Index

S. No	**Name of Programs**	**Date of Experiment**	**Date of submission**	**Signature of Faculty with date**	**Remarks**
1.	Program for Recursive Binary & Linear Search.				
2.	Program for Heap Sort				
3.	Program for Merge Sort				
4.	Program for Selection Sort				
5.	Program for Insertion Sort				
6.	Program for Quick Sort				
7.	Program of Knapsack Problem using Greedy Solution				
8.	Perform Travelling Salesman Problem				
9.	Find Minimum Spanning Tree using Kruskal’s Algorithm				
10.	Implement N Queen Problem using Backtracking				
11.					
12.					
13.					
14.					
15.					

Design and Analysis of Algorithm Lab

(KCS- 553)

Submitted To:

Ms. Rajani Singh
Assistant Professor
IT Department

Submitted By:

Name :
Roll no :
Class : 3rd Year, IT-C

Department of Information Technology

G. L. Bajaj Institute of Technology and Management

Greater Noida, 201306

PROGRAM NO. :- 01

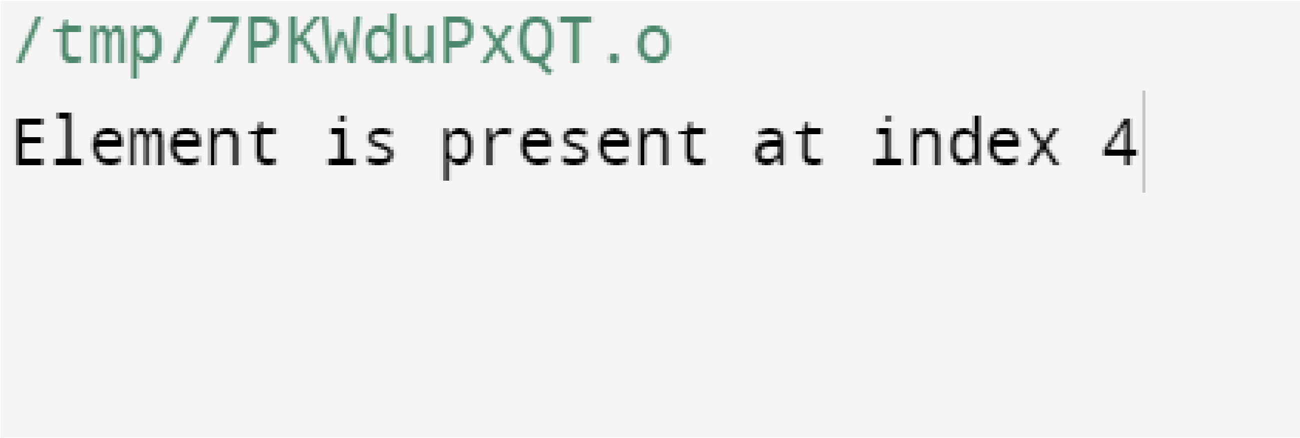
OBJECTIVE: Program for Recursive Binary & Linear Search

(I) Recursive Binary Search

CODE :-

```
#include<stdio.h>

int binarySearch(int arr[], int l, int r, int x)
{
   if (r >= l)
   {
      int mid = l + (r - l)/2;
      if (arr[mid] == x) return mid;
      if (arr[mid] > x) return binarySearch(arr, l, mid-1, x);
      return binarySearch(arr, mid+1, r, x);
   }
   return -1;
}
int main(void)
{
   int arr[] = {1, 4, 3, 65, 80};
   int n = sizeof(arr)/ sizeof(arr[0]);
   int x = 80; int result = binarySearch(arr, 0, n-1, x);
   (result == -1)?
   printf("Element is not present in array"): printf("Element is present at
        index %d", result);
   return 0;
}
```

