Ramanujan College

Delhi University

Practical File

Design And Analysis of Algorithms

(Core)

Semester-IV

Submitted to:

Ms. Sheetal Singh

Assistant Professor

Ramanujan College

Submitted by:

Hitesh Harsh

20016570007

BSc(H) Computer Science

1.

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

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

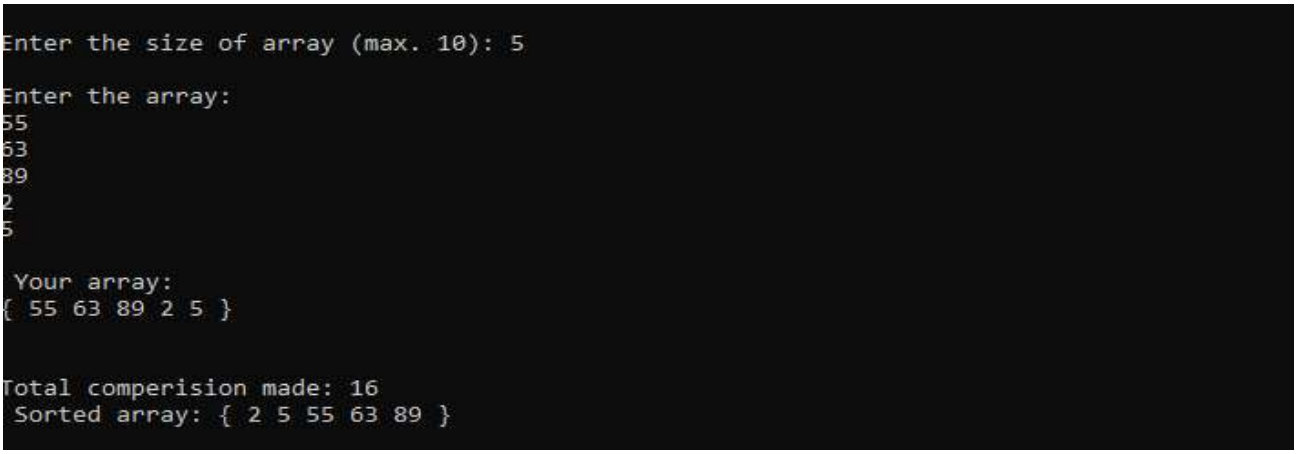
Answer:

i.

```
#include<iostream.h>
#include<conio.h>
using namespace std;
void display(int *a, int size) {
   cout<<"{ ";
   for(int i=0; i<size; i++ )
cout<<a[i]<<' ';
   cout<<"}"<<endl;
}

int insertionSort(int *a, int n)
{
  int i, j, k, comparison = 0;
  for(i=1; i<n; i++)
  {
    comparison++;
    k = a[i];
    for(j=i-1; (a[j]>k) && (j>=0); j--)
    {
```

```
        comparison += 2;
        a[j+1] = a[j];
    }
    a[j+1] = k;
  }
  return comparison;
}
int main()
{
int size, i, *arr;
cout<<"\nEnter the size of array (max. 10): ";
cin>>size;
arr = new int[size];
cout<<"\nEnter the array: \n";
for(i=0; i<size; i++)
   cin>>arr[i];
cout<<"\n Your array: \n";
display(arr, size);
getch();
cout<<"\n\nTotal comperision made: "<<insertionSort(arr, size);
cout<<"\n Sorted array: ";
display(arr, size);
getch();
return 0;
}
```

Output:



ii.

```
pace std;
int comparison = 0;
void display(int *a, int size) {
   cout<<"{ ";
   for(int i=0; i<size; i++ )
cout<<a[i]<<' ';
   cout<<"}"<<endl;
}

void merge(int *a, int beg, int mid, int end)
{
  int size = end - beg + 1;
  int *temp = new int[size];
  int i=beg, j=mid+1, k=0;

  //arranging in order
  while (i <= mid && j<=end){
   temp[k++] = (a[i]<a[j]) ? a[i++] : a[j++];
```

```
   comparison += 3;
  }
  while(i<=mid){
   comparison++;
   temp[k++] = a[i++];
  }
  while(j<end){
   comparison++;
   temp[k++] = a[j++];
  }

  for(i=0; i<k; i++)
  {
   a[i+beg] = temp[i];
  }
}

void mergeSort(int* a,int beg, int end)
{
   if (beg < end) {
comparison++;
       int mid = (beg+end)/2;
       mergeSort(a, beg, mid);
       mergeSort(a, mid+1, end);
       merge(a, beg, mid, end);
   }
}
```

```
int main()
{
int size, i, *arr;
cout<<"\nEnter the size of array (max. 10): ";
cin>>size;
arr = new int[size];
cout<<"\nEnter the array: \n";
for(i=0; i<size; i++)
  cin>>arr[i];
cout<<"\n Your array: \n";
display(arr, size);

getch();
mergeSort(arr, 0, size-1);
cout<<"\n\nTotal comperision made: "<<comparison;

cout<<"\n Sorted array: \n";
display(arr, size);

getch();
}
```

