VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
on

Analysis and Design of Algorithms

Submitted by

Preeti T Korishettar (1BM22CS208)

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
April-2024 to August-2024

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated to Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "**Analysis and Design of Algorithms**" carried out by **Preeti T Korishettar (1BM22CS208),** who is bonafide student of **B.M.S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the academic semester April-2024 to August-2024. The Lab report has been approved as it satisfies the academic requirements in respect of an **Analysis and Design of Algorithms (23CS4PCADA)** work prescribed for the said degree.

M Lakshmi Neelima	**Dr. Jyothi S Nayak**
Assistant Professor	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

Index Sheet

	Prim’s algorithm. ➢ Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal’s algorithm.	
12	Implement Fractional Knapsack using Greedy technique.	
13	From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra’s algorithm.	
14	Implement “N-Queens Problem” using Backtracking.	

Course Outcome

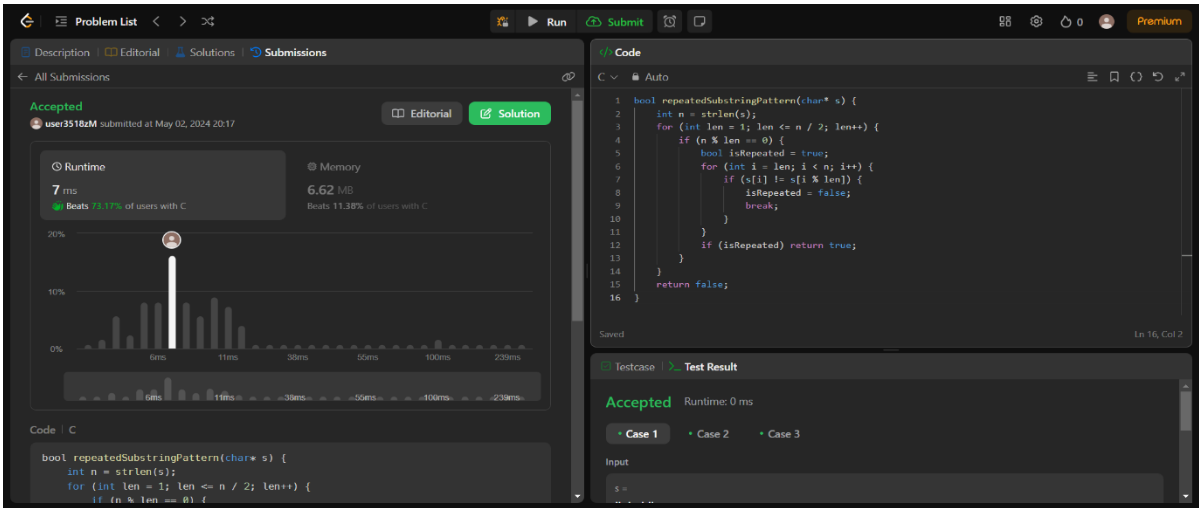
CO1	Analyze time complexity of Recursive and Non-recursive algorithms using asymptotic notations.
CO2	Apply various design techniques for the given problem.
CO3	Apply the knowledge of complexity classes P, NP, and NP-Complete and prove certain problems are NP-Complete
CO4	Design efficient algorithms and conduct practical experiments to solve problems.

1.Leetcode exercises on Stacks, Queues, Circular Queues, Priority Queues.

Repeated substring pattern

```
bool repeatedSubstringPattern(char* s) {
    int n = strlen(s);
    for (int len = 1; len <= n / 2; len++) {
        if (n % len == 0) {
            bool isRepeated = true;
            for (int i = len; i < n; i++) {
                if (s[i] != s[i % len]) {
                    isRepeated = false;
                    break;
                }}
            if (isRepeated) return true;
        } }
    return false;}
```

OUTPUT:



Accepted

user3518zM submitted at May 02, 2024 20:17

Runtime

7 ms

Beats 73.17% of users with C

Memory

6.62 MB

Beats 11.38% of users with C

Code | C

```
bool repeatedSubstringPattern(char* s) {
    int n = strlen(s);
    for (int len = 1; len <= n / 2; len++) {
        if (n % len == 0) {
```

```
bool repeatedSubstringPattern(char* s) {
    int n = strlen(s);
    for (int len = 1; len <= n / 2; len++) {
        if (n % len == 0) {
            bool isRepeated = true;
            for (int i = len; i < n; i++) {
                if (s[i] != s[i % len]) {
                    isRepeated = false;
                    break;
                }
            }
            if (isRepeated) return true;
        }
    }
    return false;
}
```

Saved

Ln 16, Col 2

Accepted Runtime: 0 ms

Case 1 • Case 2 • Case 3

Input

s =

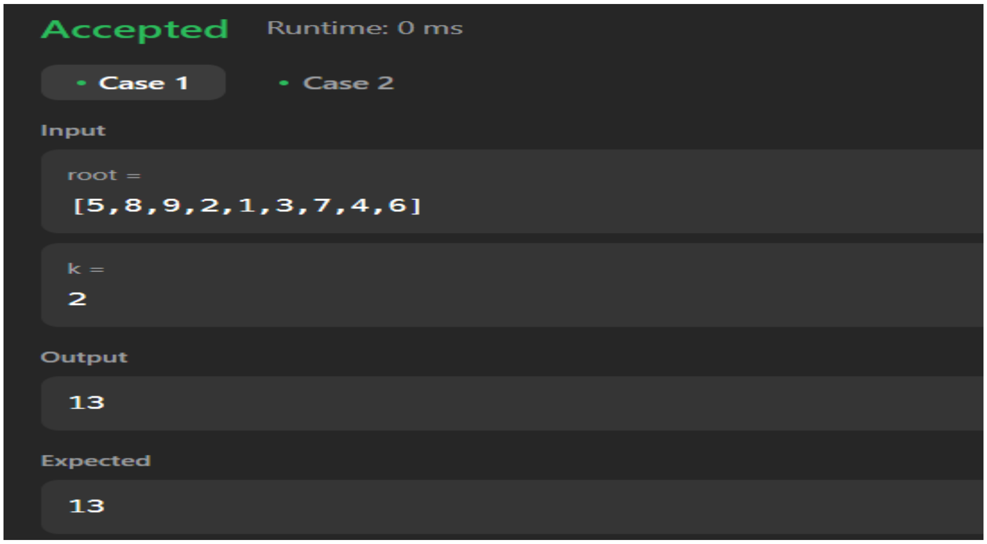
2 .Leetcode exercises on DFS, BFS.

Kth largest element in the tree

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */
void dfs(struct TreeNode* root, int level, long long levelSum[])
 {
    if (root == NULL)
        return;
    levelSum[level] += root->val;
    dfs(root->left, level + 1, levelSum);
    dfs(root->right, level + 1, levelSum);
}
long long kthLargestLevelSum(struct TreeNode* root, int k)
{
    if (root == NULL)
        return -1; // If the tree is empty

    long long* levelSum = (long long*)calloc(1000, sizeof(long long));

    dfs(root, 0, levelSum);
    long long* levelSums = (long long*)malloc(1000 * sizeof(long long));

    int numLevels = 0;
    for (int i = 0; i < 1000 && levelSum[i] != 0; ++i)
   {
        levelSums[numLevels++] = levelSum[i];
    }

    for (int i = 0; i < numLevels - 1; ++i) {
        for (int j = i + 1; j < numLevels; ++j) {
            if (levelSums[i] < levelSums[j]) {
                long long temp = levelSums[i];
                levelSums[i] = levelSums[j];
                levelSums[j] = temp;
            }
        }
    }
   if (k <= numLevels) {
```