OUTPUT :-



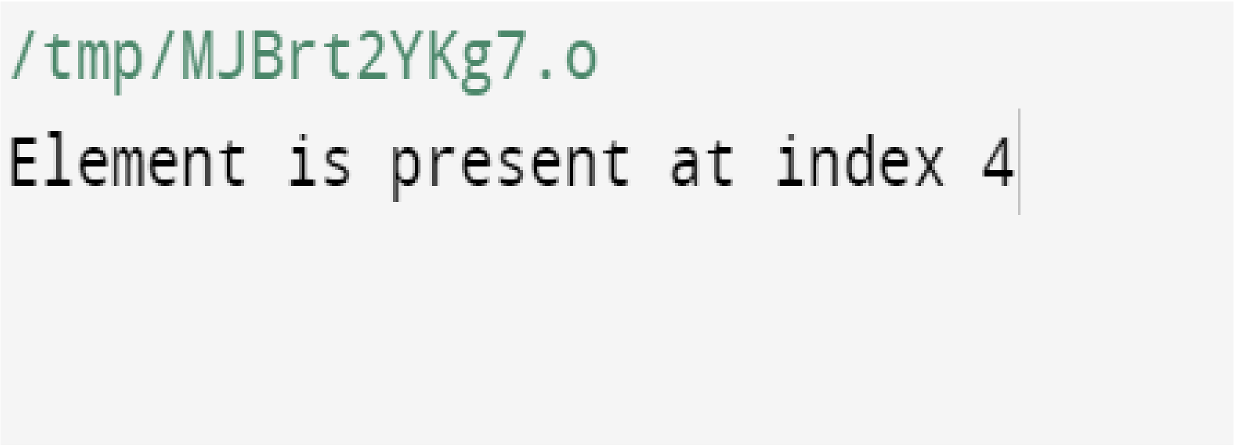
```
/tmp/7PKWduPxQT.o
Element is present at index 4
```

(II) Linear Search

CODE :-

```
#include <iostream>
using namespace std;
int search(int arr[], int N, int x)
{
   int i;
   for (i = 0; i < N; i++)
   if (arr[i] == x)
   return i;

   return -1;
}

int main(void)
{
   int arr[] = {1, 4, 3, 65, 80};
   int x = 80;
   int N = sizeof(arr) / sizeof(arr[0]);
   int result = search(arr, N, x);
   (result == -1)?
   cout << "Element is not present in array":
   cout << "Element is present at index "<< result;
   return 0;

}
```

OUTPUT :-



```
/tmp/MJBrt2YKg7.o
Element is present at index 4
```

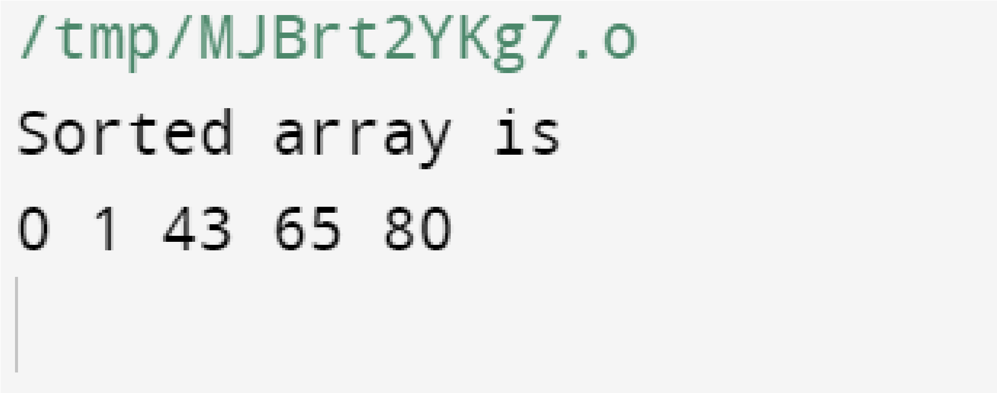
PROGRAM NO. :- 02

OBJECTIVE: Program for Heap Sort

CODE :-

```
#include <iostream>
using namespace std;
void heapify(int arr[], int N, int i)
{
    int largest = i;
    int l = 2 * i + 1;
    int r = 2 * i + 2;
    if (l < N && arr[l] > arr[largest])
        largest = l;
    if (r < N && arr[r] > arr[largest])
        largest = r;
    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 arr[] = { 0, 43, 65, 1, 80 };

    int N = sizeof(arr) / sizeof(arr[0]);
    heapSort(arr, N);
    cout << "Sorted array is \n";
    printArray(arr, N);
}
```