Output:

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

Answer:

```
#include <iostream>
#include <conio.h>
using namespace std;
int comparison = 0;

void display(int *a, int size) {
   cout<<"{ ";
   for(int i=0; i<size; i++ )
cout<<a[i]<<' ';
   cout<<"}"<<endl;
}

void swap(int *a, int x, int y){
  int temp = a[y];
  a[y] = a[x];
  a[x] = temp;
}
```

```
void maxHeapify(int *a, int index, int heapSize)
{
  int left = index*2 + 1;
  int right = index*2 + 2;
  int largest = index;

  if(left < heapSize && a[left] > a[largest]){
    largest = left;
  comparison+=2;
  }
  if(right < heapSize && a[right] > a[largest]){
    largest = right;
  comparison+=2;
  }

  if(largest != index){
    comparison++;
    swap(a, largest, index);
    maxHeapify(a, largest, heapSize);
  }
}

void buildMaxHeap(int *a, int n)
{
  for (int i = (n/2) - 1; i >= 0; i--) {
    maxHeapify(a, i, n);
    comparison++;
  }
```

```
}

void heapSort(int *a, int size)
{
  buildMaxHeap(a, size);
  int heapSize = size, i;
  for(i=size-1; i>=0; i--) {
            swap(a, 0, i);
            heapSize--;
            comparison++;
            maxHeapify(a,0,heapSize);
  }
}

int main()
{
int size, i, *arr;
cout<<"\nEnter the size of array (max. 10): ";
cin>>size;
arr = new int[size];
cout<<"\nEnter the array: \n";
for(i=0; i<size; i++)
   cin>>arr[i];
cout<<"\n Your array: \n";
display(arr, size);
getch();
heapSort(arr, size);
cout<<"\n\nTotal comperision made: "<<comparison;
cout<<"\n Sorted array: \n";
```

```
display(arr, size);

getch();

return 0;

}
```

Output:

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

Answer:

```
#include <conio.h>

#include <iostream>

#include <stdlib.h>

#include <stdio.h>

using namespace std;

int comparison = 0;


void display(int *a, int size) {

   cout<<"{ ";

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

```
cout<<a[i]<<' ';
    cout<<"}"<<endl;
}

void swap(int *a, int x, int y){
  int temp = a[y];
  a[y] = a[x];
  a[x] = temp;
}

int partition(int *a, int p, int r)
{
  int i = p-1, j, x;
  for (j = p; j<r; j++)
    if(a[j] <= a[r]){
      comparison+=2;
      i++;
      swap(a, j, i);
    }
  swap(a, i+1, r);

  return i+1;
}

int randomizedPartition(int *a, int beg, int end)
{
  int t = (rand()%(end-beg)) + beg;
  swap(a, end, t);
  return partition(a, beg, end);
```

```
}

void randomizedQuickSort(int *a, int p, int r)
{
  if (p<r) {
    comparison++;
    int q = randomizedPartition(a, p, r);
    randomizedQuickSort(a, p, q-1);
    randomizedQuickSort(a, q+1, r);
  }
}

int main()
{
int size, i, *arr;
cout<<"\nEnter the size of array (max. 10): ";
cin>>size;
arr = new int[size];

cout<<"\nEnter the array: \n";
for(i=0; i<size; i++)
   cin>>arr[i];
cout<<"\n Your array: \n";
display(arr, size);
getch();
randomizedQuickSort(arr, 0, size-1);
cout<<"\n\nTotal comperision made: "<<comparison;
cout<<"\n Sorted array: \n";
display(arr, size);
```

```
getch();
}
```

Output:

4. Implement Radix Sort

Answer:

```
#include <iostream>
#include <conio.h>
#include <math.h>
using namespace std;
void countSort(int arr[], int size, int num) {
   int x[10];
for(int c=0; c<10; c++)
  x[c] = 0;

   int *disp = new int[size];
   for(int i=0; i<size; i++)
      x[int(arr[i]/num)%10]++;
   for(int i=1; i<10; i++) {
      x[i] += x[i-1];
```

```
    }
    for(int j=0; j<size; j++) {
        disp[x[int(arr[j]/num)%10] - 1] = arr[j];
        x[int(arr[j]/num)%10]--;
    }
    for(int k=0; k<size; k++) {
        arr[k] = disp[k];
    }
}

void radixSort(int arr[], int size) {
    int max = arr[0];

    for(int i=1; i<size; i++)
        if(max<arr[i])
            max = arr[i];

    for(int p=1; max/p>0; p*=10)
        countSort(arr, size, p);
}

int main()
{
int arr[10], size, largest, i;
cout<<"\nEnter the size of array (max. 10): ";
cin>>size;
cout<<"\nEnter positive elements in the array: \n";
for(i=0; i<size; i++)
cin>>arr[i];
```