```
 return levelSums[k - 1];
    }
    else {
        return -1; // If there are fewer than k levels in the tree
    }
}
```

OUTPUT:



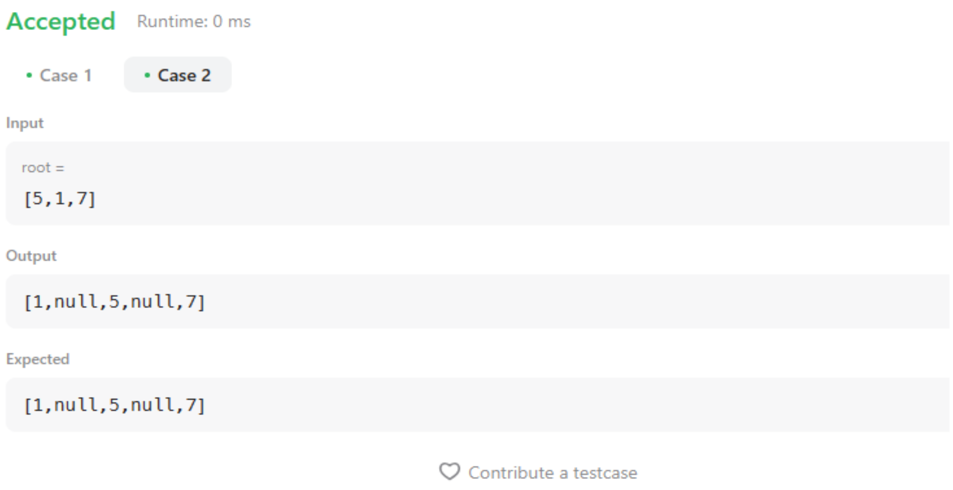
```
Accepted    Runtime: 0 ms
• Case 1    • Case 2
Input
root =
[5,8,9,2,1,3,7,4,6]
k =
2
Output
13
Expected
13
```

3.Leetcode exercises on Trees and Graphs.

Increasing order search tree.

```
/**

 * Definition for a binary tree node.

 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */
struct TreeNode* increasingBST(struct TreeNode* root) {
    if (!root)
        return root;

    struct TreeNode* lft = increasingBST(root->left);
    if (lft){
        struct TreeNode* temp = lft;
        while (temp->right)
        temp = temp->right;
        root->left = NULL;
        temp->right = root;
        root->right = increasingBST(root->right);
        root = lft;
    }
    else
        root->right = increasingBST(root->right);
 return root;
}
```

Output:



Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

root =

```
[5,1,7]
```

Output

```
[1,null,5,null,7]
```

Expected

```
[1,null,5,null,7]
```

WEEK -04

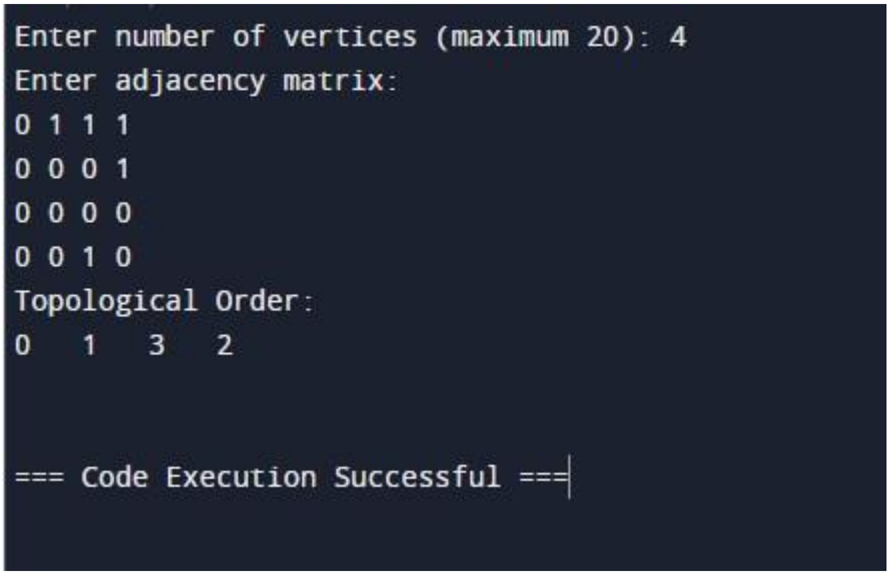
Program to obtain the Topological ordering of vertices in a given digraph

1)DFS Technique

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 100
int graph[MAX][MAX];
int visited[MAX];
int stack[MAX];
int top = -1;
int num_nodes;
void initializeGraph() {
    int i, j;
    for (i = 0; i < num_nodes; i++) {
        visited[i] = 0;
        stack[i] = -1;
        for (j = 0; j < num_nodes; j++) {
            graph[i][j] = 0;
        }}}
void addEdge(int from, int to) {
    graph[from][to] = 1;
}
void dfs(int node) {
    int i;
    visited[node] = 1;
    for (i = 0; i < num_nodes; i++) {
        if (graph[node][i] && !visited[i]) {
            dfs(i);
        }} stack[++top] = node;}
```

```
void topologicalSort() {
    int i;
    for (i = 0; i < num_nodes; i++) {
        if (!visited[i]) {
            dfs(i);}
    }
    printf("Topological Sorting Order: ");
    for (i = top; i >= 0; i--) {
        printf("%d ", stack[i]);
    }
    printf("\n");
}
int main() {
    int num_edges, i, from, to;
    printf("Enter the number of vertices: ");
    scanf("%d", &num_nodes);
    initializeGraph();
    printf("Enter the number of edges: ");
    scanf("%d", &num_edges);
    printf("Enter the edges (from to): \n");
    for (i = 0; i < num_edges; i++) {
        scanf("%d %d", &from, &to);
        addEdge(from, to);
    }
    topologicalSort();
    return 0;
}
```

OUTPUT:



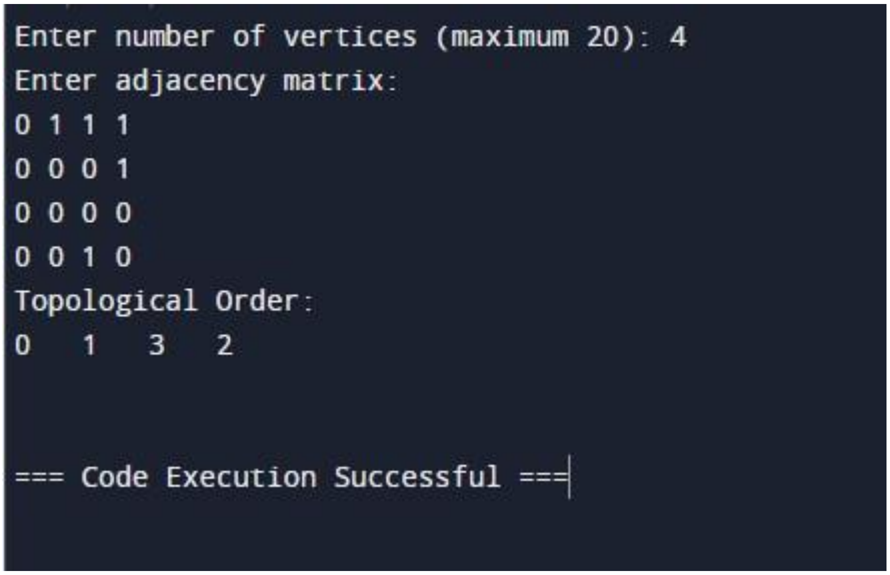
```
Enter number of vertices (maximum 20): 4
Enter adjacency matrix:
0 1 1 1
0 0 0 1
0 0 0 0
0 0 1 0
Topological Order:
0   1   3   2


=== Code Execution Successful ===
```

2)Source Removal Technique

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 100//source removal method
int graph[MAX][MAX];
int visited[MAX];
int indegree[MAX];
int num_nodes;
void initializeGraph() {
    int i, j;
    for (i = 0; i < num_nodes; i++) {
        visited[i] = 0;
        indegree[i] = 0;
        for (j = 0; j < num_nodes; j++) {
            graph[i][j] = 0;
        }
    }
```