OUTPUT :-



```
/tmp/MJBrt2YKg7.o
Sorted array is
0 1 43 65 80
```

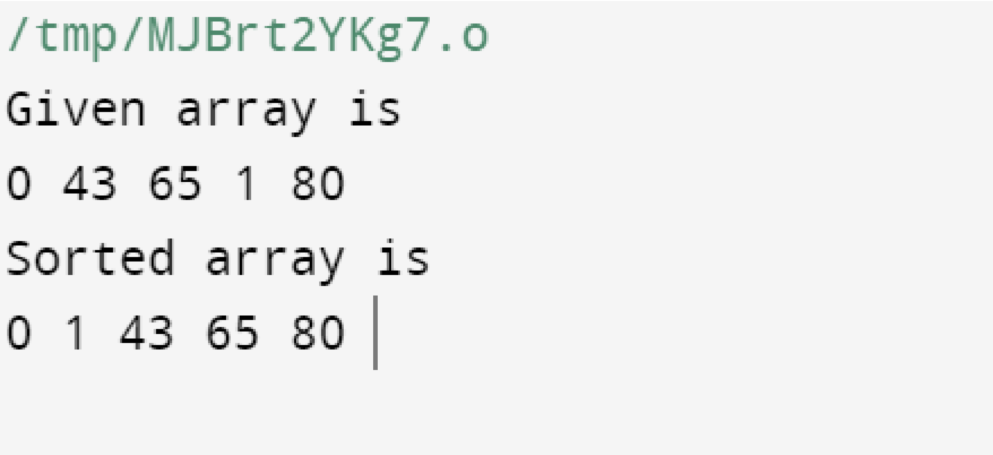
PROGRAM NO. :- 03

OBJECTIVE: Program for Merge Sort

CODE :-

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

void merge(int array[], int const left, int const mid, int const right) {
    auto const subArrayOne = mid - left + 1;
    auto const subArrayTwo = right - mid;
    auto *leftArray = new int[subArrayOne],
         *rightArray = new int[subArrayTwo];
    for (auto i = 0; i < subArrayOne; i++)
        leftArray[i] = array[left + i];

    for (auto j = 0; j < subArrayTwo; j++)
        rightArray[j] = array[mid + 1 + j];

    auto indexOfSubArrayOne = 0, indexOfSubArrayTwo = 0;
    int indexOfMergedArray = left;

    while (indexOfSubArrayOne < subArrayOne && indexOfSubArrayTwo < subArrayTwo)
{
        if (leftArray[indexOfSubArrayOne] <= rightArray[indexOfSubArrayTwo]) {
            array[indexOfMergedArray] = leftArray[indexOfSubArrayOne];
            indexOfSubArrayOne++;
        } else {
            array[indexOfMergedArray] = rightArray[indexOfSubArrayTwo];
            indexOfSubArrayTwo++;
        }
        indexOfMergedArray++;
    }

    while (indexOfSubArrayOne < subArrayOne) {
        array[indexOfMergedArray] = leftArray[indexOfSubArrayOne];
        indexOfSubArrayOne++;
        indexOfMergedArray++;
    }

    while (indexOfSubArrayTwo < subArrayTwo) {
        array[indexOfMergedArray] = rightArray[indexOfSubArrayTwo];
        indexOfSubArrayTwo++;
        indexOfMergedArray++;
    }

    delete[] leftArray;
    delete[] rightArray;
}

void mergeSort(int array[], int const begin, int const end) {
    if (begin >= end)
        return;

    auto mid = begin + (end - begin) / 2;
    mergeSort(array, begin, mid);
    mergeSort(array, mid + 1, end);
    merge(array, begin, mid, end);
}

void printArray(int A[], int size) {
    for (auto i = 0; i < size; i++)
        cout << A[i] << " ";
}

int main() {
    int arr[] = { 0, 43, 65, 1, 80 };
    auto arr_size = sizeof(arr) / sizeof(arr[0]);

    cout << "Given array is \n";
    printArray(arr, arr_size);