```
cout<<"\n Your array: \n";
for(i=0; i<size; i++)
{
cout<<arr[i]<<" ";
}

getch();
radixSort(arr, size);
cout<<"\n Sorted array: ";
for(i=0; i<size; i++)
{
cout<<arr[i]<<" ";
}
getch();
}
```

Output:

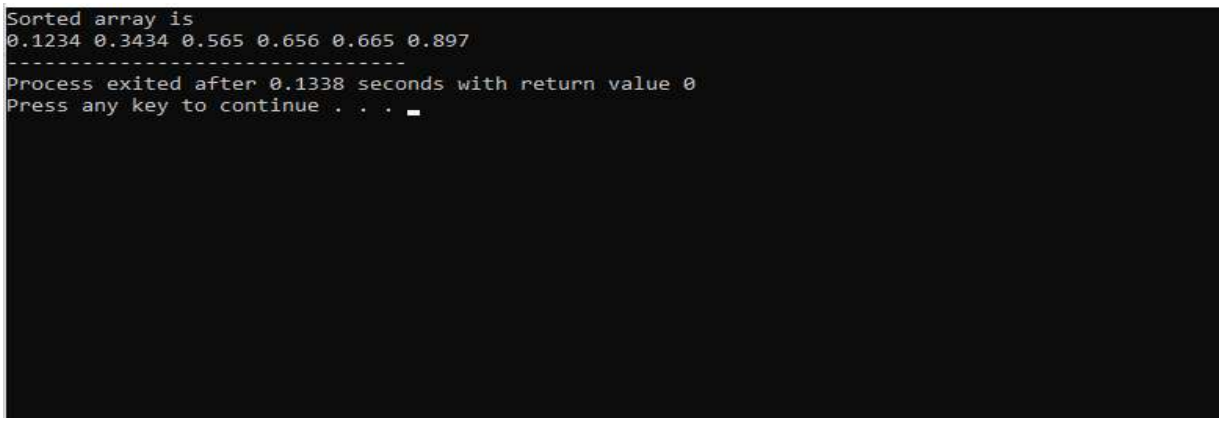
5. Implement Bucket Sort

Answer:

```
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;

// Function to sort arr[] of
// size n using bucket sort
void bucketSort(float arr[], int n)
{

    // 1) Create n empty buckets
    vector<float> b[n];

    // 2) Put array elements
    // in different buckets
    for (int i = 0; i < n; i++) {
        int bi = n * arr[i]; // Index in bucket
        b[bi].push_back(arr[i]);
    }

    // 3) Sort individual buckets
    for (int i = 0; i < n; i++)
        sort(b[i].begin(), b[i].end());

    // 4) Concatenate all buckets into arr[]
    int index = 0;
```

```
    for (int i = 0; i < n; i++)
        for (int j = 0; j < b[i].size(); j++)
            arr[index++] = b[i][j];
}


int main()
{
    float arr[]
        = { 0.897, 0.565, 0.656, 0.1234, 0.665, 0.3434 };
    int n = sizeof(arr) / sizeof(arr[0]);
    bucketSort(arr, n);

    cout << "Sorted array is \n";
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    return 0;
}
```

Output:



6. Implement Randomized Select

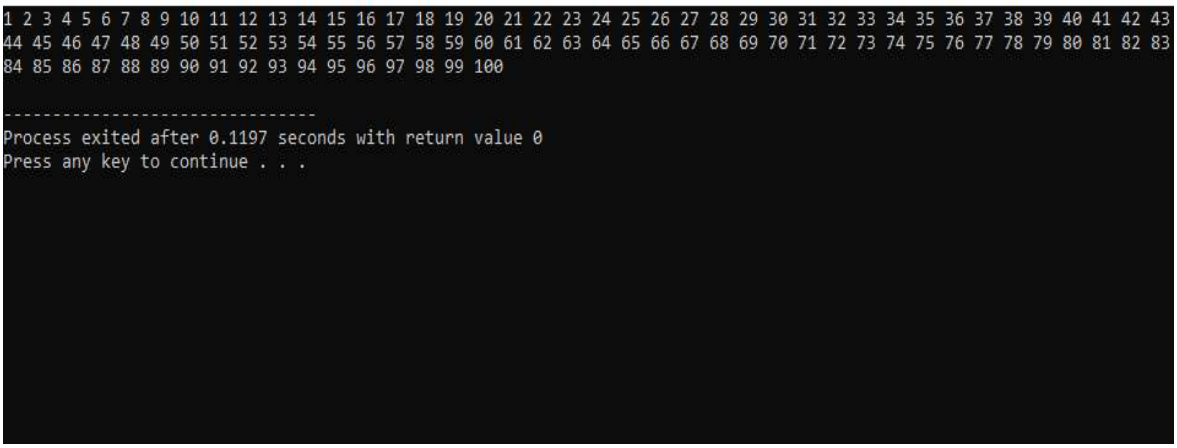
Answer:

