



RAMANUJAN COLLEGE

DELHI UNIVERSITY

DAA PRACTICAL FILE

Name – **Riya Dhawan**

Roll No. – **20191434/19020570031**

Course – **B. Sc. (H) Computer Science**

College – **Ramanujan College, Delhi University**

Submitted to – **Mr. Vipin Rathi**

Q1.

a) Implement Insertion Sort (The program should report the number of comparisons)

```
#include <bits/stdc++.h>
```

```
using namespace std;
```

```
int insertionSort(int arr[], int n)
```

```
{
```

```
int i, key, j, count;
```

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

```

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

    while (j >= 0 )
    {if(arr[j] > key){
        arr[j + 1] = arr[j];
        j = j - 1;
        count++;}
        else{
            count++;
            break;
        }
    }
    arr[j + 1] = key;
}
return count;
}

```

```

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

```


```

int main()
{
    int size,comp,ext;

```

```
begin:
cout<<"Enter the size of array : ";
cin>>size;
int arr[size];
cout<<"\n\n";
for(int i=0;i<size;i++)
{
    cout<<"Enter element "<<i+1<<" in an array : ";
    cin>>arr[i];
}

    cout << "\n\nGiven array is : ";
printArray(arr, size);
comp = insertionSort(arr, size);
cout<<"\n\nSorted array is: ";
printArray(arr, size);
cout<<"\nNo of comparisons are : "<<comp;
cout<<"\n\nPress 1 to start again / any other key to exit : ";
    cin>>ext;
    if(ext == 1)
        goto begin;
return 0;
}
```

 F:\Users\a\Desktop\daa practical\Insertion Sort(1.a).exe

```
Enter the size of array :  
5  
  
Enter element 1 in an array : 12  
Enter element 2 in an array : 445  
Enter element 3 in an array : 51  
Enter element 4 in an array : 120  
Enter element 5 in an array : 11  
  
Given array is :    12 445 51 120 11  
  
Sorted array is:    11 12 51 120 445  
  
No of comparisons are : 9  
  
Press 1 to start again / any other key to exit :
```

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

```
#include <iostream>
```

```
using namespace std;
```

```
int comp =0;
```

```
void merge(int arr[], int l, int m, int r)
```

```
{
```

```
int n1 = m - l + 1;
```

```
int n2 = r - m;
```

```
int L[n1], R[n2];
```

```
for (int i = 0; i < n1; i++)
```

```
    L[i] = arr[l + i];
```

```
for (int j = 0; j < n2; j++)
```

```
    R[j] = arr[m + 1 + j];
```

```
int i = 0;
```

```
int j = 0;
```

```
int k = l;
```

```
while (i < n1 && j < n2) {
```

```
    if (L[i] <= R[j]) {
```

```
        arr[k] = L[i];
```

```
        i++;
```

```
        comp++;
```

```
    }
```

```
    else {
```

```
        arr[k] = R[j];
```

```
        j++;
```

```
        comp++;
```

```
    }
```

```
    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){
```

```
return;
```

```
}
```

```
int m =l+ (r-l)/2;
```

```
mergeSort(arr,l,m);
```

```
mergeSort(arr,m+1,r);
```

```
merge(arr,l,m,r);
```

```
}
```

```
void printArray(int arr[], int n)
```

```
{
```

```
int i;
```

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

```
cout << arr[i] << " ";
```

```
cout << endl;
```

```
}
```

```
int main()
```

```
{
```

```
int size,ext;
```

```
    begin:
```

```
        comp=0;
```

```
cout<<"Enter the size of array : ";
```

```
cin>>size;
```

```
int arr[size];
```

```
cout<<"\n\n";
```

```
for(int i=0;i<size;i++)
```

```
{
```

```
cout<<"Enter element "<<i+1<<" in an array : ";
```

```
cin>>arr[i];
```

```
}
```

```
    cout << "\n\nGiven array is :  ";
```

```
printArray(arr, size);
```

```
mergeSort(arr, 0, size - 1);
```

```
cout<<"\n\nSorted array is:  ";
```

```
printArray(arr, size);
```


```
cout<<"\nNo of comparisons are : "<<comp;
```

```
cout<<"\n\nPress 1 to start again / any other key to exit : ";
```

```
    cin>>ext;
```

```
    if(ext == 1)
```

```
    goto begin;
return 0;
}
```

 F:\Users\a\Desktop\daa practical\Merge Sort(1.b).exe

```
Enter the size of array : 5

Enter element 1 in an array : 125
Enter element 2 in an array : 22220
Enter element 3 in an array : 123546
Enter element 4 in an array : 6542
Enter element 5 in an array : 2131

Given array is :    125 22220 123546 6542 2131

Sorted array is:    125 2131 6542 22220 123546

No of comparisons are : 7

Press 1 to start again / any other key to exit :
```