    mergeSort(arr, 0, arr_size - 1);

    cout << "\nSorted array is \n";
    printArray(arr, arr_size);

    return 0;
}
```

OUTPUT :-



```
/tmp/MJBrt2YKg7.o
Given array is
0 43 65 1 80
Sorted array is
0 1 43 65 80
```

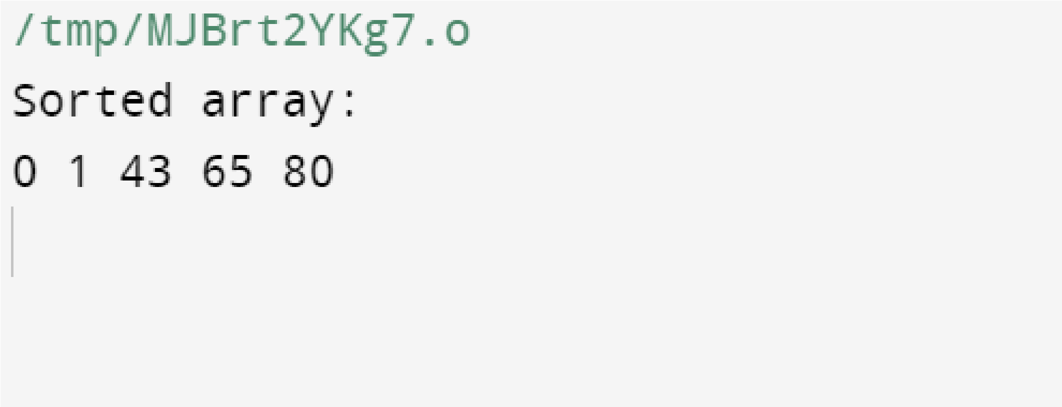
PROGRAM NO. :- 04

OBJECTIVE: Program for Selection Sort

CODE :-

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

void swap(int *xp, int *yp) {
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

void selectionSort(int arr[], int n) {
    int i, j, min_idx;
    for (i = 0; i < n - 1; i++) {
        min_idx = i;
        for (j = i + 1; j < n; j++)
            if (arr[j] < arr[min_idx])
                min_idx = j;

        if (min_idx != i)
            swap(&arr[min_idx], &arr[i]);
    }
}

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

int main() {
    int arr[] = { 0, 43, 65, 1, 80 };
    int n = sizeof(arr) / sizeof(arr[0]);
    selectionSort(arr, n);
    cout << "Sorted array: \n";
    printArray(arr, n);
    return 0;
}
```

OUTPUT :-



```
/tmp/MJBrt2YKg7.o
Sorted array:
0 1 43 65 80
```

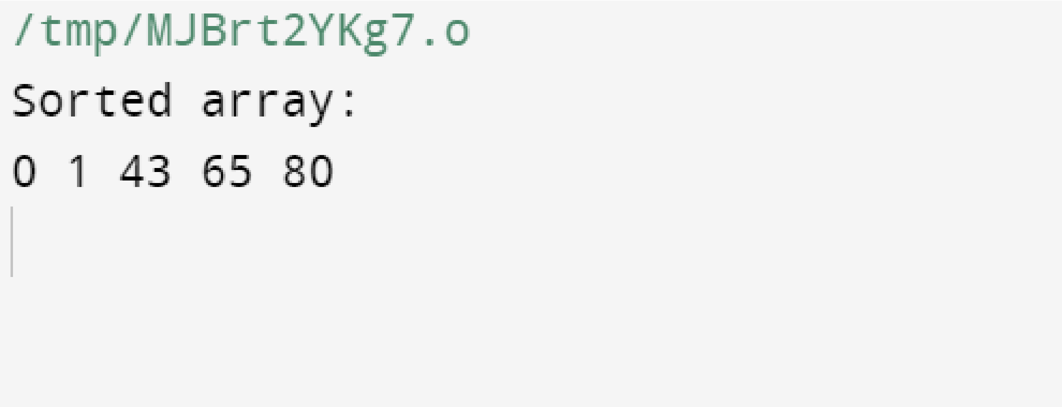
PROGRAM NO. :- 05

OBJECTIVE: Program for Insertion Sort

CODE :-

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

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++)
        cout << arr[i] << " ";
    cout << endl;
}

