
    int n;
    cout<<"Enter number of elements in the array"<<endl;
    cin>>n;
    int a[n];
    cout<<"Enter elements of the array” "<<endl;
    for(int i=0; i<n; i++){
        cin>>a[i];
    }
    cout<<"Before sorting array elements are - \n";
    printArr(a, n);
    heapSort(a, n);
    cout<<"\nAfter sorting array elements are - \n";
    printArr(a, n);
    cout<<"Total number of comparisons are: "<<count<<endl;
    return 0;
}
```

```
Enter elements of the arrayΓÇ¥
12 54 85 63 214 563 987 2 0 9 5 45 87 3 99 0 148 2 1 35 15 89 26 58 41 02 35 58 95 21 21
 12 36 64 46 78 95 82 20 40 75 50 34 79 64 13 29 27 73 00
Before sorting array elements are -
12 54 85 63 214 563 987 2 0 9 5 45 87 3 99 0 148 2 1 35 15 89 26 58 41 2 35 58 95 21 21
12 36 64 46 78 95 82 20 40 75 50 34 79 64 13 29 27 73 0

After sorting array elements are -
0 0 0 1 2 2 2 3 5 9 12 12 13 15 20 21 21 26 27 29 34 35 35 36 40 41 45 46 50 54 58 58 63
 64 64 73 75 78 79 82 85 87 89 95 95 99 148 214 563 987
Total number of comparisons are: 263
```

```
 Enter number of elements in the array
 25
 Enter elements of the arrayΓÇ¥
 1 2 3 8 0 3 6 4 1 20 5 96 4 87 5 36 32 56 98 58 0 45 89 4
 56
 Before sorting array elements are -
 1 2 3 8 0 3 6 4 1 20 5 96 4 87 5 36 32 56 98 58 0 45 89 4 56

 After sorting array elements are -
 0 0 1 1 2 3 3 4 4 4 5 5 6 8 20 32 36 45 56 56 58 87 89 96 98
 Total number of comparisons are: 105
```



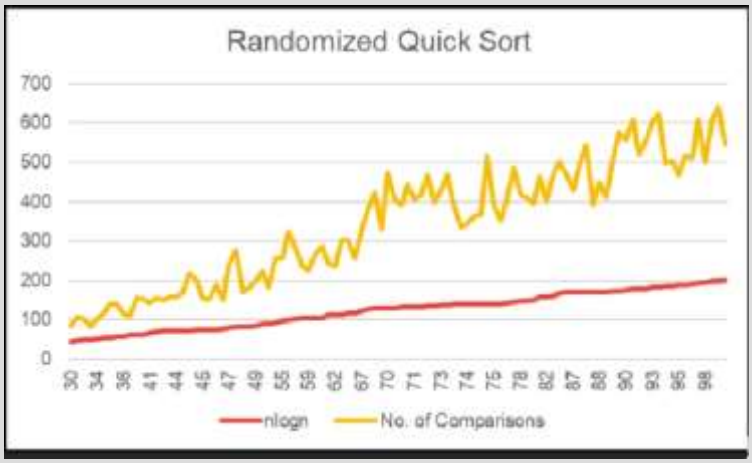
Q3) Implement Randomized Quick sort (The program should report the number of comparisons).

```cpp
#include <iostream>
using namespace std;
int count = 0;
void quickSort(int arr[], int start, int end);
int partition(int arr[], int start, int end);
void printArr(int arr[], int n)
{
    for (int i = 0; i < n; i++)
    {
        cout << arr[i] << " ";
    }
    cout << endl;
}
int main()
{
    int n;
    cout << "Enter number of elements in the array" << endl;
    cin >> n;
```