```
}
void addEdge(int from, int to) {
    graph[from][to] = 1;
    indegree[to]++;
}
void topologicalSort() {
    int i, j, k;
    for (i = 0; i < num_nodes; i++) {
        // Find a node with indegree 0
        for (j = 0; j < num_nodes; j++) {
            if (!visited[j] && indegree[j] == 0) {
                visited[j] = 1;
                printf("%d ", j);
                // Remove edges starting from this node
                for (k = 0; k < num_nodes; k++) {
                    if (graph[j][k]) {
                        indegree[k]--;}}
                break;
            }}}
    printf("\n");
}
int main() {
    int num_edges, i, from, to;
    printf("Enter the number of nodes: ");
    scanf("%d", &num_nodes);
    initializeGraph();
    printf("Enter the number of edges: ");
    scanf("%d", &num_edges);
    printf("Enter the edges (from to): \n");
    for (i = 0; i < num_edges; i++) {
        scanf("%d %d", &from, &to);
        addEdge(from, to);
    }
    printf("Topological Sorting Order: ");
    topologicalSort();
    return 0;
}
```

OUTPUT:



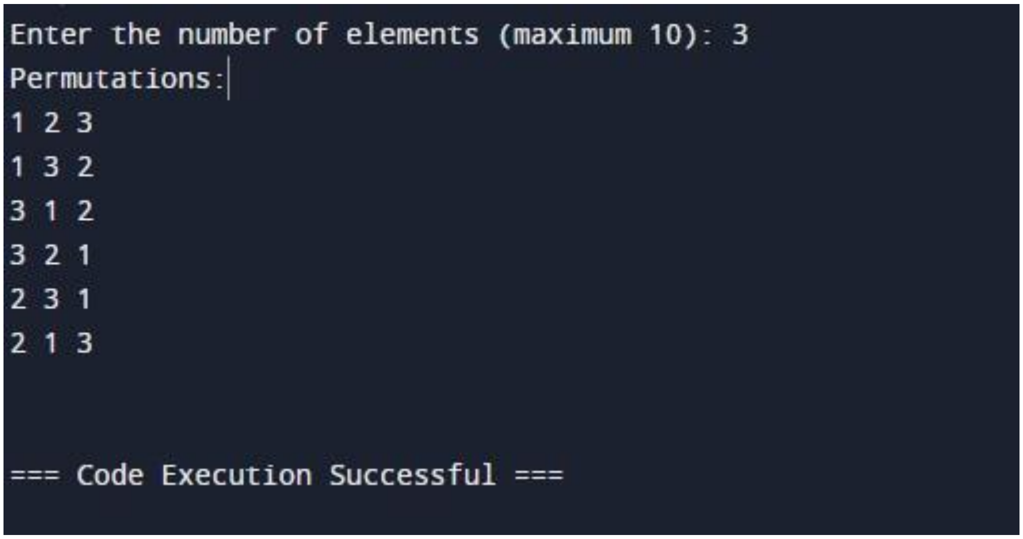
```
Enter number of vertices (maximum 20): 4
Enter adjacency matrix:
0 1 1 1
0 0 0 1
0 0 0 0
0 0 1 0
Topological Order:
0   1   3   2


=== Code Execution Successful ===
```

WEEK 05

Implementation of Johnson Trotter algorithm to generate permutations.

```
#include <stdio.h>
#include <stdbool.h>
#define MAXN 10
// Direction array, dir[i] stores the direction of ith element in permutation
int dir[MAXN];
int n; // Number of elements in the permutation
// Function to print the current permutation
void printPermutation(int perm[]) {
    for (int i = 0; i < n; i++)
        printf("%d ", perm[i]);
    printf("\n");
}
// Function to swap two integers
void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;}
// Function to generate all permutations using Johnson-Trotter algorithm
void generatePermutations() {
    int perm[MAXN]; // Current permutation
    for (int i = 0; i < n; i++) {
        perm[i] = i + 1; // Initialize permutation to 1, 2, ..., n
        dir[i] = -1; // All directions initially set to -1 (left)
    }
printPermutation(perm); // Print the initial permutation
 int k, mobile, pos;
    bool found;
while (true) {
        // Step 1: Find the largest mobile integer
        mobile = -1;
        for (int i = 0; i < n; i++) {
            if ((dir[i] == -1 && i != 0 && perm[i] > perm[i - 1]) ||
                (dir[i] == 1 && i != n - 1 && perm[i] > perm[i + 1])) {
                if (mobile == -1 || perm[i] > perm[mobile]) {
                    mobile = i;}}}
        if (mobile == -1) // No more mobile integers, algorithm terminates
            break;
      // Step 2: Swap the mobile integer with the adjacent integer it is pointing to
        k = mobile + dir[mobile];
        swap(&perm[mobile], &perm[k]);
        swap(&dir[mobile], &dir[k]);

        // Step 3: Reverse the direction of all integers greater than the mobile integer
        for (int i = 0; i < n; i++) {
            if (perm[i] > perm[k]) {
                dir[i] = -dir[i];}}
 // Print the current permutation
        printPermutation(perm);}}
int main() {
    printf("Enter the number of elements (maximum %d): ", MAXN);
    scanf("%d", &n);
    if (n <= 0 || n > MAXN) {
        printf("Invalid input. Number of elements should be between 1 and %d.\n", MAXN);
        return 1;
    } generatePermutations(); // Generate permutations using Johnson-Trotter algorithm
return 0;}
```

OUTPUT:



```
Enter the number of elements (maximum 10): 3
Permutations:
1 2 3
1 3 2
3 1 2
3 2 1
2 3 1
2 1 3


=== Code Execution Successful ===
```

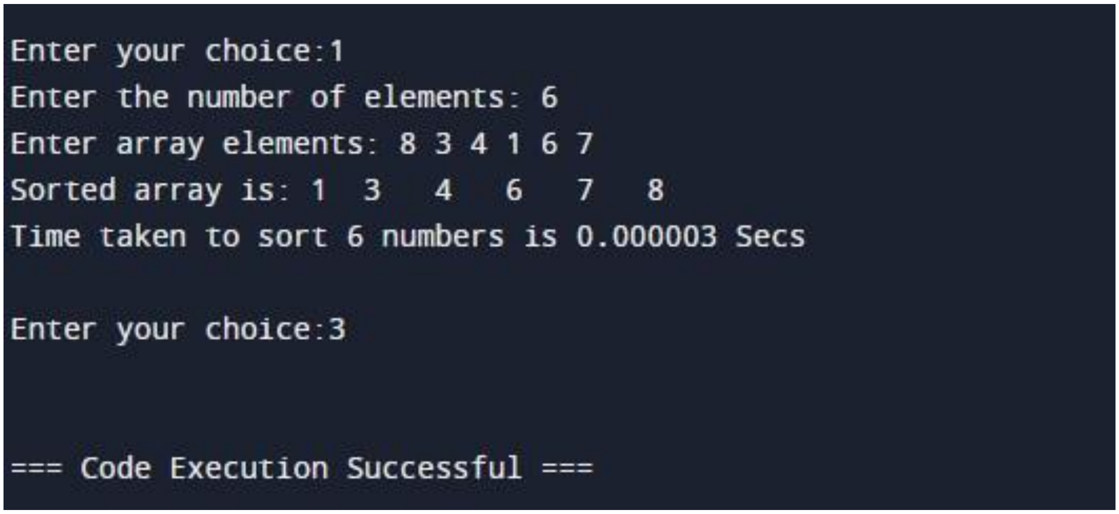
WEEK 06

Sort a given set of N integer elements using Merge Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort.