int main() {
    int arr[] = { 0, 43, 65, 1, 80 };

    int N = sizeof(arr) / sizeof(arr[0]);

    insertionSort(arr, N);
    cout << "Sorted array: \n";
    printArray(arr, N);

    return 0;
}
```

OUTPUT :-



```
/tmp/MJBrt2YKg7.o
Sorted array:
0 1 43 65 80
```

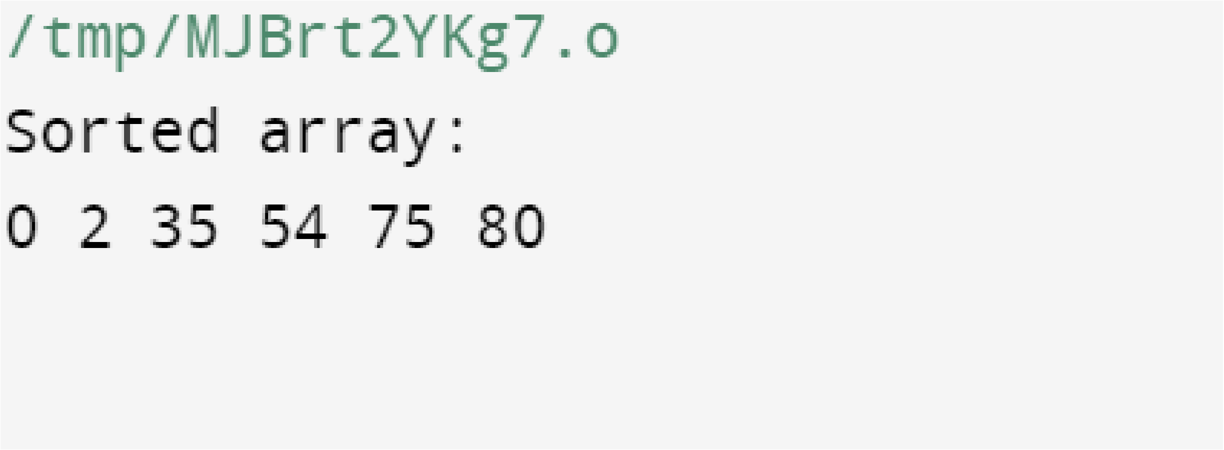
PROGRAM NO. :- 06

OBJECTIVE: Program for Quick Sort

CODE :-

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

void swap(int* a, int* b) {
    int t = *a;
    *a = *b;
    *b = t;
}

int partition(int arr[], int low, int high) {
    int pivot = arr[high]; // pivot
    int i = (low - 1);

    for (int j = low; j <= high - 1; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}

void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

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

int main() {
    int arr[] = { 0, 35, 75, 2, 80, 54};
    int n = sizeof(arr) / sizeof(arr[0]);
    quickSort(arr, 0, n - 1);

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

    return 0;
}
```