Q2. Implement Heap Sort (The program should report the number of comparisons). **#include**
#include <iostream>


```
using namespace std;
```

```
int comp =0;
```

```
void heapify(int arr[], int n, int i)
```

```
{
```

```
int largest = i;
```

```
int l = 2 * i + 1;
```

```
int r = 2 * i + 2;
```

```
if (l < n ){
```

```
if(arr[l] > arr[largest])
```

```
largest = l;
```

```
comp++;
```

```
}
```

```
if (r < n ){
```

```
if(arr[r] > arr[largest])
```

```
largest = r;
```

```
comp++;
```

```
}
```

```
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);
```

```
}
```

```
}
```

```
void printArray(int arr[], int n)
```

```
{
```

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

```
        cout << arr[i] << " ";
```

```
    cout << "\n";
```

```
}
```

```
int main()
```

```
{
```

```
    int size,ext;
```


```
    begin:
```

```
        comp=0;
cout<<"Enter the size of array : ";
cin>>size;
int arr[size];
cout<<"\n\n";
for(int i=0;i<size;i++)
{
    cout<<"Enter element "<<i+1<<" in an array : ";
    cin>>arr[i];
}

    cout << "\n\nGiven array is : ";
    printArray(arr, size);

    heapSort(arr, size);

    cout<<"\n\nSorted array is: ";
    printArray(arr, size);
    cout<<"\nNo of comparisons are : "<<comp;
    cout<<"\n\nPress 1 to start again / any other key to exit : ";
        cin>>ext;
        if(ext == 1)
            goto begin;
    return 0;}
```

 F:\Users\a\Desktop\daa practical\Heap Sort (2 que).exe

```
Enter the size of array : 5

Enter element 1 in an array : 1320302
Enter element 2 in an array : 31651
Enter element 3 in an array : 316461
Enter element 4 in an array : 323246846
Enter element 5 in an array : 3321

Given array is :    1320302 31651 316461 323246846 3321

Sorted array is:    3321 31651 316461 1320302 323246846

No of comparisons are : 12

Press 1 to start again / any other key to exit :
```

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

```
#include <cstdlib>
```

```
#include <iostream>
```

```
#include <time.h>
```

```
using namespace std;
```

```
int comp =0;
```

```
int partition(int arr[], int low, int high)
```

```
{
```

```
int pivot = arr[high];
```

```
int i = (low - 1);
```

```
for (int j = low; j <= high - 1; j++)
```

```
{
```

```
    if (arr[j] <= pivot) {
```

```
        i++;
```

```
        swap(arr[i], arr[j]);
```

```
    }
```

```
    comp++;
```

```
}
```

```
swap(arr[i + 1], arr[high]);
```

```
return (i + 1);
```

```
}
```

```
int partition_r(int arr[], int low, int high)
```

```
{
```

```
    srand(time(NULL));
```

```
    int random = low + rand() % (high - low);
```

```
swap(arr[random], arr[high]);
```

```
return partition(arr, low, high);
```

```
}
```

```
void quickSort(int arr[], int low, int high)
```

```
{
```

```
if (low < high) {
```

```
    int pi = partition_r(arr, low, high);
```

```
    quickSort(arr, low, pi - 1);
```

```
    quickSort(arr, pi + 1, high);
```

```
}
```

```
}
```

```
void printArray(int arr[], int n)
```

```
{
```

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

```
    cout << arr[i] << " ";
```

```
cout << "\n";
```

```
}
```

```
int main()
```

```
{
```

```
int size,ext;

begin:

    comp=0;

cout<<"Enter the size of array : ";

cin>>size;

int arr[size];

cout<<"\n\n";

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

    cout<<"Enter element "<<i+1<<" in an array : ";

    cin>>arr[i];

}


    cout << "\n\nGiven array is :  ";

printArray(arr, size);


quickSort(arr, 0, size - 1);


cout<<"\n\nSorted array is:  ";

printArray(arr, size);

cout<<"\nNo of comparisons are : "<<comp;

cout<<"\n\nPress 1 to start again / any other key to exit : ";


    cin>>ext;

    if(ext == 1)

        goto begin;

return 0;
```

```
}
```

 F:\Users\a\Desktop\daa practical\Randomized Quick Sort(3 que).exe

```
Enter the size of array : 6

Enter element 1 in an array : 313213
Enter element 2 in an array : 24546168
Enter element 3 in an array : 65489
Enter element 4 in an array : 1329256
Enter element 5 in an array : 5132354
Enter element 6 in an array : 321032

Given array is :      313213 24546168 65489 1329256 5132354 321032

Sorted array is:      65489 313213 321032 1329256 5132354 24546168

No of comparisons are : 8

Press 1 to start again / any other key to exit :
```

Q4. Implement Radix Sort.

```
#include <iostream>
```

```
using namespace std;
```

```
int getMax(int arr[], int n)
```

```
{
```

```
    int mx = arr[0];
```



```
        for (int i = 1; i < n; i++)
            if (arr[i] > mx)
                mx = arr[i];
        return mx;
    }
```

```
void countSort(int arr[], int n, int exp)
```

```
{
    int output[n];
    int i, count[10] = { 0 };
    for (i = 0; i < n; i++)
        count[(arr[i] / exp) % 10]++;

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

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

    for (i = 0; i < n; i++)
        arr[i] = output[i];
}
```

```
void radixsort(int arr[], int n)
```

```
{
```

```

        int m = getMax(arr, n);

        for (int exp = 1; m / exp > 0; exp *= 10)
            countSort(arr, n, exp);
    }

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

    int main()
    {
        int size, ext;

        begin:
        cout << "Enter the size of array : ";
        cin >> size;
        int arr[size];
        cout << "\n\n";
        for (int i = 0; i < size; i++)
        {
            cout << "Enter element "<< i + 1 << " in an array : ";
            cin >> arr[i];
        }

        cout << "\n\nGiven array is : ";
    }
}

