1)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int bocount = 0;
void quick_sort(int a[20], int l, int h);

void quick_sort(int a[20], int low, int high)
{
    int pivot, i, j, temp;
    bocount += 100;
    if (low < high)
    pivot = low;
    i = low;
    j = high;
    while (i < j)
    {
        while (a[i] <= a[pivot] && i <= high)
            i++;
        while (a[j] > a[pivot] && j >= low)
            j--;
        if (i < j)
        {
            temp = a[i];
            a[i] = a[j];
            a[j] = temp;
        }
        temp = a[j];
        a[j] = a[pivot];
        a[pivot] = temp;
        quick_sort(a, low, j - 1);
        quick_sort(a, j + 1, high);
```

```
    }
}
int main()
{
    int a[20], n, i;
    printf("Enter the size of the array : ");
    scanf("%d", &n);
    printf("Enter %d Elements : \n", n);

    for (i = 0; i < n; i++)
        scanf("%d", &a[i]);

    quick_sort(a, 0, n - 1);

    printf("\nArray after Sorting : \n");
    for (i = 0; i < n; i++)
        printf("%d\t", a[i]);
    printf("\nTherotical Time is %f", (float)bocount / n);

    return 0;
}
```

******OUTPUT******

```
Enter the size of the array : 4
Enter 4 Elements :
21 11 78 12

Array after Sorting :
11      12      21      78
Therotical Time is 325.000000
```

2)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int bocount = 0;
int l, m, h;
int main()
{
    int a[20], n, i;
    void mergesort(int a[], int l, int h);
    printf("Enter the size of the array : \n");
    scanf("%d", &n);
    printf("Enter %d elements : \n", n);
    for (i = 0; i < n; i++)
        scanf("%d", &a[i]);

    mergesort(a, 0, n - 1);

    printf("Array after Sorting : \n");
    for (i = 0; i < n; i++)
        printf("%d\n", a[i]);

    printf("Therotical Time is %f", (float)bocount / n);

    return 0;
}

void mergesort(int a[], int l, int h)
{
    int m;
    void merge(int a[], int l, int h, int m);
    bocount += 100;
```

```
    if (l < h)
    {
        m = (l + h) / 2;
        mergesort(a, l, m);
        mergesort(a, m + 1, h);
        merge(a, l, h, m);
    }
}
void merge(int a[], int l, int h, int m)
{
    int i, j, k, b[10];
    i = l;
    j = m + 1;
    k = 0;
    while ((i <= m) && (j <= h))
    {
        if (a[i] < a[j])
        {
            b[k] = a[i];
            i++;
        }
        else
        {
            b[k] = a[j];
            j++;
        }
        k++;
    }
    while (i <= m)
    {
        b[k] = a[i];
        k++;
```

```
        i++;
    }
    while (j <= h)
    {
        b[k] = a[j];
        k++;
        j++;
    }
    i = l;
    k = 0;
    while (i <= h)
    {
        a[i] = b[k];
        i++;
        k++;
    }
}
```

******OUTPUT******

```
Enter the size of the array :  4
Enter 4 elements :  4 3 2 1
Array after Sorting :
1 2 3 4
Therotical time is 180.000000
```

3)

```
#include <stdio.h>

void findindegree(int[10][10], int[10], int);
void topological(int, int[10][10]);

void topological(int n, int a[10][10])
{
    int k, top, t[100], i, stack[20], u, v, indegree[20];
    k = 1;
    top = -1;
    findindegree(a, indegree, n);
    for (i = 1; i <= n; i++)
    {
        if (indegree[i] == 0)
        {
            stack[++top] = i;
        }
    }
    while (top != -1)
    {
        u = stack[top--];
        t[k++] = u;
        for (v = 1; v < n; v++)
        {
            if (a[u][v] == 1)
            {
                indegree[v]--;
                if (indegree[v] == 0)
                {
                    stack[++top] = v;
                }
```

```
            }
        }
    }
    printf("\nTopological Sequence is \n");
    for (i = 1; i < n; i++)
        printf("%d\t", t[i]);
}

void findindegree(int a[10][10], int indegree[10], int n)
{
    int i, j, sum;
    for (j = 1; j <= n; j++)
    {
        sum = 0;
        for (i = 1; i <= n; i++)
        {
            sum = sum + a[i][j];
        }
        indegree[j] = sum;
    }
}
int main()
{
    int a[10][10], i, j, n;
    printf("Enter the number of nodes: ");
    scanf("%d", &n);
    printf("\nEnter the adjacency matrix : \n");
    for (i = 1; i <= n; i++)
        for (j = 1; j <= n; j++)
            scanf("%d", &a[i][j]);
    topological(n, a);
```

```
    return 0;
}
```