<u>OUTPUT :-</u>



```
/tmp/MJBrt2YKg7.o
Sorted array:
0 2 35 54 75 80
```

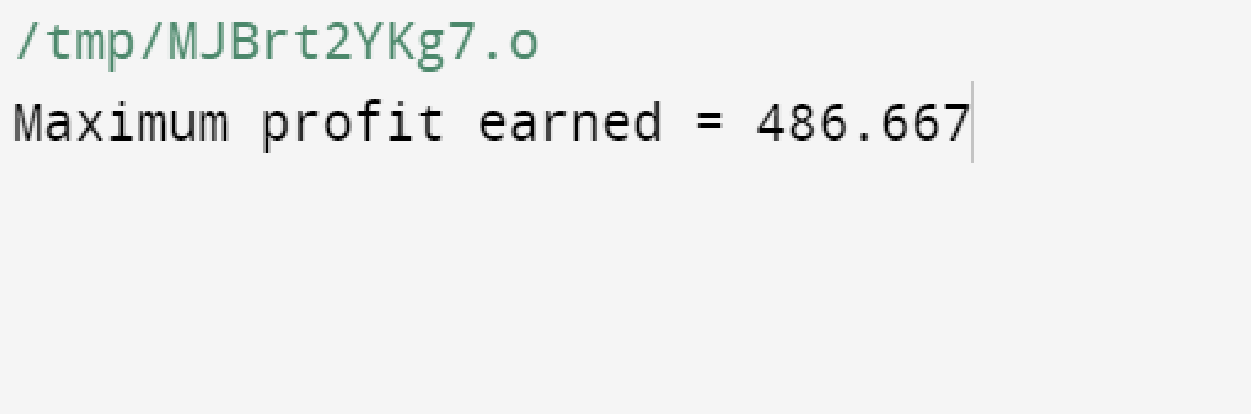
PROGRAM NO. :- 07

OBJECTIVE: Program of Knapsack Problem using Greedy Solution

CODE :-

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

struct Item {
    int value, weight;

    Item(int value, int weight) : value(value), weight(weight) {}
};

bool cmp(struct Item a, struct Item b) {
    double r1 = (double)a.value / a.weight;
    double r2 = (double)b.value / b.weight;
    return r1 > r2;
}
// Main greedy function to solve the problem
double fractionalKnapsack(struct Item arr[], int N, int size) {
    sort(arr, arr + size, cmp);
    int curWeight = 0;
    double finalvalue = 0.0;

    for (int i = 0; i < size; i++) {
        if (curWeight + arr[i].weight <= N) {
            curWeight += arr[i].weight;
            finalvalue += arr[i].value;
        } else {
            int remain = N - curWeight;
            finalvalue += arr[i].value * ((double)remain / arr[i].weight);
            break;
        }
    }
    return finalvalue;
}

int main() {
    int N = 60;
    Item arr[] = {{200, 20}, {180, 30}, {220, 30}, {320, 48}};
    int size = sizeof(arr) / sizeof(arr[0]);

    cout << "Maximum profit earned = " << fractionalKnapsack(arr, N, size);

    return 0;
}
```

OUTPUT :-



```
/tmp/MJBrt2YKg7.o
Maximum profit earned = 486.667
```

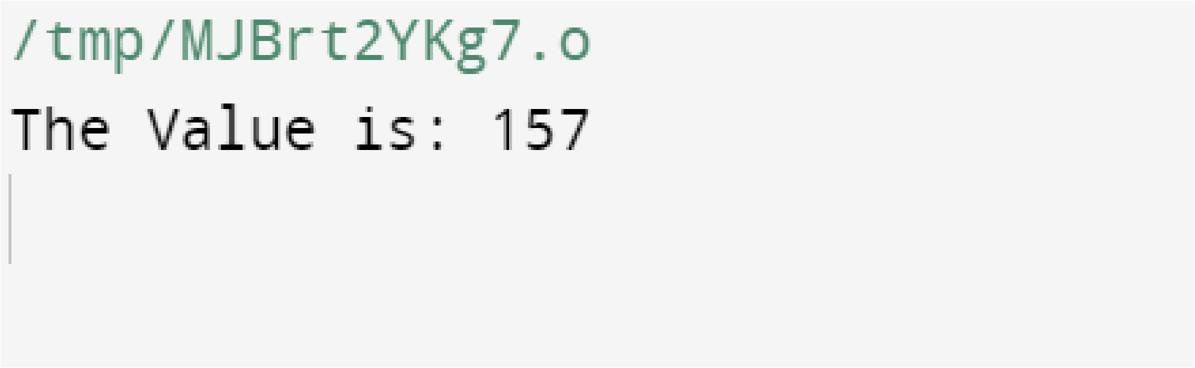
PROGRAM NO. :- 08

OBJECTIVE: Perform Travelling Salesman Problem

CODE :-

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

#define V 4
int travllingSalesmanProblem(int graph[][V], int s) {
    vector<int> vertex;
    for (int i = 0; i < V; i++)
        if (i != s)
            vertex.push_back(i);

    int min_path = INT_MAX;
    do {
        int current_pathweight = 0;
        int k = s;

        for (int i = 0; i < vertex.size(); i++) {
            current_pathweight += graph[k][vertex[i]];
            k = vertex[i];
        }

        current_pathweight += graph[k][s];
        min_path = min(min_path, current_pathweight);

    } while (next_permutation(vertex.begin(), vertex.end()));

    return min_path;
}

int main() {
    int graph[][V] = {
        {0, 21, 32, 23},
        {31, 0, 43, 26},
        {65, 76, 0, 65},
        {43, 32, 45, 0}
    };

    int s = 0;

    cout <<"The Value is: "<< travllingSalesmanProblem(graph, s) << endl;

    return 0;
}
```