```


```

    printArray(arr, size);

    radixsort(arr, size);

    cout<<"\n\nSorted array is: ";
    printArray(arr, size);
    cout<<"\n\nPress 1 to start again / any other key to exit : ";
    cin>>ext;
    if(ext == 1)
        goto begin;
    return 0;
}

```

 F:\Users\A\Desktop\daa practical\Radix Sort(4 que).exe

```

Enter the size of array : 10

Enter element 1 in an array : 21321
Enter element 2 in an array : 4546
Enter element 3 in an array : 78798
Enter element 4 in an array : 45421
Enter element 5 in an array : 2132132
Enter element 6 in an array : 01213245
Enter element 7 in an array : 21321
Enter element 8 in an array : 00000213
Enter element 9 in an array : 2148
Enter element 10 in an array : 51212

Given array is :      21321 4546 78798 45421 2132132 1213245 21321 213 2148 51212

Sorted array is:      213 2148 4546 21321 21321 45421 51212 78798 1213245 2132132

Press 1 to start again / any other key to exit : █

```

Q5. Create a Red-Black Tree and perform following operations on it: i. Insert a node
ii. Delete a node iii. Search for a number & also report the color of the node
containing this number.

```
#include <cstdlib>
```

```
#include <iostream>
```

```
#include <time.h>
```

```
using namespace std;
```

```
int comp =0;
```

```
int partition(int arr[], int low, int high)
```

```
{
```

```
    int pivot = arr[high];
```

```
    int i = (low - 1);
```

```
    for (int j = low; j <= high - 1; j++)
```

```
    {
```

```
        if (arr[j] <= pivot) {
```

```
            i++;
```

```
            swap(arr[i], arr[j]);
```

```
        }
```

```
    comp++;
```

```
}
```

```
swap(arr[i + 1], arr[high]);
```

```
return (i + 1);
```

```
}
```

```
int partition_r(int arr[], int low, int high)
```

```
{
```

```
    srand(time(NULL));
```

```
    int random = low + rand() % (high - low);
```

```
    swap(arr[random], arr[high]);
```

```
    return partition(arr, low, high);
```

```
}
```

```
void quickSort(int arr[], int low, int high)
```

```
{
```

```
    if (low < high) {
```

```
int pi = partition_r(arr, low, high);
```

```
quickSort(arr, low, pi - 1);
```

```
quickSort(arr, pi + 1, high);
```

```
}
```

```
}
```

```
void printArray(int arr[], int n)
```

```
{
```

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

```
cout << arr[i] << " ";
```

```
cout << "\n";
```

```
}
```

```
int main()
```

```
{
```

```
int size,ext;
```

```
begin:
```

```
        comp=0;

cout<<"Enter the size of array : ";

cin>>size;

int arr[size];

cout<<"\n\n";

for(int i=0;i<size;i++)

{

    cout<<"Enter element "<<i+1<<" in an array : ";

    cin>>arr[i];

}

    cout << "\n\nGiven array is :  ";

    printArray(arr, size);

    quickSort(arr, 0, size - 1);

    cout<<"\n\nSorted array is:  ";

    printArray(arr, size);

    cout<<"\nNo of comparisons are : "<<comp;

    cout<<"\n\nPress 1 to start again / any other key to exit : ";


    cin>>ext;
```

```
        if(ext == 1)


            goto begin;

return 0;


}
```

 F:\Users\a\Desktop\daa practical\Red_Black_Tree(5 que).exe

```
Enter the number to be inserted in tree.
2212_
```

 F:\Users\a\Desktop\daa practical\Red_Black_Tree(5 que).exe

```
Enter number to be deleted.
12121
Press any key to continue . . .
```


 F:\Users\a\Desktop\daa practical\Red_Black_Tree(5 que).exe

Enter number to be searched.

1212123

Number is not present.Press any key to continue . .

Q6. Write a program to determine the LCS of two given sequences

```
#include<iostream>
```

```
#include<bits/stdc++.h>
```

```
#include<string>
```

```
using namespace std;
```

```
int max(int a, int b);
```

```
int lcs( string &X, string &Y, int m, int n )
```

```
{
```

```
    int L[m + 1][n + 1];
```

```
    for (int i = 0; i <= m; i++)
```

```
    {
```

```
        for (int j = 0; j <= n; j++)
```

```
        {
```

```
            if (i == 0 || j == 0)
```

```

        L[i][j] = 0;

    else if (X[i - 1] == Y[j - 1])
        L[i][j] = L[i - 1][j - 1] + 1;

    else
        L[i][j] = max(L[i - 1][j], L[i][j - 1]);
    }
}

//printing LCS
string tmp;
int i = m, j = n;
while(i!=0&& j!=0){
    if(L[i][j]==L[i][j-1])
        j--;
    else if(L[i][j]==L[i-1][j])
        i--;
    else{
        tmp+=Y[j-1];
        i--;
        j--;
    }
}

reverse(tmp.begin(),tmp.end());
cout<<"\n\nLCS is : "<<tmp<<"\n\n";

return L[m][n];