```
#include<iostream>
#include<cstdlib>
#include<ctime>
#define MAX 100
using namespace std;
void random_shuffle(int arr[]) {
  //function to shuffle the array elements into random positions
  srand(time(NULL));
  for (int i = MAX - 1; i > 0; i--) {
    int j = rand()%(i + 1);
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
  }
}
// Partitioning the array on the basis of values at high as pivot value.
int Partition(int a[], int low, int high) {
  int pivot, index, i;
  index = low;
  pivot = high;
  for(i=low; i < high; i++) {
    // finding index of pivot.
    if(a[i] < a[pivot]) {
      swap(a[i], a[index]);
      index++;
```

```
    }
  }
  swap(a[pivot], a[index]);
  return index;
}
int RandomPivotPartition(int a[], int low, int high){
  // Random selection of pivot.
  int pvt, n, temp;
  n = rand();
  pvt = low + n%(high-low+1); // Randomizing the pivot value from sub-array.
  swap(a[high], a[pvt]);
  return Partition(a, low, high);
}
void quick_sort(int arr[], int p, int q) {
  //recursively sort the list
  int pindex;
  if(p < q) {
    pindex = RandomPivotPartition(arr, p, q); //randomly choose pivot
    // Recursively implementing QuickSort.
    quick_sort(arr, p, pindex-1);
   quick_sort(arr, pindex+1, q);
}
}
int main() {
int i;
int arr[MAX];
for (i = 0;i < MAX; i++)
arr[i] = i + 1;
random_shuffle(arr); //To randomize the array
```

```
quick_sort(arr, 0, MAX - 1); //sort the elements of array

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

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

cout << endl;

return 0;

}
```

Output:



7. Implement Breadth-First Search in a graph

Answer:

```
#include<iostream>

using namespace std;

#include<conio.h>

Using namespace std;


int graph[20][2], rows=-1, index = 0;



////////////////////////////////

/////     QUEUE           /////
```

```
//////////////////////////////////

class node
{
  public:
int info;
node *next;

    node(int data, node *ptr = 0)
{
  info = data;
  next = ptr;
}
};

class queue
{
node *head, *tail;

  public:
  queue()
 {
    head = 0;
    tail = 0;
 }

  int giveHead()
  {
    return head->info;
```

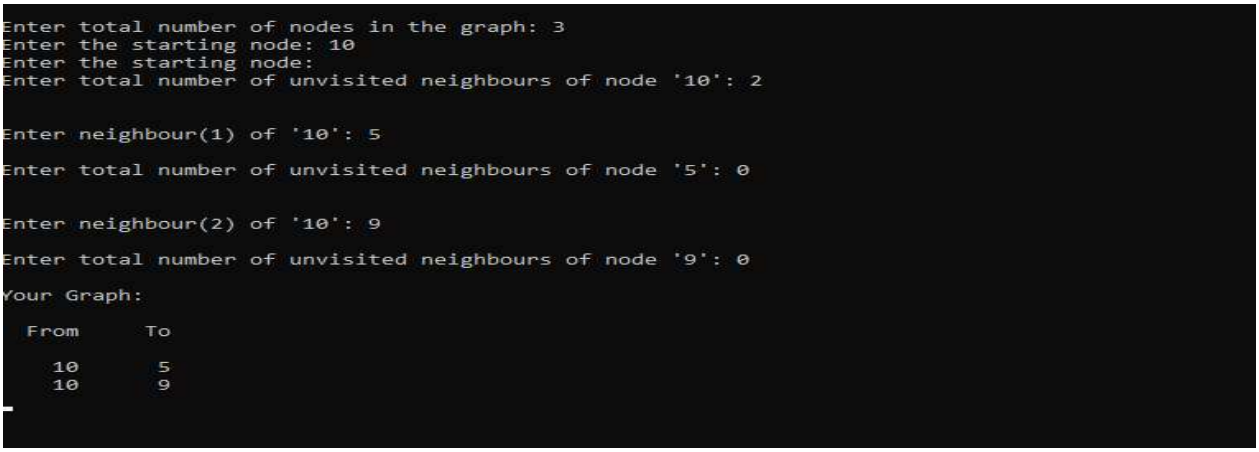
```
  }
void enqueue(int val)
  {
    if(head != 0)
    {
      tail->next = new node(val);
      tail = tail->next;
    }
    else
      head = tail = new node(val);
  }

int dequeue()
  {
    int temp;
    if(head != 0)
    {
       temp = head->info;
     head = head->next;
       return temp;
    }
    else
     return -1;
  }
} Q;