```
#include<stdio.h>
#include<time.h>
#include<stdlib.h> /* To recognise exit function when compiling with gcc*/
void split(int[],int,int);
void combine(int[],int,int,int);
void main()
{
  int a[15000],n, i,j,ch, temp;
  clock_t start,end;
  while(1)
  {
 printf("\n1:For manual entry of N value and array elements");
 printf("\n2:To display time taken for sorting number of elements N in the range 500 to 14500");
 printf("\n3:To exit");
    printf("\nEnter your choice:");
    scanf("%d", &ch);
    switch(ch)
    {
      case 1:  printf("\nEnter the number of elements: ");
                      scanf("%d",&n);
                      printf("\nEnter array elements: ");
                      for(i=0;i<n;i++)
                       {
                        scanf("%d",&a[i]);
                       }start=clock();
                      split(a,0,n-1);
```

```
                end=clock();
                printf("\nSorted array is: ");
                for(i=0;i<n;i++)
                printf("%d\t",a[i]);
printf("\n Time taken to sort %d numbers is %f Secs",n, (((double)(end-start))/CLOCKS_PER_SEC));
                break;
 case 2:
             n=500;
             while(n<=14500) {
             for(i=0;i<n;i++)
                  {
                   //a[i]=random(1000);
                   a[i]=n-i; }
             start=clock();
             split(a,0,n-1);
        //Dummy loop to create delay
           for(j=0;j<500000;j++){ temp=38/600;}
             end=clock();
printf("\n Time taken to sort %d numbers is %f Secs",n, (((double)(end-start))/CLOCKS_PER_SEC));
                 n=n+1000;}
            break;
  case 3: exit(0); }
  getchar();}}
void split(int a[],int low,int high){
 int mid;
 if(low<high) {
  mid=(low+high)/2;
  split(a,low,mid);
  split(a,mid+1,high);
  combine(a,low,mid,high);}}
void combine(int a[],int low,int mid,int high){
 int c[15000],i,j,k;
 i=k=low;
 j=mid+1;
 while(i<=mid&&j<=high){
  if(a[i]<a[j]) {
   c[k]=a[i];
   ++k;
   ++i; }
  else{
   c[k]=a[j];
   ++k;
   ++j; }}
 if(i>mid)
 {while(j<=high)
  {c[k]=a[j];
   ++k;
   ++j; }}
 if(j>high)
 { while(i<=mid) {
   c[k]=a[i];
   ++k;
   ++i; }}
 for(i=low;i<=high;i++)
 {
  a[i]=c[i];
 }
```

OUTPUT:



```
Enter your choice:1
Enter the number of elements: 6
Enter array elements: 8 3 4 1 6 7
Sorted array is: 1  3   4   6   7   8
Time taken to sort 6 numbers is 0.000003 Secs

Enter your choice:3


=== Code Execution Successful ===
```

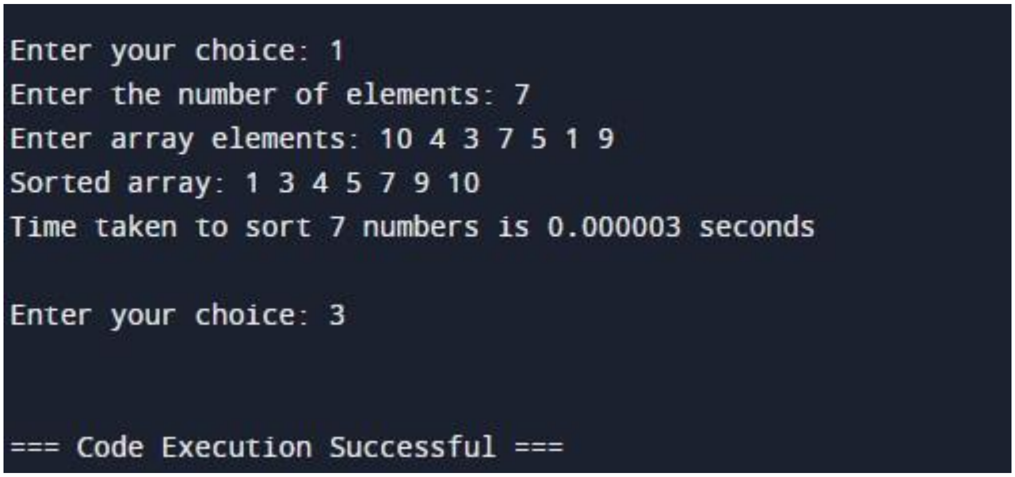
WEEK 07

Sort a given set of N integer elements using Quick Sort technique and compute its time taken.

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
int a[20];
void quicksort(int [],int ,int);
int partition(int [],int ,int);
void swap(int *,int *);
void main()
{
   int a[15000],n, i,j,ch, temp;
   clock_t start,end;
    int low,high;
   while(1)
   {
 printf("\n1:For manual entry of N value and array elements");
 printf("\n2:To display time taken for sorting number of elements N in the range 500 to 14500");
 printf("\n3:To exit");
     printf("\nEnter your choice:");
     scanf("%d", &ch);
     switch(ch)
     {
       case 1:  printf("\nEnter the number of elements: ");
                    scanf("%d",&n);
                    printf("\nEnter array elements: ");
                    for(i=0;i<n;i++)
                     {
                      scanf("%d",&a[i]);
                     }
                    low=0;
                    high=n-1;
                    start=clock();
                    quicksort(a,low,high);
                    end=clock();
                    printf("\nSorted array is: ");
                    for(i=0;i<n;i++)
          printf("%d\t",a[i]);
printf("\n Time taken to sort %d numbers is %f Secs",n, (((double)(end-
start))/CLOCKS_PER_SEC));
                    break;
    case 2:
                n=500;
                while(n<=14500) {
                for(i=0;i<n;i++)
                      {
          //a[i]=random(1000);
                       a[i]=n-i;
                      }
                start=clock();

                quicksort(a,0,n-1);
         //Dummy loop to create delay
            for(j=0;j<500000;j++){ temp=38/600;}
                end=clock();
printf("\n Time taken to sort %d numbers is %f Secs",n, (((double)(end-
start))/CLOCKS_PER_SEC));
                     n=n+1000;
                      }break;
```

```
   case 3: exit(0);
   }
   getchar();
    }
}
void quicksort(int a[],int low,int high)
{
    int split;
    if(low<high)
    {
        split=partition(a,low,high);
        quicksort(a,low,split-1);
        quicksort(a,split+1,high);
    }
}
 int partition(int a[],int low,int high)
 {
     int pivot=a[low];
     int i=low,j=high+1;
     printf("%d",i);
     do{
 do
        {
             i++;
        }while(a[i]<pivot);
        do
          {
              j--;
          }while(a[j]>pivot);
        swap(&a[i],&a[j]);
     }while(i<j);
     swap(&a[i],&a[j]);
     swap(&a[j],&a[low]);
     return j;
 }
void swap(int *a,int *b)
 { int temp;
     temp=*a;
     *a=*b;
     *b=temp; }
```

OUTPUT:



```
Enter your choice: 1
Enter the number of elements: 7
Enter array elements: 10 4 3 7 5 1 9
Sorted array: 1 3 4 5 7 9 10
Time taken to sort 7 numbers is 0.000003 seconds

Enter your choice: 3


=== Code Execution Successful ===
```

WEEK O8

Sort a given set of N integer elements using Heap Sort technique and compute its time taken.