******OUTPUT******

```
Enter the number of nodes: 6

Enter the adjacency matrix :
0 0 1 1 0 0
0 0 0 1 1 0
0 0 0 1 0 1
0 0 0 0 0 1
0 0 0 0 0 0
0 0 0 0 0 1

Topological Sequence is
2       5       1       3       4
```

4)

```
#include <stdio.h>

int s, c[100][100], ver;
float optimum = 999, sum;

void swap(int v[], int i, int j)
{
    int t;
    t = v[i];
    v[i] = v[j];
    v[j] = t;
}
void brute_force(int v[], int n, int i)
{
    int j, sum1, k;
    if (i == n)
    {
        if (v[0] == s)
        {
            for (j = 0; j < n; j++)
                printf("%d\t", v[j]);
            sum1 = 0;
            for (k = 0; k < n - 1; k++)
            {
                sum1 = sum1 + c[v[k]][v[k + 1]];
            }
            sum1 = sum1 + c[v[n - 1]][s];
            printf("Sum = %d\n", sum1);
            if (sum1 < optimum)
                optimum = sum1;
        }
```

```
    }
    else
        for (j = i; j < n; j++)
        {
            swap(v, i, j);
            brute_force(v, n, i + 1);
            swap(v, i, j);
        }
}
int main()
{
    int ver, v[100], i, j;

    printf("Enter n : ");
    scanf("%d", &ver);

    for (i = 0; i < ver; i++)
        v[i] = i + 1;

    printf("Enter Cost Matrix : \n");
    for (i = 1; i <= ver; i++)
        for (j = 1; j <= ver; j++)
            scanf("%d", &c[i][j]);

    printf("\nEnter source :");
    scanf("%d", &s);

    brute_force(v, ver, 0);
    printf("\nOptimum solution with brute force technique is =
%f\n", optimum);
}
```

******OUTPUT******

```
Enter n : 4
Enter Cost Matrix :
1 2 3 4
5 6 7 8
3 4 5 6
9 8 4 3

Enter source :1 4 3 2 1
1       2       3       4       Sum = 24
1       2       4       3       Sum = 17
1       3       2       4       Sum = 24
1       3       4       2       Sum = 22
1       4       3       2       Sum = 17
1       4       2       3       Sum = 22

Optimum solution with brute force technique is = 17.000000
```

5)

```
#include <stdio.h>
#include <conio.h>
#define MAX 50

int p[MAX], w[MAX], n;
int max(int a, int b);

int knapsack(int i, int m)
{
    if (i == n)
        return (w[n] > m) ? 0 : p[n];
    if (w[i] > m)
        return knapsack(i + 1, m);
    return max(knapsack(i + 1, m), knapsack(i + 1, m - w[i]) +
p[i]);
}
int max(int a, int b)
{
    if (a > b)
        return a;
    else
        return b;
}

int main()
{
    int m, i, optsoln;

    printf("Enter no of objects: ");
    scanf("%d", &n);

    printf("Enter the weights: ");
```

```
    for (i = 1; i <= n; i++)
        scanf("%d", &w[i]);

    printf("Enter the profits: ");
    for (i = 1; i <= n; i++)
        scanf("%d", &p[i]);

    printf("Enter the knapsack capacity: ");
    scanf("%d", &m);

    optsoln = knapsack(1, m);

    printf("The Optimal solution is : %d", optsoln);

    return 0;
}
```

******OUTPUT******

```
Enter no of objects: 3
Enter the weights: 100 20
50 10
Enter the profits: 150 30
Enter the knapsack capacity: 50
The Optimal solution is : 150
```

6)

```
#include <stdio.h>

int nodes;
int f = 0, r = -1, i, j, q[10], visited[10], a[10][10];

void bfs(int v)
{
    for (i = 1; i <= nodes; i++)
        if (a[v][i] == 1 && visited[i] == 0)
            q[++r] = i;
    if (f <= r)
    {
        visited[q[f]] = 1;
        bfs(q[f++]);
    }
}
int main()
{
    int v;
    printf("Enter the number of nodes : ");
    scanf("%d", &nodes);
    for (i = 1; i <= nodes; i++)
    {
        q[i] = 0;
        visited[i] = 0;
    }
    printf("Read adjacency matrix : \n");
    for (i = 1; i <= nodes; i++)
        for (j = 1; j <= nodes; j++)
            scanf("%d", &a[i][j]);
    printf("Enter the source node : ");
```

```
    scanf("%d", &v);
    bfs(v);
    printf("The node which are reachable are : ");
    for (i = 1; i <= nodes; i++)
        if (visited[i] == 1)
            printf("%d\t", i);

        else
            printf("%d Node is not Reachable\n", i);
    return 0;
}
```