////////////////////////////////

void printGraph()
```

```
{


cout<<"\nYour Graph: ";
cout<<"\n\n  From    To\n\n";
  for(int i = 0; i < rows; i++) {
    for (int j = 0; j < 2; j++) {
      cout<<"   "<<graph[i][j]<<"\t";
    }
    cout<<endl;
  }
}

int checkPeresence(int a, int row, int col)
{
   for(int i=0; i<row; i++)
     if(graph[i][col] == a)
       return 1;
   return 0;
}

void drawGraph(int x, int i,  int limit)
{
    int n;

    cout<<"\nEnter total number of unvisited neighbours of node \'"<<x<<"\': ";
    cin>>n;
    rows += n;

    if(n>0)
```

```
    {
      for (int j = i; j < i+n; j++) {
       graph[j][0] = x;
      }

      for (int j = i; j < i+n; j++) {
cout<<"\n\nEnter neighbour("<<j-i+1<<") of \'"<<x<<"\': ";
cin>>graph[j][1];

if(checkPeresence(graph[j][1], rows, 0) == 0)
  drawGraph(graph[j][1], rows, limit);
      }
    }
}

////////////////////////////////////////////////////////////
void fill(int x)
{
  cout<<x<<"  ";
  Q.enqueue(x);
}

void BFS()
{
  while ((index < rows) && (Q.giveHead() == graph[index][0])) {
    if(checkPeresence(graph[index][1], index, 1) == 0)
      fill(graph[index][1]);
    index++;
  }
```

```
  if(index < rows)
  {
    do {
        if (Q.dequeue() == -1)
        return;
    } while(Q.giveHead() != graph[index][0]);

    BFS();
  }
  return;
}

int main()
{

    int t, start;
    rows = 0;

    cout<<"\nEnter total number of nodes in the graph: ";
    cin>>t;
    cout<<"Enter the starting node: ";
    cin>>start;


    cout<<"Enter the starting node: ";
    drawGraph(start, 0, t-1);
    printGraph();

    getch();
```

```
  cout<<"------------------\n\n"
   <<"Output of Breadth First Search (BFS) for this graph:"
   <<"\n\n";
  fill(start);
  BFS();

 getch();
}
```



8. Implement Depth-First Search in a graph

Answer:

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

int graph[20][3], rows=-1;
int vFlag[20][2], tNodes, index = 0;

////////////////////////////////
/////  STACK          /////
////////////////////////////////
```

```
class node
{
  public:
int info;
node *next;

  node(int data, node *ptr = 0)
{
  info = data;
  next = ptr;
}
};

class stack
{
node *top;

  public:
  stack()
{ top = 0; }

  int givetop()
  {  return top->info;  }

  void push(int val)
  {
    if(top != 0)
      top = new node(val, top);
```

```
    else
      top = new node(val);
  }

  void pop()
  {
    if(top != 0)
    top = top->next;
    else
    return;
  }

};

////////////////////////////////

int getFirstIndex(int n)
{
  for (int c = 0; c < rows; c++)
    if(graph[c][0] == n)
      return c;
  return -1;
}

void printGraph()
{

cout<<"\nYour Graph: ";
cout<<"\n\n  From    To\n\n";
```

```
  for(int i = 0; i < rows; i++) {
    for (int j = 0; j < 2; j++) {
      cout<<"    "<<graph[i][j]<<"\t";
    }
    cout<<endl;
  }
}

int checkPeresence(int a, int row, int col)
{
   for(int i=0; i<row; i++)
     if(graph[i][col] == a)
       return 1;
   return 0;
}

void drawGraph(int x, int i,  int limit)
{
    int n;

    cout<<"\nEnter total number of unvisited neighbours of node \'"<<x<<"\': ";
    cin>>n;
    rows += n;

    if(n>0)
    {
      for (int j = i; j < i+n; j++) {
       graph[j][0] = x;
      }
```