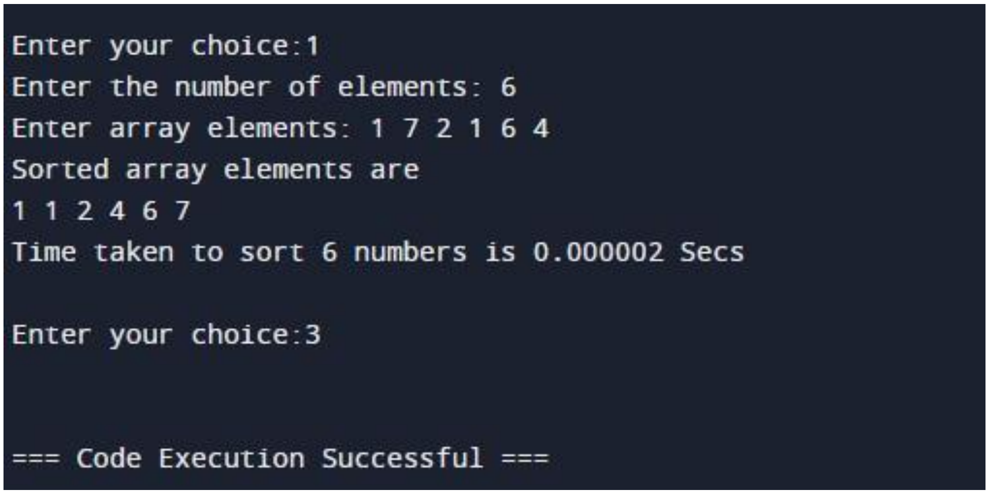
```
#include<stdio.h>
#include<time.h>
#include<stdlib.h>
void heapsort(int n, int a[]);
void heapify(int n, int a[]);
void swap(int* a, int* b);
void main()
{int a[15000], n, i, j, ch, temp;
    clock_t start, end;
 while (1)
    {
        printf("\n 1:For sorting of array elements");
        printf("\n 2:To display time taken for sorting number of elements N in the range 500 to 14500");
        printf("\n 3:To exit");
        printf("\nEnter your choice:");
        scanf("%d", &ch);
 switch (ch)
        {
        case 1:
            printf("\nEnter the number of elements: ");
            scanf("%d", &n);
            printf("\nEnter array elements: ");
            for (i = 0; i < n; i++)
            {
```

```
 scanf("%d", &a[i]);

            }

            start = clock();

            heapsort(n, a);

            end = clock();

             printf("Sorted array elements are\n");

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

    {

        printf("%d ", a[i]);

    }

    printf("\n");

            printf("\nTime taken to sort %d numbers is %f Secs\n", n, (((double)(end - start)) /
CLOCKS_PER_SEC));

            break;  case 2:

                  n=500;

      while(n<=14500) {

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

 {

   //a[i]=random(1000);

   a[i]=n-i;

 }start=clock();

      heapsort(n,a);

          //Dummy loop to create delay

    for(j=0;j<500000;j++){ temp=38/600;}

              end=clock();

         printf("\n Time taken to sort %d numbers is %f Secs",n, (((double)(end-
start))/CLOCKS_PER_SEC));

            n=n+1000; }
```

```
break;
case 3:
            exit(0);}
getchar(); // Consume newline character left in input buffer
    }
}
void heapify(int n, int a[])
{
    int i, p, c, item;
    for (p = (n - 1) / 2; p >= 0; p--)
    {
        item = a[p];
        c = 2 * p + 1;
        while (c < n)
        {
            if (c + 1 < n && a[c] < a[c + 1])
            {
                c++;
            }
            if (item >= a[c])
                break;
            a[p] = a[c];
            p = c;
            c = 2 * p + 1;
        }
        a[p] = item;
    }
```

```
}
void swap(int* a, int* b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
void heapsort(int n, int a[])
{
    int i;
    heapify(n, a);
    for (i = n - 1; i > 0; i--)
    {
        swap(&a[0], &a[i]);
        heapify(i, a);
    }
}
```

OUTPUT:



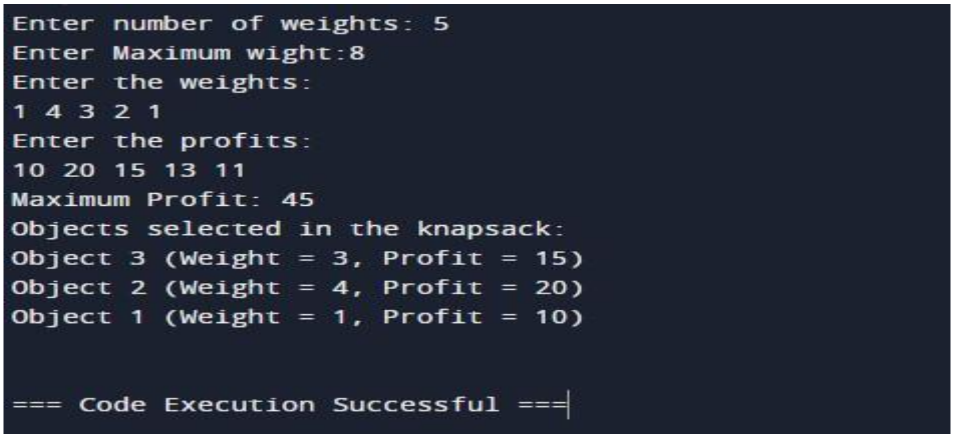
```
Enter your choice:1
Enter the number of elements: 6
Enter array elements: 1 7 2 1 6 4
Sorted array elements are
1 1 2 4 6 7
Time taken to sort 6 numbers is 0.000002 Secs

Enter your choice:3


=== Code Execution Successful ===
```

WEEK 09

Implement 0/1 Knapsack problem using dynamic programming.

```
#include <stdio.h>
#define N 4
int max(int a, int b) {
    return (a > b) ? a : b;
}
void knapsack(int W, int weights[], int profits[]) {
    int dp[N + 1][W + 1];
    for (int i = 0; i <= N; i++) {
        for (int w = 0; w <= W; w++) {
            if (i == 0 || w == 0)
                dp[i][w] = 0;
            else if (weights[i - 1] <= w)
                dp[i][w] = max(profits[i - 1] + dp[i - 1][w - weights[i - 1]], dp[i - 1][w]);
            else
                dp[i][w] = dp[i - 1][w];
        }
    }
    int maxProfit = dp[N][W];
    printf("Maximum Profit: %d\n", maxProfit);
    int w = W;
    printf("Objects selected in the knapsack:\n");
    for (int i = N; i > 0 && maxProfit > 0; i--) {
        if (maxProfit == dp[i - 1][w])
            continue;
        else {
            printf("Object %d (Weight = %d, Profit = %d)\n", i, weights[i - 1], profits[i - 1]);
            maxProfit -= profits[i - 1];
            w -= weights[i - 1];
        }
    }
}
int main() {
    int weights[20],n;
    int profits[20];
    printf("Enter number of weights: ");
    scanf("%d", &n);
    int W = 50;
    printf("Enter the weights:\n")
    for (int j = 0; j < n; j++) {
            scanf("%d", weights[i]);
        }
    printf("Enter the profits:\n");
    for (int j = 0; j < n; j++) {
            scanf("%d", profits[i]);
        }
    knapsack(W, weights, profits);
    return 0;
}
```

OUTPUT:



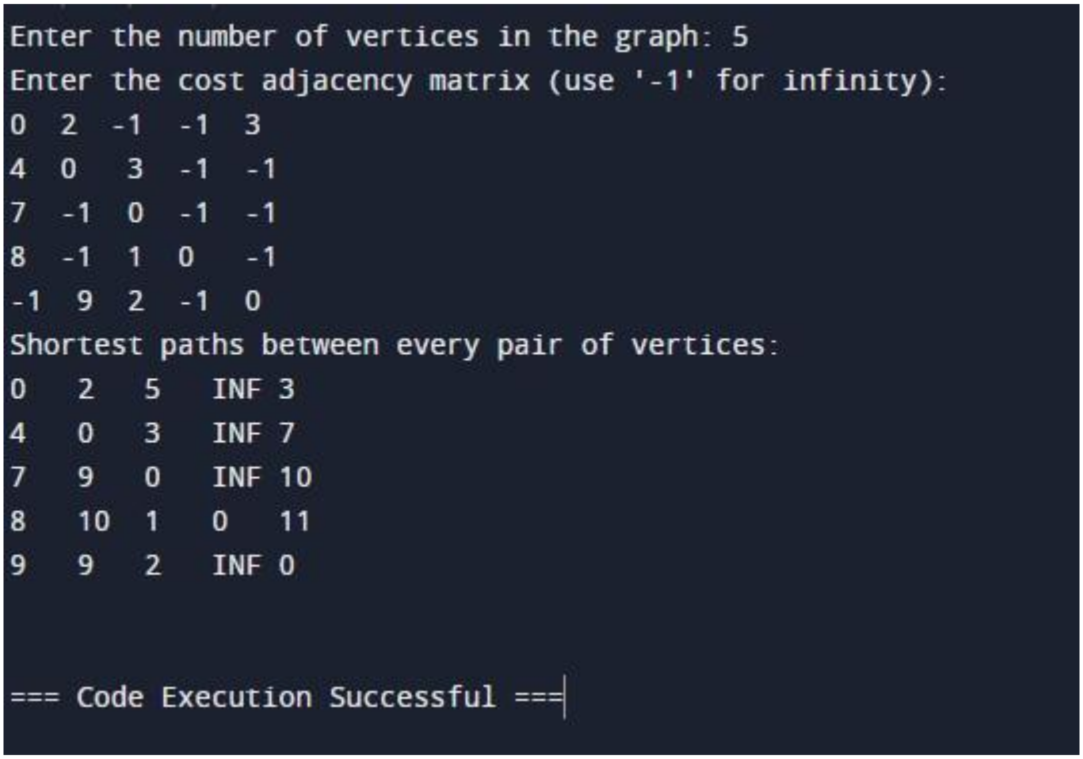
```
Enter number of weights: 5
Enter Maximum wight:8
Enter the weights:
1 4 3 2 1
Enter the profits:
10 20 15 13 11
Maximum Profit: 45
Objects selected in the knapsack:
Object 3 (Weight = 3, Profit = 15)
Object 2 (Weight = 4, Profit = 20)
Object 1 (Weight = 1, Profit = 10)


=== Code Execution Successful ===
```