```
    int arr[n];
    cout << "Enter elements of the array” " << endl;
    for (int i = 0; i < n; i++)
    {
        cin >> arr[i];
    }
    quickSort(arr, 0, n - 1);
    printArr(arr, n);
    cout<<"The number of comparisons are: "<<count<<endl;
    return 0;
}
void quickSort(int arr[], int start, int end)
{
    if (start < end)
    {
        int p = partition(arr, start, end);
        quickSort(arr, start, p - 1);
        quickSort(arr, p + 1, end);
    }
}

int partition(int arr[], int start, int end)
{
    int pivotEle = arr[end];
    int i = start - 1;

    for (int j = start; j < end; j++)
    {
        count++;
        if (arr[j] < pivotEle)
        {
            i++;
            int hold = arr[i];
            arr[i] = arr[j];
            arr[j] = hold;
        }
    }
    int temp = arr[i + 1];
    arr[i + 1] = arr[end];
    arr[end] = temp;
    return (i + 1);
}
```

```
Enter number of elements in the array
71
Enter elements of the arrayΓÇ¥
468 335 501 170 725 479 359 963 465 706 146 282 828 962 492 996 943 828 437 392 605  903
 154 293 383 422 717 896 448 727 772  539 870 913 668  300 36 895 704 812 323 334 674 66
5 142 712 254 869 548 645 663 758 38  860 724 530 779 317 36 1 843 289 107 41 943 265 64
9 447 806
55
4
1 4 36 36 38 41 55 107 142 146 154 170 254 265 282 289 293 300 317 323 334 335 359 383 3
92 422 437 447 448 465 468 479 492 501 530 539 548 605 645 649 663 665 668 674 704 706 7
12 717 724 725 727 758 772 779 806 812 828 828 843 860 869 870 895 896 903 913 943 943 9
62 963 996
The number of comparisons are: 440
```

```
Enter number of elements in the array
100
Enter elements of the arrayΓÇ¥
12 32 21 52 45 63 35 2 10 6 98 56 6 4 78 9 55 11 33 78 85 41 16 3 0 10 20 64 46 93 39 62
 26 45 58 95 56 65 78 87 15 51 42  24 30 17 3 5 8 9 13 31 65 66 9 58 75 85 15 50 64 52 3
2 86 51 46 28 96 91 39 49 78 82 95 43 125 26 94 36 23 95 6
2 6 02 3
22 512 14 558 55 12 12 18 17 85 25 45 86 10
0 2 2 2 3 3 3 4 5 6 6 6 6 8 9 9 9 10 10 10 11 12 12 12 13 14 15 15 16 17 17 18 20 21 22
23 24 25 26 26 28 30 31 32 32 33 35 36 39 39 41 42 43 45 45 45 46 46 49 50 51 51 52 52 5
5 55 56 56 58 58 62 63 64 64 65 65 66 75 78 78 78 78 82 85 85 85 86 86 87 91 93 94 95 95
 95 96 98 125 512 558
The number of comparisons are: 668
```



Q4) Implement Radix Sort.

```cpp
#include <iostream>

using namespace std;

int getMax(int a[], int n)
{
  int max = a[0];
  for (int i = 1; i < n; i++)
  {
    if (a[i] > max)
      max = a[i];
  }
  return max;
}
```

```
void countingSort(int a[], int n, int place)
{
  int output[n + 1];
  int count[10] = {0};

  for (int i = 0; i < n; i++)
    count[(a[i] / place) % 10]++;

  for (int i = 1; i < 10; i++)
    count[i] += count[i - 1];

  for (int i = n - 1; i >= 0; i--)
  {
    output[count[(a[i] / place) % 10] - 1] = a[i];
    count[(a[i] / place) % 10]--;
  }

  for (int i = 0; i < n; i++)
    a[i] = output[i];
}

void radixsort(int a[], int n)
{

  int max = getMax(a, n);

  for (int place = 1; max / place > 0; place *= 10)
    countingSort(a, n, place);
}

void printArray(int a[], int n)
{
  for (int i = 0; i < n; ++i)
    cout << a[i] << " ";
}

int main()
{
  int a[] = {171, 279, 380, 111, 135, 726, 504, 878, 112};
  int n = sizeof(a) / sizeof(a[0]);
  cout << "Before sorting array elements are - \n";
  printArray(a, n);
  radixsort(a, n);
  cout << "\n\nAfter applying Radix sort, the array elements are - \n";
  printArray(a, n);
  return 0;
}
```

```
Before sorting array elements are -
171 279 380 111 135 726 504 878 112

After applying Radix sort, the array elements are -
111 112 135 171 279 380 504 726 878
PS D:\prachi c\algo>
```

Q5) Implement Bucket Sort.