OUTPUT :-



```
/tmp/MJBrt2YKg7.o
The Value is: 157
```

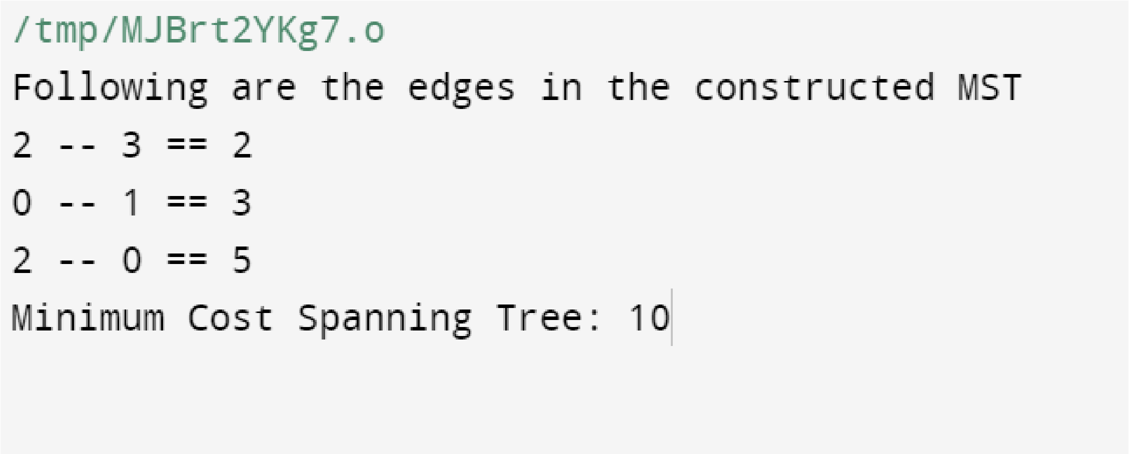
PROGRAM NO. :- 09

OBJECTIVE: Find Minimum Spanning Tree using Kruskal's Algorithm

CODE :-

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

class DSU {
    int* parent;
    int* rank;

public:
    DSU(int n) {
        parent = new int[n];
        rank = new int[n];

        for (int i = 0; i < n; i++) {
            parent[i] = -1;
            rank[i] = 1;
        }
    }

    int find(int i) {
        if (parent[i] == -1)
            return i;

        return parent[i] = find(parent[i]);
    }

    void unite(int x, int y) {
        int s1 = find(x);
        int s2 = find(y);

        if (s1 != s2) {
            if (rank[s1] < rank[s2]) {
                parent[s1] = s2;
                rank[s2] += rank[s1];
            } else {
                parent[s2] = s1;
                rank[s1] += rank[s2];
            }
        }
    }
};

class Graph {
    vector<vector<int>> edgelist;
    int V;

public:
    Graph(int V) {
        this->V = V;
    }

    void addEdge(int x, int y, int w) {
        edgelist.push_back({w, x, y});
    }

    void kruskals_mst() {
        sort(edgelist.begin(), edgelist.end());
        DSU s(V);
        int ans = 0;

        cout << "Following are the edges in the constructed MST" << endl;

        for (auto edge : edgelist) {
            int w = edge[0];
            int x = edge[1];
            int y = edge[2];

            if (s.find(x) != s.find(y)) {
                s.unite(x, y);
                ans += w;
                cout << x << " -- " << y << " == " << w << endl;
            }
        }

        cout << "Minimum Cost Spanning Tree: " << ans;
    }
};

int main() {
    Graph g(4);

    g.addEdge(0, 1, 3);
    g.addEdge(1, 3, 6);
    g.addEdge(2, 3, 2);
    g.addEdge(2, 0, 5);
    g.addEdge(0, 3, 7);