WEEK 10

Implement All Pair Shortest paths problem using Floyd’s algorithm

```
#include <stdio.h>
#include <limits.h>
void floyd(int n, int cost[][n], int D[][n]) {
    int i, j, k;
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            D[i][j] = cost[i][j];
        }
    }
for (k = 0; k < n; k++) {
        for (i = 0; i < n; i++) {
            for (j = 0; j < n; j++) {
                if (D[i][k] != INT_MAX && D[k][j] != INT_MAX && D[i][j] > D[i][k] + D[k][j]) {
                    D[i][j] = D[i][k] + D[k][j];
                }
            }
        }
    }
}
void printShortestPaths(int n, int D[][n]) {
    printf("Shortest paths between every pair of vertices:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (D[i][j] == INT_MAX) {
                printf("INF\t");
            } else {
                printf("%d\t", D[i][j]);
            }
        }
        printf("\n");
    }
}
int main() {
    int n;
    printf("Enter the number of vertices in the graph: ");
    scanf("%d", &n);

    int cost[n][n];
    printf("Enter the cost adjacency matrix (use 'INF' for infinity):\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &cost[i][j]);
            if (cost[i][j] == -1) {
                cost[i][j] = INT_MAX;
            }
        }
    }
    int D[n][n];
    floyd(n, cost, D); printShortestPaths(n, D);
return 0;}
```

OUTPUT:



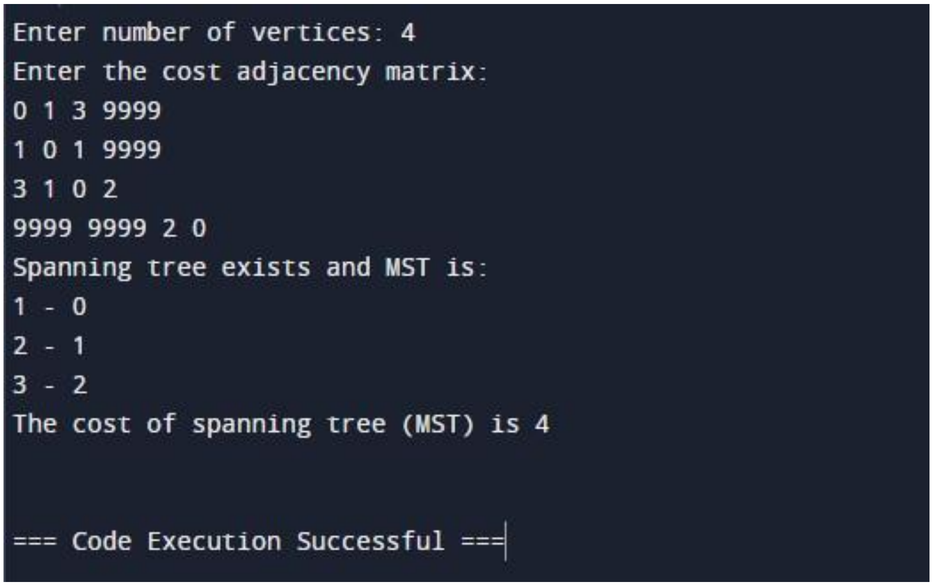
```
Enter the number of vertices in the graph: 5
Enter the cost adjacency matrix (use '-1' for infinity):
0  2  -1  -1  3
4  0   3  -1  -1
7  -1  0  -1  -1
8  -1  1  0   -1
-1  9  2  -1  0
Shortest paths between every pair of vertices:
0   2   5   INF 3
4   0   3   INF 7
7   9   0   INF 10
8   10  1   0   11
9   9   2   INF 0


=== Code Execution Successful ===
```

WEEK 11

- **Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s algorithm.**

```
#include <stdio.h>
#define MAX 9999
void prims(int n, int cost[n][n]) {
    int i, j, u, min, sum = 0, source, K = 0;
    int S[n], d[n], P[n], T[n-1][2];
    min = MAX;
    source = 0;
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            if (cost[i][j] != 0 && cost[i][j] < min) {
                min = cost[i][j];
                source = i;
            }
        }
    }
    for (i = 0; i < n; i++) {
        S[i] = 0;
        d[i] = cost[source][i];
        P[i] = source;
    }
    S[source] = 1;
    for (i = 1; i < n; i++) {
        min = MAX;
        u = -1;
        for (j = 0; j < n; j++) {
            if (S[j] == 0 && d[j] <= min) {
                min = d[j];
                u = j;
            }
        }
        T[K][0] = u;
        T[K][1] = P[u];
        K++;
        sum += cost[u][P[u]];
        S[u] = 1;
        for (j = 0; j < n; j++) {
            if (S[j] == 0 && cost[u][j] < d[j]) {
                d[j] = cost[u][j];
                P[j] = u;
            }
        }
    }
    if (sum >= MAX) {
        printf("Spanning tree does not exist.\n");
    } else {
        printf("Spanning tree exists and MST is:\n");
        for (i = 0; i < n-1; i++) {
            printf("%d - %d\n", T[i][0], T[i][1]);
        }
        printf("The cost of spanning tree (MST) is %d\n", sum);
    }
}
int main() {
    int n;
    printf("Enter number of vertices: ");
    scanf("%d", &n);
    int cost[n][n];
    printf("Enter the cost adjacency matrix:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &cost[i][j]);
        }
    }
prims(n, cost);
return 0;
}
```

OUTPUT:

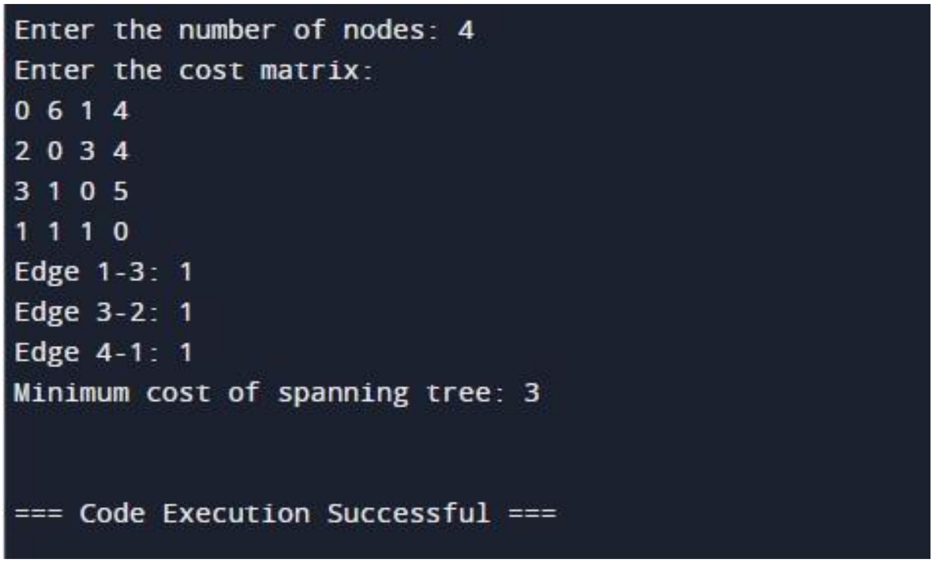


```
Enter number of vertices: 4
Enter the cost adjacency matrix:
0 1 3 9999
1 0 1 9999
3 1 0 2
9999 9999 2 0
Spanning tree exists and MST is:
1 - 0
2 - 1
3 - 2
The cost of spanning tree (MST) is 4


=== Code Execution Successful ===
```