```cpp
#include <iostream>
using namespace std;
int getMax(int arr[], int n)
{
    int max = arr[0];
    for (int i = 0; i < n; i++)
    {
        if (arr[i] > max)
        {
            max = arr[i];
        }
    }
    return max;
}
void bucket(int arr[], int n)
{
    int max = getMax(arr, n);
    int bucket[max];
    for (int i = 0; i <= max; i++)
    {
        bucket[i] = 0;
    }
    for (int i = 0; i < n; i++)
    {
        bucket[arr[i]]++;
    }
    for (int i = 0, j = 0; i <= max; i++)
    {
        while (bucket[i] > 0)
        {
            arr[j++] = i;
            bucket[i]--;
        }
    }
}
void printArr(int arr[], int n)
{
    for (int i = 0; i < n; ++i)
    {
        cout << arr[i] << " ";
    }
}
int main()
{
    int arr[] = {34, 42, 74, 57, 99, 84, 9, 5};
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Before sorting array elements are: ";
    printArr(arr, n);
    bucket(arr, n);
    cout << "\nAfter sorting array elements are: ";
    printArr(arr, n);
}
```

```
Before sorting array elements are: 34 42 74 57 99 84 9 5
After sorting array elements are: 5 9 34 42 57 74 84 99
```

Q6) Implement Randomized Select.

```cpp
#include <iostream>
#include <stdlib.h>
using namespace std;

int randomno(int lower, int upper)
{

    int num = (rand() % (upper - lower + 1)) + lower;
    return num;
}

int PARTITION(int A[], int p, int r)
{
    int x, temp;
    x = A[r];
    int i = p - 1;

    for (int j = p; j <= r - 1; j++)
    {
        if (A[j] <= x)
        {

            i = i + 1;
            temp = A[i];
            A[i] = A[j];
            A[j] = temp;
        }
    }

    temp = A[i + 1];
    A[i + 1] = A[r];
    A[r] = temp;

    return (i + 1);
}

int randomizedpartition(int A[], int p, int r)
{
    int T;

    int i = randomno(p, r);
    T = A[r];
    A[r] = A[i];
    A[i] = T;

    return PARTITION(A, p, r);
}
int randomizedselect(int A[], int p, int r, int i)
{
    if (p == r)
    {
        return A[p];
    }

    int q, k;
    q = randomizedpartition(A, p, r);
    k = q - p + 1;

    if (i == k)
    {
        return A[q];
    }
    else if (i < k)
    {
        return randomizedselect(A, p, q - 1, i);
    }

    else
    {
        return randomizedselect(A, q + 1, r, i - k);
    }
}
int main()
{
```

```
    int n, i;
    int small;
    int smallele;

    cout << "\nEnter the number of elements :  ";
    cin >> n;
    int A[n];

    cout << "\nEnter the elements :  ";
    for (i = 1; i <= n; i++)
    {
        cin >> A[i];
    }
    cout << "\n Enter the ith smallest element you want : ";
    cin >> small;

    smallele = randomizedselect(A, 1, n, small);

    cout << "The " << small << "th smallest element is\t" << smallele;
    return 0;
}
```

```
Enter the number of elements :  5

Enter the elements :  11 50 32 1 3

 Enter the ith smallest element you want : 2
The 2th smallest element is     3
PS D:\prachi c\algo>
```

Q7) Implement Breadth-First Search in a graph.