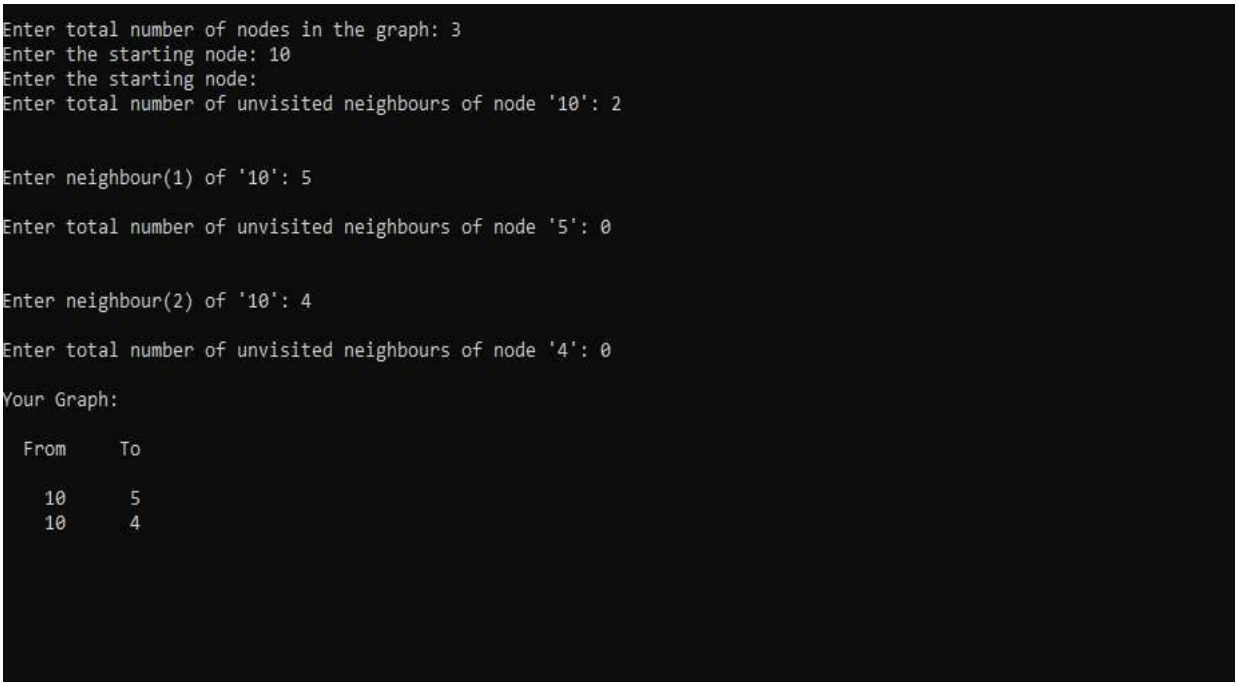
```
    for (int j = i; j < i+n; j++) {
            cout<<"\n\nEnter neighbour("<<j-i+1<<") of \'"<<x<<"\': ";
            cin>>graph[j][1];

      for(int c=0; (c<index) && (graph[j][1] != vFlag[c][0]); c++);
            if(int c = index)
             vFlag[index++][0] = graph[j][1];

            if(checkPeresence(graph[j][1], rows, 0) == 0)
             drawGraph(graph[j][1], rows, limit);
    }
   }
}

////////////////////////////////////////////////////////
void markVisited(int val) {
  for(int i=0; i<tNodes; i++)
   if(vFlag[i][0] == val)
   {
    vFlag[i][1] = 1;
    return;
   }
}

int isVisited(int x){
  for (int i = 0; i < tNodes; i++)
   if (vFlag[i][0] == x){
    return (vFlag[i][1] == 1) ? 0 : -1;
```

```
    }
  return -1;
}

void fill(int x)
{
  cout<<x<<"  ";
  markVisited(x);
  S.push(x);
}

void DFS(int v, int i)
{
  int j, temp;
  if(isVisited(v == -1))
    fill(v);

  while((S.givetop() == graph[i][0])&&(i<rows)){
    if(isVisited(graph[i][1]) == -1){
      j = getFirstIndex(graph[i][1]);
      if(j != -1){
      DFS(graph[i][1], j);
      }
      else{
      fill(graph[i][1]);
      S.pop();
      }
    }
    i++;
```

```
  }
  S.pop();
}

int main()
{

    int start, t=0;
    rows = 0;

    cout<<"\nEnter total number of nodes in the graph: ";
    cin>>tNodes;
    cout<<"Enter the starting node: ";
    cin>>start;
    vFlag[index++][0] = start;



    cout<<"Enter the starting node: ";
    drawGraph(start, 0, tNodes-1);
    printGraph();

    for (int c = 0; c < tNodes; c++) {
     vFlag[c][1] = 0;
    }

    getch();

    cout<<"------------------\n\n"
      <<"Output of Depth First Search (DFS) for this graph:"
```

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


   DFS(start, 0);


  getch();
}
```

Output:



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

Answer:

```
#include <iostream>

#include <climits>

using namespace std;