    // Function call
    g.kruskals_mst();

    return 0;
}
```

OUTPUT :-



```
/tmp/MJBrt2YKg7.o
Following are the edges in the constructed MST
2 -- 3 == 2
0 -- 1 == 3
2 -- 0 == 5
Minimum Cost Spanning Tree: 10
```

PROGRAM NO. :- 10

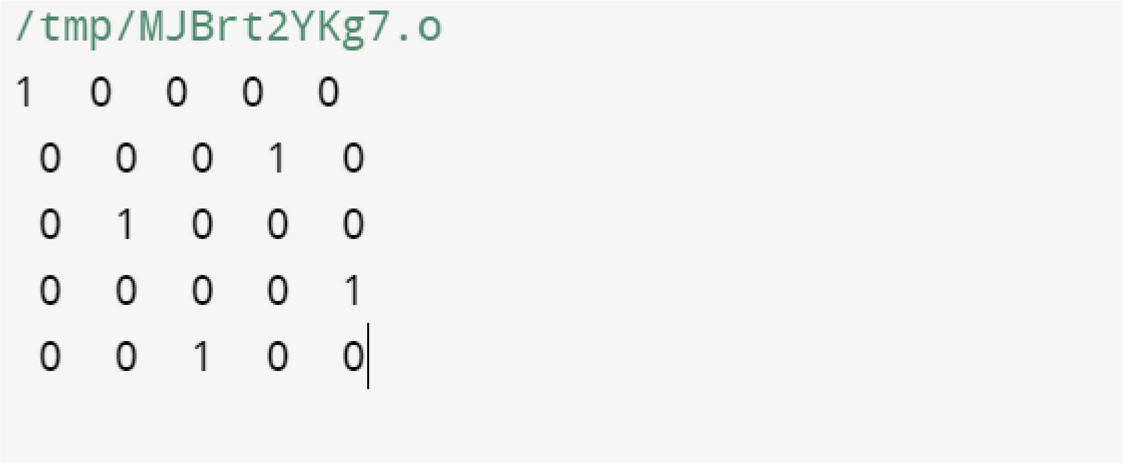
OBJECTIVE: Implement N Queen Problem using Backtracking

CODE :-

```
#include <bits/stdc++.h>
#define N 5
using namespace std;

void
printSolution (int board[N][N])
{
 for (int i = 0; i < N; i++)
   {
     for (int j = 0; j < N; j++)
   cout << " " << board[i][j] << " ";
     printf ("\n");
   }
}

bool
isSafe (int board[N][N], int row, int col)
{
 int i, j;

 // Check in the same row
 for (i = 0; i < col; i++)
   if (board[row][i])
     return false;

 // Check upper diagonal on the left side
 for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
   if (board[i][j])
     return false;

 // Check lower diagonal on the left side
 for (i = row, j = col; j >= 0 && i < N; i++, j--)
   if (board[i][j])
     return false;

 return true;
}

bool
solveNQUtil (int board[N][N], int col)
{
 if (col == N)
```

```
  return true;

 for (int i = 0; i < N; i++)
  {
   if (isSafe (board, i, col))
  {
   board[i][col] = 1;
   if (solveNQUtil (board, col + 1))
    return true;

   board[i][col] = 0;    // BACKTRACK
  }
  }

 return false;
}

bool
solveNQ ()
{
 int board[N][N] = {
  {0, 0, 0},
  {0, 0, 0},
  {0, 0, 0},
  {0, 0, 0},
  {0, 0, 0}
 };

 if (!solveNQUtil (board, 0))
  {
   cout << "Solution does not exist";
   return false;
  }

 printSolution (board);
 return true;
}

int
main ()
{
 solveNQ ();
 return 0;
}
```

OUTPUT :-



```
/tmp/MJBrt2YKg7.o
1   0   0   0   0
 0   0   0   1   0
 0   1   0   0   0
 0   0   0   0   1
 0   0   1   0   0
```