```

```
}
```

```
int max(int a, int b)
```

```
{
```

```
    return (a > b)? a : b;
```

```
}
```

```
int main()
```

```
{  int ext;
```

```
    string st1 ;
```

```
    string st2 ;
```

```
begin:
```

```
    cout<<"Enter the first sequence : ";
```

```
    cin>>st1;
```

```
    cout<<"Enter the Second sequence : ";
```

```
    cin>>st2;
```

```
    int lm = st1.length();
```

```
    int ln = st2.length();
```

```
    cout << "Length of LCS is "
```


```
        << lcs( st1, st2, lm, ln );
```

```
    cout<<"\n\nPress 1 to start again / any other key to exit : ";
```

```
    cin>>ext;
```

```
    if(ext == 1)
```

```
    goto begin;
    return 0;
}
```

 F:\Users\A\Desktop\daa practical\LCS (6 que).exe

```
Enter the first sequence : 13213asd21asdns1kfjnskd fsj
Enter the Second sequence : 2135465d4vsd5v4sd5c4s5s

LCS is : 213dsdsds
Length of LCS is 9
Press 1 to start again / any other key to exit :
```

Q7. Implement Breadth-First Search in a graph

```
#include<iostream>

#include <list>

using namespace std;

class Graph
```

```
{  
    int V;  
  
    list<int> *adj;  
public:  
    Graph(int V);  
    void addEdge(int v, int w);  
  
    void BFS(int s,int strt);  
};  
  
Graph::Graph(int V)  
{  
    this->V = V;  
    adj = new list<int>[V];  
}  
  
void Graph::addEdge(int v, int w)  
{  
    adj[v].push_back(w);  
}  
  
void Graph::BFS(int s,int strt)  
{  
    bool *visited = new bool[V];  
    for(int i = 0; i < V; i++)
```

```
visited[i] = false;
```

```
list<int> queue;
```

```
visited[s] = true;
```

```
queue.push_back(s);
```

```
list<int>::iterator i;
```

```
while(!queue.empty())
```

```
{
```

```
    s = queue.front();
```

```
    cout << s + strt << " ";
```

```
    queue.pop_front();
```

```
    for (i = adj[s].begin(); i != adj[s].end(); ++i)
```

```
    {
```

```
        if (!visited[*i])
```

```
        {
```

```
            visited[*i] = true;
```

```
            queue.push_back(*i);
```

```
        }
```

```
    }
```

```
}
```

```
}
```

```

int main()
{
    int nv,ne,start,dir,help,ep,stf,ext;

    begin:

    cout<<"Enter the no. of vertices : ";

    cin>>nv;

    cout<<"Enter the no. of edges : ";

    cin>>ne;

    if(nv<0 || ne<0){
        cout<<"\n\nError : no. of vertices or edges cannot be zero.\n\n";
        goto begin;
    }

    cout<<"Enter the no. from which graph starts : ";

    cin>>start;

    cout<<"Is the graph directional (0 = no/other number for yes) : ";

    cin>>dir;

    Graph g(nv);

    cout<<"Press 0 to enter edges manually or Press 1 to enter edges with help : ";

    cin>>help;

    cout<<endl;

    if(help==1){
        if(dir==0){
            for(int i = 0;i<nv;i++){
                for(int j = i;j<nv;j++){
                    cout<<"Is there a edge between "<<i+start<<" and "<<j+start<<" vertices
:(0=n/1=y) : ";

```

```

        cin>>ep;
        if(ep==1){
            g.addEdge(i,j);
            g.addEdge(j,i);
        }
    }
}

else{
    for(int i = 0;i<nv;i++){
        for(int j = 0;j<nv;j++){
            cout<<"Is there a edge between "<<i+start<<" and "<<j+start<<" vertices
:(0=n/1=y) : ";

            cin>>ep;
            if(ep==1){
                g.addEdge(i,j);
            }
        }
    }
}

else if(help ==0){
    if(dir==0){
        while(ne!=0){
            int i,j;

            cout<<"Enter both vertices of an edge with a space between them : ";

            cin>>i>>j;

            g.addEdge(i-start,j-start);

```



```

                g.addEdge(j-start,i-start);
                ne--;
            }
        }
    else{
        while(ne!=0){
            int i,j;
            cout<<"Enter starting and ending vertices of an edge with a space between them : ";

            cin>>i>>j;
            g.addEdge(i-start,j-start);
            ne--;
        }
    }
}

cout<<"\n\nEnter the vertex from which you want to start the traversal : ";
cin>>stf;

    cout << "\n\nFollowing is Breadth First Traversal "
        << "(starting from vertex "<<stf<<") : ";


    g.BFS(stf-start,start);

cout<<"\n\nPress 1 to search again / any other key to exit : ";
cin>>ext;

if(ext == 1)
    goto begin;

    return 0;
}

```

 F:\Users\A\Desktop\daa practical\Breadth First Search (7 que).exe

```
Enter the no. of vertices : 1
Enter the no. of edges : 2
Enter the no. from which graph starts : 1
Is the graph directional (0 = no/other number for yes) : 0
Press 0 to enter edges manually or Press 1 to enter edges with help : 1

Is there a edge between 1 and 1 vertices :(0=n/1=y) : 0

Enter the vertex from which you want to start the traversal : 1

Following is Breadth First Traversal (starting from vertex 1) :    1

Press 1 to search again / any other key to exit : █
```

Q8. Implement Depth-First Search in a graph.

```
#include<iostream>
```

```
#include <list>
```

```
using namespace std;
```

```
int start;
```

```
class Graph {
```

```
    int numVertices;
```

```
    list<int> *adjLists;
```

```
    bool *visited;
```

```
public:
    Graph(int V);
    void addEdge(int src, int dest);
    void DFS(int vertex);
};

Graph::Graph(int vertices) {
    numVertices = vertices;
    adjLists = new list<int>[vertices];
    visited = new bool[vertices];
}

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

void Graph::DFS(int vertex) {
    visited[vertex] = true;
    list<int> adjList = adjLists[vertex];

    cout << vertex +start<< " ";

    list<int>::iterator i;
    for (i = adjList.begin(); i != adjList.end(); ++i)
        if (!visited[*i])
            DFS(*i);
}
```

```
int main()
{
    int nv,ne,dir,help,ep,stf,ext;

    begin:

    cout<<"Enter the no. of vertices : ";

    cin>>nv;

    cout<<"Enter the no. of edges : ";

    cin>>ne;

    if(nv<0 || ne<0){
        cout<<"\n\nError : no. of vertices or edges cannot be zero.\n\n";
        goto begin;
    }

    cout<<"Enter the no. from which graph starts : ";

    cin>>start;

    cout<<"Is the graph directional (0 = no/other number for yes) : ";

    cin>>dir;

    Graph g(nv);

    cout<<"Press 0 to enter edges manually or Press 1 to enter edges with help : ";

    cin>>help;

    cout<<endl;

    if(help==1){
        if(dir==0){
```

```

        for(int i = 0;i<nv;i++){
            for(int j = i;j<nv;j++){
                cout<<"Is there a edge between "<<i+start<<" and "<<j+start<<"
vertices :(0=n/1=y) : ";

                cin>>ep;

                if(ep==1){
                    g.addEdge(i,j);
                    g.addEdge(j,i);
                }
            }
        }
    }
}

else{
    for(int i = 0;i<nv;i++){
        for(int j = 0;j<nv;j++){
            cout<<"Is there a edge between "<<i+start<<" and "<<j+start<<"
vertices :(0=n/1=y) : ";

            cin>>ep;

            if(ep==1){
                g.addEdge(i,j);
            }
        }
    }
}

}

else if(help ==0){
    if(dir==0){
        while(ne!=0){

```

```

        int i,j;

        cout<<"Enter both vertices of an edge with a space between them : ";

        cin>>i>>j;

        g.addEdge(i-start,j-start);

        g.addEdge(j-start,i-start);

        ne--;

    }

}

else{

while(ne!=0){

    int i,j;

    cout<<"Enter starting and ending vertices of an edge with a space between them : ";

    cin>>i>>j;

    g.addEdge(i-start,j-start);

    ne--;

}

}

}

```

```

    cout<<"\n\nEnter the vertex from which you want to start the traversal : ";

    cin>>stf;

cout << "\n\nFollowing is Depth First Traversal "

<< "(starting from vertex "<<stf<<") : ";

g.DFS(stf-start);

cout<<"\n\nPress 1 to search again / any other key to exit : ";

cin>>ext;

if(ext == 1)

```


```

    goto begin;

return 0;

}

```

 F:\Users\A\Desktop\daa practical\Depth First Search (8 que).exe

```

Enter the no. of vertices : 2
Enter the no. of edges : 2
Enter the no. from which graph starts : 1
Is the graph directional (0 = no/other number for yes) : 1
Press 0 to enter edges manually or Press 1 to enter edges with help : 1

Is there a edge between 1 and 1 vertices :(0=n/1=y) : 1
Is there a edge between 1 and 2 vertices :(0=n/1=y) : 1
Is there a edge between 2 and 1 vertices :(0=n/1=y) : 1
Is there a edge between 2 and 2 vertices :(0=n/1=y) : 1

Enter the vertex from which you want to start the traversal : 1

Following is Depth First Traversal (starting from vertex 1) :   1 2

Press 1 to search again / any other key to exit :

```

Q9. Write a program to determine the minimum spanning tree of a graph

```
#include <bits/stdc++.h>
```

```
using namespace std;
```

```
int minKey(int key[], bool mstSet[], int V)
```

```
{
```

```

// Initialize min value
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, int V, int strt)
{
    cout<<"Edge \tWeight\n";
    for (int i = 1; i < V; i++)
        cout<<parent[i]+strt<<" - "<<i+strt<<" \t"<<graph[i][parent[i]]<<" \n";
}

void primMST(int **graph,int V, int strt)
{
    // Array to store constructed MST
    int parent[V];

    // Key values used to pick minimum weight edge in cut
    int key[V];

    // To represent set of vertices included in MST
    bool mstSet[V];

```



```

// Initialize all keys as INFINITE
for (int i = 0; i < V; i++)
    key[i] = INT_MAX, mstSet[i] = false;

// Always include first 1st vertex in MST.
// Make key 0 so that this vertex is picked as first vertex.
key[0] = 0;
parent[0] = -1; // First node is always root of MST

// The MST will have V vertices
for (int count = 0; count < V - 1; count++)
{
    // Pick the minimum key vertex from the
    // set of vertices not yet included in MST
    int u = minKey(key, mstSet, V);

    // Add the picked vertex to the MST Set
    mstSet[u] = true;

    // Update key value and parent index of
    // the adjacent vertices of the picked vertex.
    // Consider only those vertices which are not
    // yet included in MST
    for (int v = 0; v < V; v++)

        // graph[u][v] is non zero only for adjacent vertices of m

```

```
        // mstSet[v] is false for vertices not yet included in MST
        // Update the key only if graph[u][v] is smaller than key[v]
        if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v])
            parent[v] = u, key[v] = graph[u][v];
    }
}
```

```
// print the constructed MST
printMST(parent, graph, V, strt);
}
```

```
// Driver code
```

```
int main()
{
    int nv, ne, start, help, ew, ext;

    begin:
    cout<<"Enter the no. of vertices : ";
    cin>>nv;

    cout<<"Enter the no. of edges : ";
    cin>>ne;

    if(nv<0 || ne<0){
        cout<<"\n\nError : no. of vertices or edges cannot be zero.\n\n";
        goto begin;
    }

    cout<<"Enter the no. from which graph starts : ";
    cin>>start;
```

```
int **graph = new int*[nv];
```

```
for(int i=0;i<nv;i++){
```

```
graph[i]=new int[nv];
```

```
}
```

```
for(int i=0;i<nv;i++){
```

```
for(int j=0;j<nv;j++){
```

```
graph[i][j]=0;
```

```
}
```

```
}
```

```
cout<<"Press 0 to enter edges manually or Press 1 to enter edges with help : ";
```

```
cin>>help;
```

```
cout<<endl;
```

```
if(help==1){
```

```
for(int i = 0;i<nv;i++){
```

```
for(int j = i;j<nv;j++){
```

```
err1:
```

```
cout<<"Enter weight of the edge between "<<i+start<<" and
```

```
"<<j+start<<" vertices :(0 if no edge present) : ";
```

```
cin>>ew;
```

```
if(ew>=0){
```

```
graph[i][j]=ew;
```

```
graph[j][i]=ew;
```

```
}
```

```
else{
```

```
cout<<"\n\nError : weight of an edge cannot be zero.\n\n";
```

```

        goto err1;
    }
}
}
}

else if(help ==0){

while(ne!=0){
    int i,j;
    cout<<"Enter starting vertex, ending vertex and weight of an edge with a
space between all of them : ";
    cin>>i>>j>>ew;
    graph[i-start][j-start]=ew;
    graph[j-start][i-start]=ew;
    ne--;
}
}

/* cout<<"\n\nEnter the vertex from which you want to start the traversal : ";
cin>>stf;
cout << "\n\nFollowing is Breadth First Traversal "
<< "(starting from vertex "<<stf<<") : ";
g.BFS(stf-start,start);*/
primMST(graph,nv,start);
cout<<"\n\nPress 1 to search again / any other key to exit : ";
cin>>ext;
if(ext == 1)

```


```
        goto begin;

return 0;

// Print the solution

return 0;

}
```

 F:\Users\A\Desktop\daa practical\Minimum Spanning Tree (9 que).exe

```
Enter the no. of vertices : 2
Enter the no. of edges : 1
Enter the no. from which graph starts : 1
Press 0 to enter edges manually or Press 1 to enter edges with help : 1

Enter weight of the edge between 1 and 1 vertices :(0 if no edge present) : 1
Enter weight of the edge between 1 and 2 vertices :(0 if no edge present) : 2
Enter weight of the edge between 2 and 2 vertices :(0 if no edge present) : 3
Edge      Weight
1 - 2      2

Press 1 to search again / any other key to exit : _
```

Comparisons of different sorts using graphs

```
#include <cstdlib>
#include <iomanip>
#include <iostream>
#include <time.h>
```

#include <iostream>

using namespace std;

int compmerge =0,compins =0,compquick =0,compheap =0;

void merge(int arr[], int l, int m, int r)

{

int n1 = m - l + 1;

int n2 = r - m;

int L[n1], R[n2];

for (int i = 0; i < n1; i++)

 L[i] = arr[l + i];

for (int j = 0; j < n2; j++)

 R[j] = arr[m + 1 + j];

int i = 0;

int j = 0;

int k = l;

while (i < n1 && j < n2) {

 if (L[i] <= R[j]) {

 arr[k] = L[i];

 i++;

 compmerge++;

 }

 else {

```
arr[k] = R[j];  
j++;  
compmerge++;  
}  
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){  
return;  
}  
int m =l+ (r-l)/2;  
mergeSort(arr,l,m);  
mergeSort(arr,m+1,r);  
merge(arr,l,m,r);s
```

```
}
```

```
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 )
```

```
{if(arr[j] > key){
```

```
arr[j + 1] = arr[j];
```

```
j = j - 1;
```

```
compins++;}
```

```
else{
```

```
compins++;
```

```
break;
```

```
}
```

```
}
```

```
arr[j + 1] = key;
```

```
}
```

```
// return count;
```

```
}
```

```
int partition(int arr[], int low, int high)
```

```
{
```


int pivot = arr[high];

int i = (low - 1);

for (int j = low; j <= high - 1; j++)

{

if (arr[j] <= pivot) {

_____ i++;

_____ swap(arr[i], arr[j]);

_____ }

_____ compquick++;

}

swap(arr[i + 1], arr[high]);

return (i + 1);

}

int partition r(int arr[], int low, int high)

{

int random = low + rand() % (high - low);

.

swap(arr[random], arr[high]);

return partition(arr, low, high);

}

void quickSort(int arr[], int low, int high)

```

{
if (low < high) {
    int pi = partition r(arr, low, high);

    quickSort(arr, low, pi - 1);
    quickSort(arr, pi + 1, high);
}
}

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

void heapify(int arr[], int n, int i)
{
int largest = i; // Initialize largest as root
int l = 2 * i + 1; // left = 2*i + 1
int r = 2 * i + 2; // right = 2*i + 2
// If left child is larger than root
if (l < n){
    if(arr[l] > arr[largest])
        largest = l;
    compheap++;
}
.
if (r < n ){

```

```

    if(arr[r] > arr[largest])
        largest = r;
    compheap++;
}
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 ext;
    int randomnum, randomval;
    cout<<"          no of comparisons required by \n";
    cout<<"          -----\n";
    cout<<"no. of values input  merge sort  insertion sort  Rquick sort  heap sort\n";
    for(int i=0;i<100;i++){
        compmerge =0,compins =0,compquick =0,compheap =0;
        randomnum = 30 + rand() % (970);

```

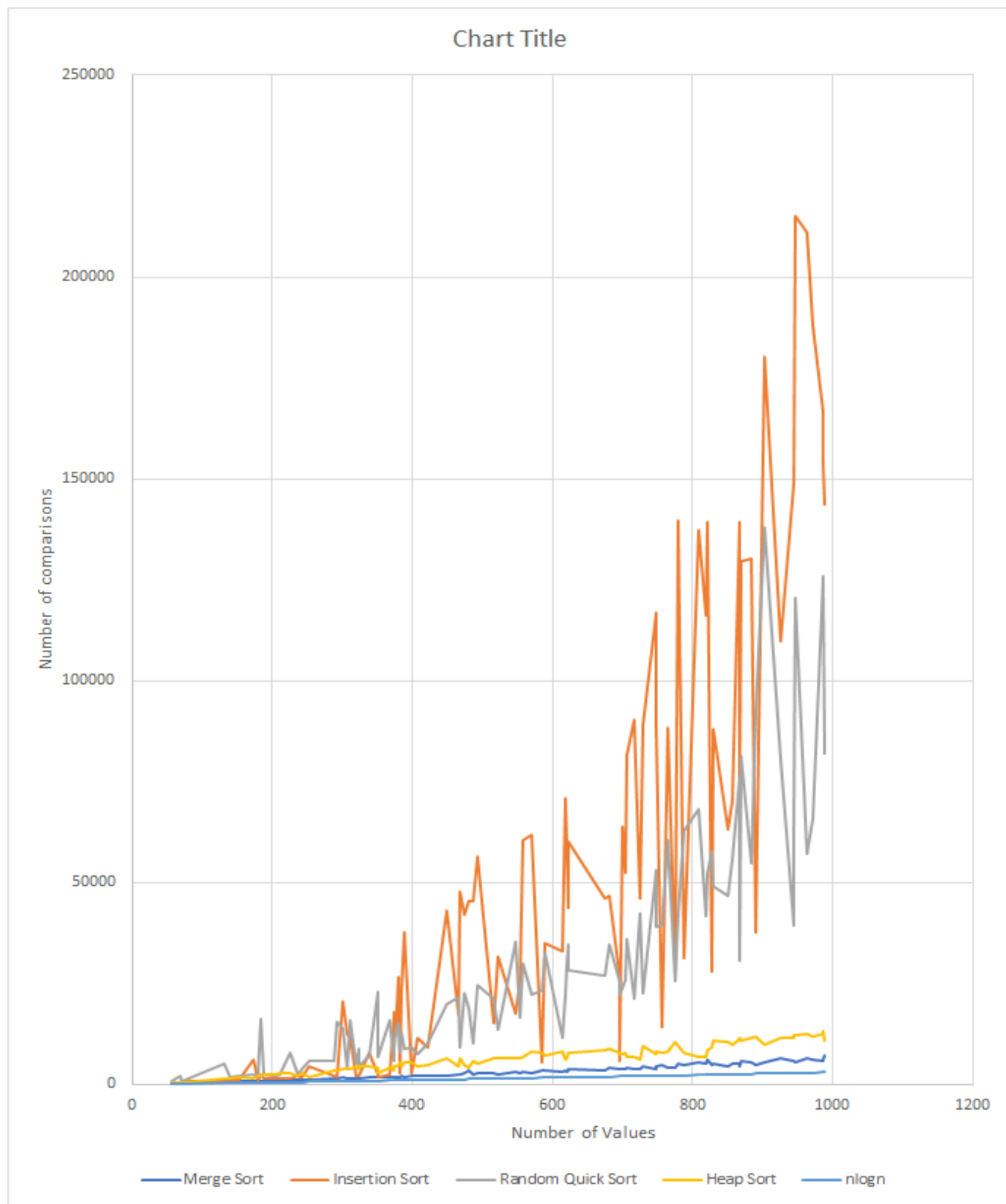
```

    int arrmerge[randomnum];
    int arrins[randomnum];
    int arrquick[randomnum];
    int arrheap[randomnum];
    cout<<setw(10)<<randomnum;
    for(int j= 0;j<randomnum;j++){
        randomval = rand()+1;
        arrmerge[i]=randomval;
        arrins[i]=randomval;
        arrquick[i]=randomval;
        arrheap[i]=randomval;
        // cout<<randomval<<endl;
    }
    mergeSort(arrmerge, 0, randomnum - 1);
    cout<<setw(18)<<compmerge;
    insertionSort(arrins, randomnum);
    cout<<setw(16)<<compins;
    quickSort(arrquick, 0, randomnum - 1);
    cout<<setw(16)<<compquick;
    heapSort(arrheap, randomnum);
    cout<<setw(13)<<compheap<<"\n"; }
    return 0;
}