******OUTPUT******

```
Enter the number of nodes : 4
Read adjacency matrix :
0 1 1 1
0 0 0 1
0 0 0 0
0 0 1 0
Enter the source node : 1
The node which are reachable are : 2    3     4
```

7)

```
#include <stdio.h>
#include <time.h>
int nodes;
int i, j, count = 0;
int visited[10], a[10][10];
void dfs(int u)
{
    visited[u] = 1;
    for (i = 1; i <= nodes; i++)
        if (a[u][i] == 1 && visited[i] == 0)
        {
            dfs(i);
        }
}

int main()
{
    printf("Enter the number of nodes : \n");
    scanf("%d", &nodes);
    for (i = 1; i <= nodes; i++)
    {
        visited[i] = 0;
        for (j = 1; j <= nodes; j++)
            a[i][j] = 0;
    }
    printf("Read Adjacency matrix : \n");
    for (i = 1; i <= nodes; i++)
    {
        for (j = 1; j <= nodes; j++)
        {
            scanf("%d", &a[i][j]);
```

```
        }
    }
    printf("Enter the Source Node : ");
    scanf("%d", &i);
    dfs(i);
    for (i = 1; i <= nodes; i++)
    {
        if (visited[i])
            count++;
    }
    if (count == nodes)
        printf("Graph is connected");
    else
        printf("Graph is not connected");

    return 0;
}
```

******OUTPUT******

```
Enter the number of nodes :
4
Read Adjacency matrix :
0 1 1 1
0 0 0 1
0 0 0 0
0 0 1 0
Enter the Source Node : 1
Graph is connected
```

8)

```
#include <stdio.h>
int loc;
int main()
{
    int a[50], k, n = 0, check = 0, i;
    int bsearch(int, int, int[]);
    printf("Enter size of the Array : ");
    scanf("%d", &n);

    printf("Enter the elements : ");
    for (i = 0; i < n; i++)
        scanf("%d", &a[i]);

    printf("Enter the key element to be Searched : ");
    scanf("%d", &k);

    check = bsearch(k, n, a);
    if (check == 1)
        printf("\nSearch Successful & Element found at Location %d",
loc + 1);
    else
        printf("\nSearch Unsuccessfull");

    return 0;
}
int bsearch(int key, int size, int aray[])
{
    int low, high, mid;
    low = 0;
    high = size - 1;
    mid = (low + high) / 2;
    while ((key != aray[mid]) && (low <= high))
```

```
    {
        if (key < aray[mid])
            high = mid - 1;
        else
            low = (low + high) / 2;
    }
    if (key == aray[mid])
    {
        loc = mid;
        return 1;
    }
    else
        return 0;
}
```

******OUTPUT******

```
Enter size of the Array : 5
Enter size of the elements : 3 21 11 4 1
Enter the key element to be Searched : 11

Search Successful & Element found at Location 3
```

9)

```
#include <stdio.h>

int main()
{
    int n, array[1000], i, j, t;
    printf("\n Enter Size : ");
    scanf("%d", &n);

    printf("\n Enter %d integers : \n", n);
    for (i = 0; i < n; i++)
    {
        scanf("%d", &array[i]);
    }

    for (i = 1; i <= n - 1; i++)
    {
        j = i;
        while (j > 0 && array[j] < array[j - 1])
        {
            t = array[j];
            array[j] = array[j - 1];
            array[j - 1] = t;
            j--;
        }
    }

    printf("Sorted list in Assending order : \n");
    for (i = 0; i <= n - 1; i++)
    {
        printf("%d\t", array[i]);
    }
```

```
    return 0;
}
```

******OUTPUT******

```
Enter Size : 4
Enter 4 integers :
12 13 17 3
Sorted list in Assending order :
 3  12  13  17
```

10)

```
#include <stdio.h>
#include <conio.h>
int n, c[20][20], i, j, visited[20];

void prim()
{
    int min, b, a, count = 0, cost = 0;
    min = 1000;
    visited[1] = 1;
    printf(" minimum cost spaning tree is\n");
    while (count < n - 1)
    {
        min = 999;
        for (i = 1; i <= n; i++)
            for (j = 1; j <= n; j++)
                if (visited[i] && !visited[j] && min > c[i][j])
                {
                    min = c[i][j];
                    a = i;
                    b = j;
                }
        printf("%d-->%d\n", a, b);
        cost += c[a][b];
        visited[b] = 1;
        count++;
    }
    printf("The total cost of spaning tree is %d\n", cost);
}
```

```
int main()
{
    printf("Enter the number of vertices : \n");
    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]);
        visited[i] = 0;
    }
    prim();

    return 0;
}
```

******OUTPUT******

```
Enter the number of vertices :
3
Enter the cost matrix :
99 2 3
2 99 5
3 5 99
Minimum cost spaning tree is
1-->2
1-->3
The total cost of spaning tree is 5
```