```
#include <iostream>
#include <list>
using namespace std;

class Graph
{
    int numV;
    list<int> *adjacentLists;
    bool *visited;

public:
    Graph(int v);
    void addEdge(int src, int dest);
    void BFS(int startV);
};

Graph::Graph(int v)
{
    numV = v;
    adjacentLists = new list<int>[v];
}

void Graph::addEdge(int src, int dest)
{
    adjacentLists[src].push_back(dest);
    adjacentLists[dest].push_back(src);
}

void Graph::BFS(int startV)
{
```

```
    visited = new bool[numV];
    for (int i = 0; i < numV; i++)
    {
        visited[i] = false;
    }

    list<int> queue;
    visited[startV] = true;
    queue.push_back(startV);

    list<int>::iterator i;
    cout << "BFS(starting from vertex" << startV << "): ";
    while (!queue.empty())
    {
        int curV = queue.front();
        cout << curV << " ";
        queue.pop_front();
        for (i = adjacentLists[curV].begin(); i != adjacentLists[curV].end(); ++i)
        {
            int adjV = *i;
            if (!visited[adjV])
            {
                visited[adjV] = true;
                queue.push_back(adjV);
            }
        }
    }
}

int main()
{
    Graph g(4);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);
    g.addEdge(3, 3);
    g.BFS(2);
    return 0;
}
```

```
BFS(starting from vertex2): 2 0 1 3
PS D:\prachi_c\algo>
```

Q8) Implement Depth-First Search in a graph.

```
#include <iostream>
#include <vector>
using namespace std;
const int N = 1e5 + 2;
bool visited[N];
vector<int> adj[N];

void dfs(int node)
{
  // preorder
  visited[node] = 1;
  cout << node << " ";

  // inorder
  vector<int>::iterator it;
  for (it = adj[node].begin(); it != adj[node].end(); it++)
  {
    if (visited[*it])
```

```
      ;
    else
    {
      dfs(*it);
    }
  }

  // postorder
}

int main()
{

  int n, m;
  cin >> n >> m;

  for (int i = 0; i <= n; i++)
  {
    visited[i] = false;
  }

  int x, y;
  for (int i = 0; i < m; i++)
  {
    cin >> x >> y;
    adj[x].push_back(y);
    adj[y].push_back(x);
  }

  dfs(1);

  return 0;
}
```

```
7 7
1 2
1 3
2 4
2 5
2 6
2 7
7 3
1 2 4 5 6 7 3
```

Q9) Write a program to determine the minimum spanning tree of a graph using both Prims and Kruskals algorithm.

Prims algorithm

```cpp
#include <bits/stdc++.h>
using namespace std;

#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;
}
void printMST(int parent[], int graph[V][V])
{
    cout << "Edge \tWeight\n";
    for (int i = 1; i < V; i++)
        cout << parent[i] << " - " << i << " \t"
             << graph[i][parent[i]] << " \n";
}

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;

    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);
}

// Driver's code
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}};

    cout << "\n The Adjajency Matrix for the graph is:" << endl;
    for (int i = 0; i < V; i++)
    {
        for (int j = 0; j < V; j++)
        {
            cout << graph[i][j] << "\t";
        }
        cout << "\n";
    }
    cout << endl;

    // Print the solution
    primMST(graph);

    return 0;
}
```

```
 The Adjajency Matrix for the graph is:
2        0        3        8        5
0        3        0        0        7
6        8        0        0        9
0        5        7        9        0

Edge    Weight
0 - 1   2
1 - 2   3
0 - 3   6
1 - 4   5
PS D:\prachi_c\algo>
```

Kruskal's algorithm

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5 + 6;
vector<int> parent(N);
vector<int> sz(N);
void make_set(int v)
{
    parent[v] = v;
    sz[v] = 1;
}

int find_set(int v)
{
    if (v == parent[v])
    {
        return v;
    }
    return parent[v] = find_set(parent[v]);
}

void union_sets(int a, int b)
{
    a = find_set(a);
    b = find_set(b);
    if (a != b)
    {
        if (sz[a] < sz[b])
        {
            swap(a, b);
        }
        parent[b] = a;
        sz[a] += sz[b];
    }
}

int main()
```

```
{
    for (int i = 0; i < N; i++)
    {
        make_set(i);
    }
    int n, m;
    cin >> n >> m;
    vector<vector<int>> edges;
    for (int i = 0; i < m; i++)
    {
        int u, v, w;
        cin >> u >> v >> w;
        edges.push_back({w, u, v});
    }
    sort(edges.begin(), edges.end());
    int cost = 0;
    for (auto i : edges)
    {
        int w = i[0];
        int u = i[1];
        int v = i[2];
        int x = find_set(u);
        int y = find_set(v);
        if (x == y)
        {
            continue;
        }
        else
        {
            cout << u << " " << v << endl;
            cost += w;
            union_sets(u, v);
        }
    }
    cout << cost;
}
```

```
8 9
1 2 5
2 3 6
4 3 2
1 4 9
3 5 5
5 6 10
6 7 7
7 8  1
8 5 1
```

```
7 8
8 5
4 3
1 2
3 5
2 3
6 7
27
```