#define v 5
```

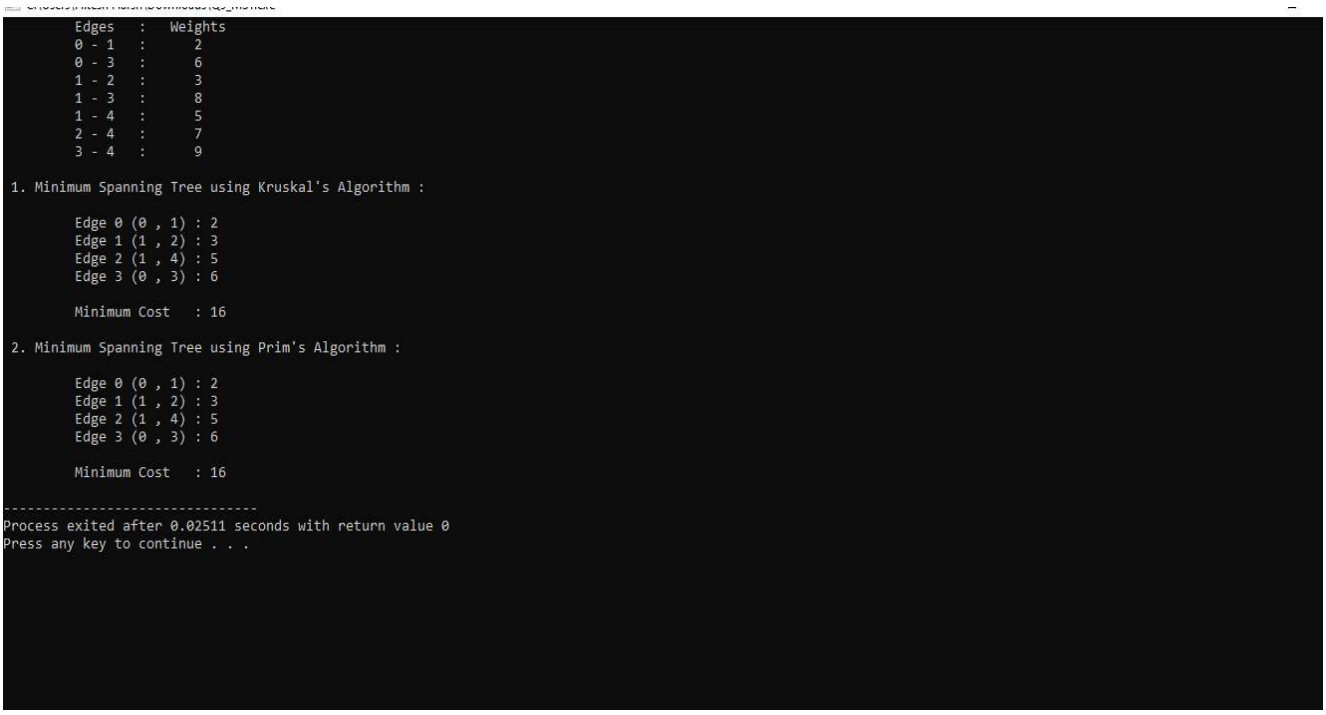
```
int parent[v];

bool visited[v] = {0};

/******      Kruskal's Algorithm       *******/

// Find set of vertex i
int find(int i)
{
   while (parent[i] != i)
      i = parent[i];
   return i;
}

void union1(int i, int j)
{
   int a = find(i);
   int b = find(j);
   parent[a] = b;
}

int kruskal_mst(int cost[][v])
{
   int mincost = 0;
   int edge_count = 0;

   for (int i = 0; i < v; i++)
      parent[i] = i;
```

```
   while (edge_count < v - 1)
   {
      int min = INT_MAX, a = -1, b = -1;
      for (int i = 0; i < v; i++)
         for (int j = 0; j < v; j++)
            if (find(i) != find(j) &&
               cost[i][j] < min)
            {
               min = cost[i][j];
               a = i;
               b = j;
            }

      union1(a, b);
      mincost += min;
      cout << "\n\t Edge " << edge_count++ << " ("
         << a << " , " << b << ") : " << min;
   }
   return mincost;
}

/******      Prim's Algorithm      *******/

bool validEdge(int a, int b)
{
   if (a == b || visited[a] == visited[b])
      return false;
   return true;
}
```

```
int prim_mst(int cost[][v])
{
   int mincost = 0, edge_count = 0;
   fill(visited, visited + v, false);
   visited[0] = true;

   while (edge_count < v - 1)
   {
      int min = INT_MAX, a = -1, b = -1;
      for (int i = 0; i < v; i++)
         for (int j = 0; j < v; j++)
            if (cost[i][j] < min &&
               validEdge(i, j))
            {
               min = cost[i][j];
               a = i;
               b = j;
            }

      visited[a] = visited[b] = true;
      mincost += min;
      cout << "\n\t Edge " << edge_count++ << " ("
         << a << " , " << b << ") : " << min;
   }
   return mincost;
}