```

no. of values input	no of comparisons required by			
	merge sort	insertion sort	Rquick sort	heap sort
71	349	1005	1008	486
481	3388	46225	9122	3928
818	5067	116197	122163	10745
610	3497	84225	43893	5124
190	876	1024	7984	2262
878	6029	164490	67118	12309
912	5570	60556	67589	11737
950	5649	61056	71681	11570
573	2967	75456	31084	8325
585	3478	11040	31377	7981
605	3512	19933	32099	7895
372	1823	33860	21433	5202
977	6057	197303	76123	12935
37	129	399	163	263
623	4312	96124	63883	9525
367	1813	29338	5819	4667
674	4082	77277	51138	9024
825	4736	138137	64642	10278
383	1861	1927	5965	5701
778	4987	32464	48591	9981
623	3312	72608	10807	7255
868	5362	145279	86311	12294
212	986	1251	12544	2644
241	1139	1919	6699	2634
825	5362	157821	48169	11144
229	1058	1352	4122	2759
995	6938	218537	90774	12596
636	3272	101345	59393	9869
241	1110	1427	7574	2810
289	1557	18370	13447	3203
856	5439	141376	51940	11699
212	986	1260	21518	2468
150	684	3270	5024	664
860	6268	151683	57288	11920
452	2223	19789	10826	4169
415	2036	8085	11211	4307
584	3434	75322	36323	7569
397	1946	28352	13061	4707
80	354	475	1207	446
637	4710	84217	41715	8364
350	1711	18643	4291	3520
102	446	602	3013	1035
132	645	1216	1916	1258
975	7617	204535	92498	13287
246	1132	1450	7823	2287
145	659	2028	7654	1709
510	2991	56942	20327	6443

510	2991	56942	20327	6443
259	1183	2904	3452	2110
354	2017	23206	21995	4213
952	6475	189625	77318	12666
954	5418	4433	79665	12895
825	4668	65845	31016	10832
802	4486	13461	32516	11580
213	998	1275	1729	1911
263	1392	2190	2789	2400
470	2603	23580	32046	6465
316	1513	20302	8494	2919
695	4474	67703	38402	8869
395	1929	29671	12178	3991
855	5476	104515	52866	11474
882	5424	47841	55910	10875
994	5851	166818	61635	11600
155	708	957	11244	1341
924	6660	173386	65976	10684
908	4986	11235	66206	11618
365	1787	1973	16704	5097
526	2845	40151	13765	6671
475	2322	18321	14933	6772
898	5320	151673	73215	12580
744	3903	78379	23618	9554
340	1665	2067	12631	4831
374	2070	20745	13378	4833
187	863	965	8637	1468
593	3702	61610	26025	7749
375	1834	28268	17230	5343
170	790	1044	2406	1468
551	3331	53256	24376	7079
720	4242	104361	66826	9957
294	1408	1791	8388	4028
432	2365	17881	6574	5355
52	214	289	718	346
723	5766	117056	61417	9943
245	1124	1144	17327	2970
204	954	3993	9580	2344
767	4856	142847	51946	9983
274	1277	1356	20872	3612
574	3569	70837	28751	7064
701	4100	106348	44420	8420
981	5958	150892	136216	13889
216	1008	1342	15453	2750
407	2092	5259	14498	5030
784	4507	94255	54189	10249
560	2819	60324	37627	5736
678	4002	59158	58693	6849
763	4482	78057	58938	7094
600	3077	34966	7072	6084



Comparisons of different sort algorithms