Q10) Write a program to solve the weighted interval scheduling problem.

```cpp
#include <bits/stdc++.h>
using namespace std;
struct Job
{
    int start, finish, profit;
};

bool jobCompare(Job s1, Job s2)
{
    return (s1.finish < s2.finish);
}

int latestNonConflict(Job arr[], int i)
{
    for (int j = i - 1; j >= 0; j--)
    {
        if (arr[j].finish <= arr[i - 1].start)
            return j;
    }
    return -1;
}

int findMaxProfitRec(Job arr[], int n)
{
    if (n == 1)
        return arr[n - 1].profit;

    int incProf = arr[n - 1].profit;
    int i = latestNonConflict(arr, n);
    if (i != -1)
    {
        incProf += findMaxProfitRec(arr, i + 1);
    }
    int excludedProfit = findMaxProfitRec(arr, n - 1);
    return max(incProf, excludedProfit);
}

int findMaxProfit(Job arr[], int n)
{
    sort(arr, arr + n, jobCompare);
    return findMaxProfitRec(arr, n);
}

int main()
{
    int n;
    cout << "Enter the number of jobs to be done" << endl;
    cin >> n;
    Job jobs[n];
    cout << "Enter the starting , finishing and profit of each job respectively" << endl;
    cout << "Start finish profit" << endl;
    for (int i = 0; i < n; i++)
    {
        cin >> jobs[i].start >> jobs[i].finish >> jobs[i].profit;
    }
    cout << "The optimal profit is " << findMaxProfit(jobs, n);
    return 0;
}
```

```
Enter the number of jobs to be done4
Enter the starting , finishing and profit of each job respectively
Start finish profit
1 2 50
3 5 20
6 19 100
2 100 200
The optimal profit is 250
```

Q11) Write a program to solve the 0-1 knapsack problem.

```cpp
#include <iostream>
using namespace std;

int max(int a, int b)
{
    return (a > b) ? a : b;
}

int knapSack(int W, int wt[], int val[], int n)
{
    if (n == 0 || W == 0) // base case
        return 0;

    if (wt[n - 1] > W)
        return knapSack(W, wt, val, n - 1);

    else
        return max(
            val[n - 1] + knapSack(W - wt[n - 1], wt, val, n - 1),
            knapSack(W, wt, val, n - 1));
}

int main()
{
    int n;

    int W = 10;
    cout << "\nEnter the number of items :  ";
    cin >> n;
    int val[n], wt[n];
    cout << "\nEnter the values of item :  ";
    for (int i = 0; i < n; i++)
    {
        cin >> val[i];
    }

    cout << "\nEnter the weights of item :  ";
    for (int i = 0; i < n; i++)
    {
        cin >> wt[i];
    }

    cout << "\nThe knapsack capacity is \t" << W;
    cout << "\n\nThe values with their weight :\n";
    cout << "\nValues\t\tWeight\t\n";
    for (int i = 0; i < 3; i++)
    {
        cout << val[i] << "\t\t" << wt[i] << "\n";
    }

    cout << "\nThe optimal value is \t" << knapSack(W, wt, val, n);
    return 0;
}
```

```
Enter the number of items :  3
Enter the values of item :  7 6 11
Enter the weights of item :  4 3 5
The knapsack capacity is        10
The values with their weight :
Values          Weight
7               4
6               3
11              5

The optimal value is    18
```