- **Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal’s algorithm.**

```
#include <stdio.h>
#define MAX 9999 // Infinity value assumed
void kruskals(int c[][100], int n);
int main() {
    int n, i, j;
    int c[100][100]; // Assuming a maximum size for the cost matrix
     printf("Enter the number of nodes: ");
    scanf("%d", &n);
     printf("Enter the cost matrix:\n");
    for (i = 1; i <= n; i++) {
        for (j = 1; j <= n; j++) {
            scanf("%d", &c[i][j]);
            if (c[i][j] == 0) // Assuming 0 represents no edge, set it to a large value
                c[i][j] = MAX;
        }
    } kruskals(c, n);
return 0;}
void kruskals(int c[][100], int n) {
    int ne = 0, mincost = 0;
    int parent[100];
    int min, u, v, a, b, i, j;
for (i = 1; i <= n; i++)
        parent[i] = 0;
while (ne != n - 1) {
        min = MAX;
for (i = 1; i <= n; i++) {
            for (j = 1; j <= n; j++) {
                if (c[i][j] < min) {
                    min = c[i][j];
                    u = a = i;
                    v = b = j;}}}
        while (parent[u] != 0)
            u = parent[u];
     while (parent[v] != 0)
            v = parent[v];
     if (u != v) {
            printf("Edge %d-%d: %d\n", a, b, min);
            parent[v] = u;
            mincost += min;
            ne++;
        }c[a][b] = c[b][a] = MAX;
    } printf("Minimum cost of spanning tree: %d\n", mincost);
}
```

OUTPUT:



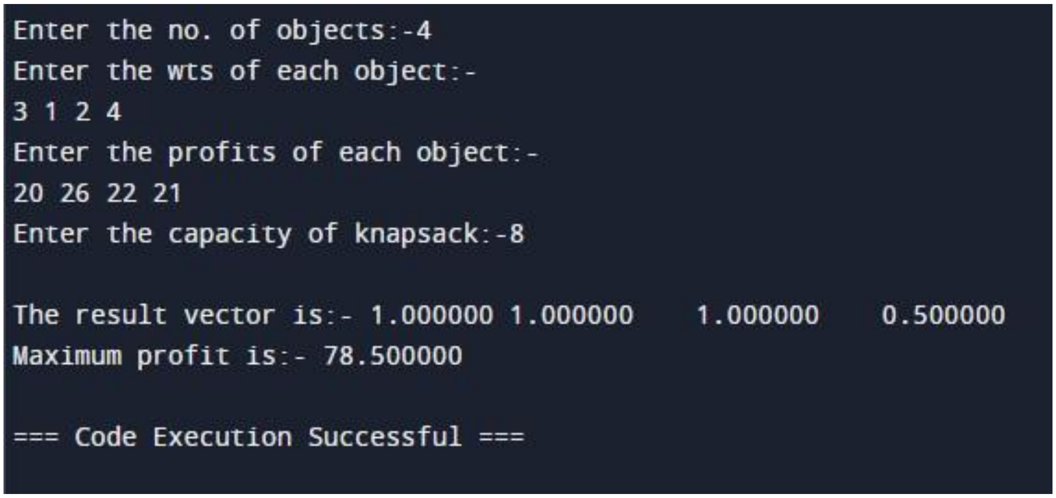
```
Enter the number of nodes: 4
Enter the cost matrix:
0 6 1 4
2 0 3 4
3 1 0 5
1 1 1 0
Edge 1-3: 1
Edge 3-2: 1
Edge 4-1: 1
Minimum cost of spanning tree: 3


=== Code Execution Successful ===
```

WEEK 12

Implement Fractional Knapsack using Greedy technique.

```
#include<stdio.h>
void knapsack(int n, float weight[], float profit[], float capacity)
{
   float x[20], tp = 0;
   int i, j, u;
   u = capacity;
   for (i = 0; i < n; i++)
      x[i] = 0.0;
   for (i = 0; i < n; i++) {
      if (weight[i] > u)
  break;
      else {
         x[i] = 1.0;
         tp = tp + profit[i];
         u = u - weight[i];
      }}
   if (i < n)
      x[i] = u / weight[i];
   tp = tp + (x[i] * profit[i]);
   printf("\nThe result vector is:- ");
   for (i = 0; i < n; i++)
      printf("%f\t", x[i]);
   printf("\nMaximum profit is:- %f", tp);
}
int main() {
   float weight[20], profit[20], capacity;
   int num, i, j;
   float ratio[20], temp;
   printf("\nEnter the no. of objects:- ");
   scanf("%d", &num);
   printf("\nEnter the wts and profits of each object:- ");
   for (i = 0; i < num; i++) {
      scanf("%f %f", &weight[i], &profit[i]);
   }
   printf("\nEnter the capacity of knapsack:- ");
   scanf("%f", &capacity);
   for (i = 0; i < num; i++) {
      ratio[i] = profit[i] / weight[i];
   }
for (i = 0; i < num; i++) {
      for (j = i + 1; j < num; j++) {
         if (ratio[i] < ratio[j]) {
           temp = ratio[j];
           ratio[j] = ratio[i];
           ratio[i] = temp;
           temp = weight[j];
           weight[j] = weight[i];
           weight[i] = temp;
           temp = profit[j];
           profit[j] = profit[i];
           profit[i] = temp;
         }
      }
   }
   knapsack(num, weight, profit, capacity);
   return(0);
}
```

OUTPUT:



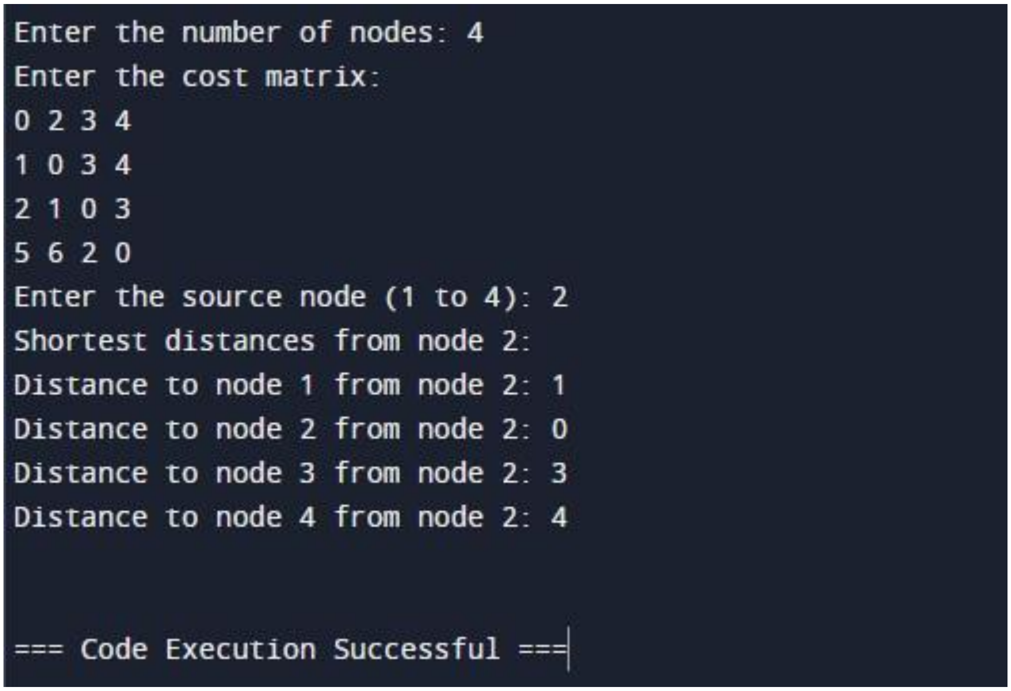
```
Enter the no. of objects:-4
Enter the wts of each object:-
3 1 2 4
Enter the profits of each object:-
20 26 22 21
Enter the capacity of knapsack:-8

The result vector is:- 1.000000 1.000000    1.000000    0.500000
Maximum profit is:- 78.500000

=== Code Execution Successful ===
```

WEEK 13

From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra’s algorithm.

```
#include <stdio.h>
#define MAX 9999 // Infinity value assumed
void dijkstras(int c[][100], int n, int src);
int main() {
    int n, src, i, j;
    int c[100][100]; // Assuming a maximum size for the cost matrix
    printf("Enter the number of nodes: ");
    scanf("%d", &n);
    printf("Enter the cost matrix:\n");
    for (i = 1; i <= n; i++) {
        for (j = 1; j <= n; j++) {
            scanf("%d", &c[i][j]);
        }
    }
    printf("Enter the source node (1 to %d): ", n);
    scanf("%d", &src);
    dijkstras(c, n, src);
    return 0;
}
void dijkstras(int c[][100], int n, int src) {
    int dist[100], vis[100];
    int count, min, u, i, j;
    // Initialization
    for (j = 1; j <= n; j++) {
        dist[j] = c[src][j];
        vis[j] = 0;
    }
    dist[src] = 0;
    vis[src] = 1;
    count = 1;
    // Main loop
    while (count != n) {
        min = MAX;
        // Find the minimum distance vertex from the set of vertices not yet processed
        for (j = 1; j <= n; j++) {
            if (dist[j] < min && vis[j] != 1) {
                min = dist[j];
                u = j;
            }
        }
        vis[u] = 1;
        count++;
        // Update dist value of the adjacent vertices of the picked vertex
        for (j = 1; j <= n; j++) {
            if (min + c[u][j] < dist[j] && vis[j] != 1) {
                dist[j] = min + c[u][j];
            }
        }
    }
    // Output shortest distances
    printf("Shortest distances from node %d:\n", src);
    for (j = 1; j <= n; j++) {
        printf("Distance to node %d from node %d: %d\n", j, src, dist[j]);
    }
}
```

OUTPUT:



```
Enter the number of nodes: 4
Enter the cost matrix:
0 2 3 4
1 0 3 4
2 1 0 3
5 6 2 0
Enter the source node (1 to 4): 2
Shortest distances from node 2:
Distance to node 1 from node 2: 1
Distance to node 2 from node 2: 0
Distance to node 3 from node 2: 3
Distance to node 4 from node 2: 4


=== Code Execution Successful ===
```

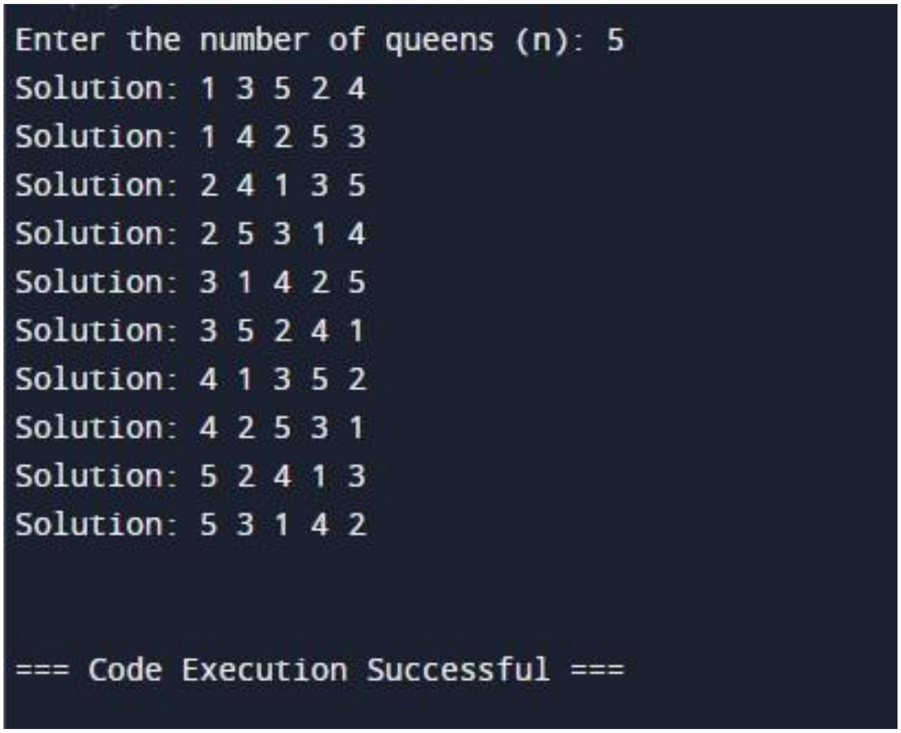
WEEK 14

Implement “N-Queens Problem” using Backtracking.

```
#include<stdio.h>
#define BOARD_SIZE 5
void displayChess(int chBoard[BOARD_SIZE][BOARD_SIZE]) {
  for (int row = 0; row < BOARD_SIZE; row++) {
    for (int col = 0; col < BOARD_SIZE; col++)
      printf("%d ", chBoard[row][col]);
    printf("\n");
  }
}
int isQueenPlaceValid(int chBoard[BOARD_SIZE][BOARD_SIZE], int crntRow, int crntCol) {
  // checking if queen is in the left or not
  for (int i = 0; i < crntCol; i++)
    if (chBoard[crntRow][i])
      return 0;
  for (int i = crntRow, j = crntCol; i >= 0 && j >= 0; i--, j--)
    //checking if queen is in the left upper diagonal or not
    if (chBoard[i][j])
      return 0;
  for (int i = crntRow, j = crntCol; j >= 0 && i < BOARD_SIZE; i++, j--)
    //checking if queen is in the left lower diagonal or not
    if (chBoard[i][j])
      return 0;
  return 1;
}
int solveProblem(int chBoard[BOARD_SIZE][BOARD_SIZE], int crntCol) {
  //when N queens are placed successfully
  if (crntCol >= BOARD_SIZE)
    return 1;
  // checking placement of queen is possible or not
  for (int i = 0; i < BOARD_SIZE; i++) {
    if (isQueenPlaceValid(chBoard, i, crntCol)) {
      //if validate, place the queen at place (i, col)
      chBoard[i][crntCol] = 1;
      //Go for the other columns recursively
      if (solveProblem(chBoard, crntCol + 1))
        return 1;
      //When no place is vacant remove that queen
      chBoard[i][crntCol] = 0;
    }
  }
  return 0;
}
int displaySolution() {
  int chBoard[BOARD_SIZE][BOARD_SIZE];
  for(int i = 0; i < BOARD_SIZE; i++)
    for(int j = 0; j < BOARD_SIZE; j++)
      //set all elements to 0
      chBoard[i][j] = 0;
  //starting from 0th column
  if (solveProblem(chBoard, 0) == 0) {
    printf("Solution does not exist");
    return 0;
  }
  displayChess(chBoard);
  return 1;
}
```

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

OUTPUT:



```
Enter the number of queens (n): 5
Solution: 1 3 5 2 4
Solution: 1 4 2 5 3
Solution: 2 4 1 3 5
Solution: 2 5 3 1 4
Solution: 3 1 4 2 5
Solution: 3 5 2 4 1
Solution: 4 1 3 5 2
Solution: 4 2 5 3 1
Solution: 5 2 4 1 3
Solution: 5 3 1 4 2


=== Code Execution Successful ===
```