/*************************************/
```

```
int main()
{
   int Graph[][v] = {
      {INT_MAX, 2, INT_MAX, 6, INT_MAX},
      {2, INT_MAX, 3, 8, 5},
      {INT_MAX, 3, INT_MAX, INT_MAX, 7},
      {6, 8, INT_MAX, INT_MAX, 9},
      {INT_MAX, 5, 7, 9, INT_MAX},
   };

   cout << "\n\t\t  Practical 9\n\tMinimum Spanning Tree Algorithms\n\t\t Kruskal & Prim\n";

   cout << "\n\n Given Graph is :\n\n\t Edges   :   Weights\n";
   for (int i = 0; i < v; i++)
      for (int j = 0; j < v; j++)
         if (Graph[i][j] != INT_MAX && !(visited[j]))
         {
            visited[i] = true;
            cout << "\t " << i << " - " << j << "   :\t" << Graph[i][j] << endl;
         }

   cout << "\n 1. Minimum Spanning Tree using Kruskal's Algorithm : \n";
   int k = kruskal_mst(Graph);
   cout << "\n\n\t Minimum Cost\t: " << k << endl;

   cout << "\n 2. Minimum Spanning Tree using Prim's Algorithm : \n";
   k = prim_mst(Graph);
   cout << "\n\n\t Minimum Cost\t: " << k << endl;
```

Output:



10. Write a program to solve the weighted interval scheduling problem.

Answer:

```
#include <iostream>

#include <algorithm>


using namespace std;


struct job

{

   int start, finish, profit;


   job(const int &s = 0, const int &f = 0, const int &p = 0)

   {
```

```
        start = s;
        finish = f;
        profit = p;
    }

    friend ostream &operator<<(ostream &out, const job &j)
    {
        out << '(' << j.start << ',' << j.finish << ',' << j.profit << ')' << endl;
        return out;
    }
};

bool cmp(const job &a, const job &b)
{
    return a.finish < b.finish;
}

int check_Overloop(const job *j, const int &n)
{
    for (int i = n - 2; i > -1; i--)
        if (j[n - 1].start >= j[i].finish)
            return i;
    return -1;
}

int wi_sch(const job *j, const int &n)
{
    if (n == 0)
        return j[0].profit;
```

```
    else
    {
        int i = check_Overloop(j, n);
        int incl = j[n - 1].profit;

        if (i != -1)
            incl += wi_sch(j, i + 1);

        int excl = wi_sch(j, n - 1);
        return (incl > excl ? incl : excl);
    }
}

int max_profit(job *j, const int &n)
{
    sort(j, j + n, cmp);
    cout << "\n\nSorted jobs according to respective finish time are :\n\n"
         << "S.N.\t  Start-time\t  Finish-time\t\tProfit\n";
    for (int i = 0; i < n; i++)
        cout << "\n " << i + 1 << "\t\t" << j[i].start << "\t\t" << j[i].finish << "\t\t" << j[i].profit;
    return wi_sch(j, n);
}

int main()
{
    job j[] = {{3, 10, 20}, {1, 24, 50}, {6, 19, 100}, {2, 100, 20}};
    int n = sizeof(j) / sizeof(j[0]);
```

```
    cout << "\n\t\t Practical 10 \n\t Weighted Interval Schedueling \n";


    cout << "\nGiven jobs are :\n\nS.N.\t  Start-time\t  Finish-time\t\tProfit\n";
    for (int i = 0; i < n; i++)
        cout << "\n " << i + 1 << "\t\t" << j[i].start << "\t\t" << j[i].finish << "\t\t" << j[i].profit;


    int mx = max_profit(j, n);
    cout << "\n\nThe Maximum Optimal Profit is : " << mx << endl;
}
```

Output:

```
                Practical 10
         Weighted Interval Schedueling

Given jobs are :

S.N.       Start-time      Finish-time          Profit

 1              3               10              20
 2              1               24              50
 3              6               19              100
 4              2               100             20

Sorted jobs according to respective finish time are :

S.N.       Start-time      Finish-time          Profit

 1              3               10              20
 2              6               19              100
 3              1               24              50
 4              2               100             20

The Maximum Optimal Profit is : 100

--------------------------------
Process exited after 0.1241 seconds with return value 0
Press any key to continue . . .
```

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

Answer:

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

using namespace std;


// A utility function that returns

// maximum of two integers
```

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

// Returns the maximum value that
// can be put in a knapsack of capacity W
int knapSack(int W, int wt[], int val[], int n)
{

    // Base Case
    if (n == 0 || W == 0)
        return 0;

    // If weight of the nth item is more
    // than Knapsack capacity W, then
    // this item cannot be included
    // in the optimal solution
    if (wt[n - 1] > W)
        return knapSack(W, wt, val, n - 1);

    // Return the maximum of two cases:
    // (1) nth item included
    // (2) not included
    else
        return max(
            val[n - 1]
                + knapSack(W - wt[n - 1],
                           wt, val, n - 1),
            knapSack(W, wt, val, n - 1));
}
```

```
// Driver code
int main()
{
    int val[] = { 60, 100, 120 };
    int wt[] = { 10, 20, 30 };
    int W = 50;
    int n = sizeof(val) / sizeof(val[0]);
    cout << knapSack(W, wt, val, n);
    return 0;
}
```

Output: