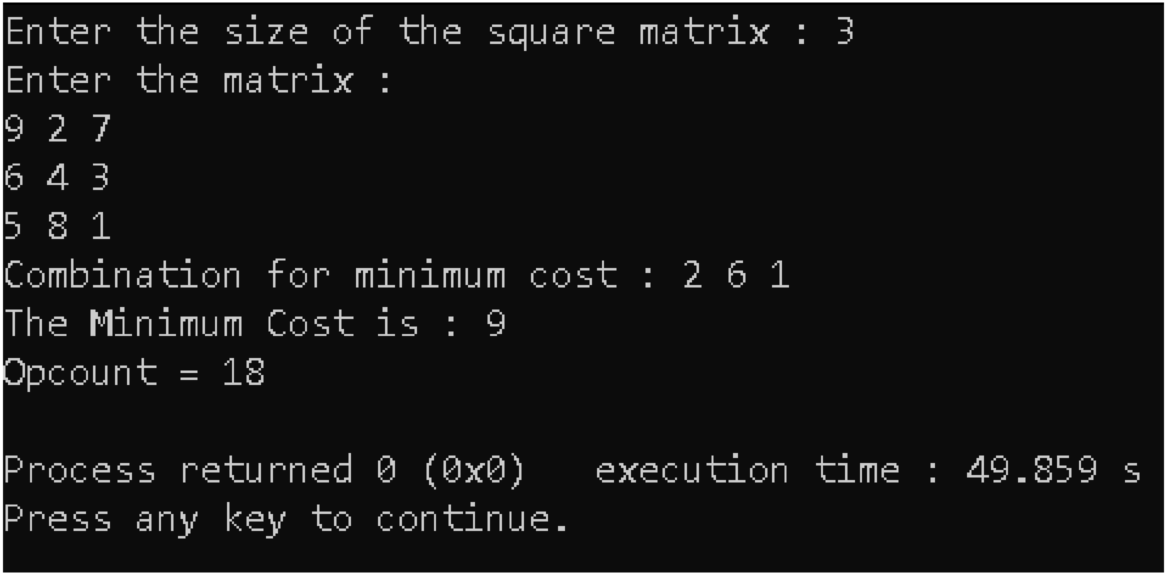
Lab Exercises:

1). Write a program for assignment problem by brute-force technique and analyse its time efficiency. Obtain the experimental result of order of growth and plot the result.

Program:

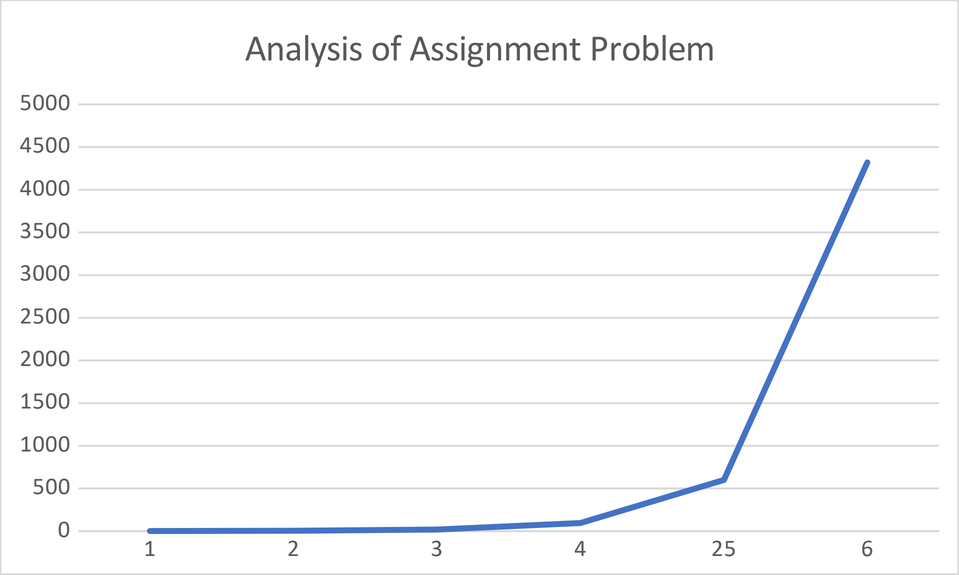
```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

int ans[1000], min = INT_MAX, opcount1 = 0, opcount2 = 0;

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

void permute(int r, int arr[][r + 1], int per[], int l)
{
    int i;
    if (l == r)
    {
        int sum = 0;
        for (i = 0; i <= r; i++)
        {
            opcount2++;
            int idx = per[i];
            sum += arr[i][idx];
        }
        if (sum < min)
        {
            for (i = 0; i <= r; i++)
            {
                int idx = per[i];
                ans[i] = arr[i][per[i]];
            }
            min = sum;
        }
    }
    else
    {
        for (i = l; i <= r; i++)
        {
            opcount1++;
            swap((per + l), (per + i));
            permute(r, arr, per, l + 1);
            swap((per + l), (per + i));
        }
    }
}

int main()
{
    int i, j, n;
    printf("Enter the size of the square matrix : ");
    scanf("%d", &n);
    int arr[n][n];
    printf("Enter the matrix : \n");
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
            scanf("%d", &arr[i][j]);
                }
    int per[n];
    for (i = 0; i < n; i++)
        per[i] = i;
    permute(n - 1, arr, per, 0);
    printf("Combination for minimum cost : ");
    for (i = 0; i < n; i++)
        printf("%d ", ans[i]);
    printf("\nThe Minimum Cost is : %d\n", min);
    printf("Opcount = %d\n", opcount1 > opcount2 ? opcount1 :
opcount2);
    return 0;
}
```

Output:



```
Enter the size of the square matrix : 3
Enter the matrix :
9 2 7
6 4 3
5 8 1
Combination for minimum cost : 2 6 1
The Minimum Cost is : 9
Opcount = 18

Process returned 0 (0x0)   execution time : 49.859 s
Press any key to continue.
```

Time Complexity Analysis:



2). Write a program for depth-first search of a graph. Identify the push and pop order of vertices.

Algorithm:

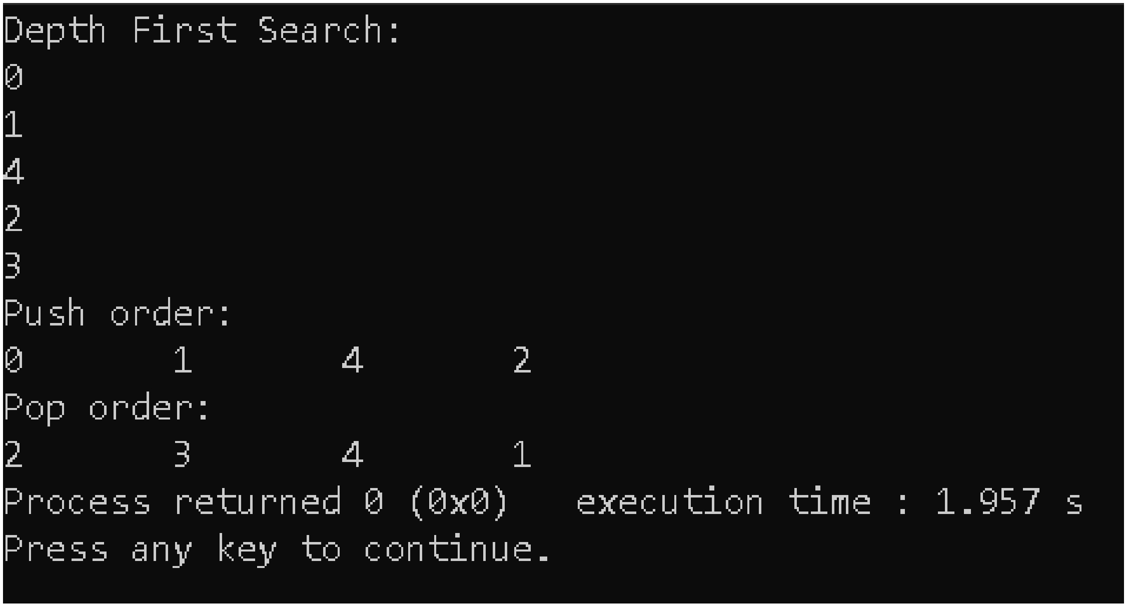
```
ALGORITHM DFS(G)
//Implements a depth-first search traversal of a given graph
//Input: Graph G = V, E
//Output: Graph G with its vertices marked with consecutive
integers
// in the order they are first encountered by the DFS
traversal
mark each vertex in V with 0 as a mark of being “unvisited”
count ←0
for each vertex v in V do
      if v is marked with 0
          dfs(v)

dfs(v)
//visits recursively all the unvisited vertices connected to
vertex v
//by a path and numbers them in the order they are encountered
//via global variable count
count ←count + 1; mark v with count
for each vertex w in V adjacent to v do
      if w is marked with 0
          dfs(w)
```

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#define MAX 5
int pushstk[100],popstk[100];
struct Vertex {
   char label;
   bool visited;
};
int stack[MAX];
int top = -1;
struct Vertex* lstVertices[MAX];
int adjMatrix[MAX][MAX];
int vertexCount = 0;
void push(int item) {
   stack[++top] = item;
}
int pop() {
   return stack[top--];
}
int peek() {
   return stack[top];
}
bool isStackEmpty() {
   return top == -1;
}
void addVertex(char label) {
   struct Vertex* vertex = (struct Vertex*)
malloc(sizeof(struct Vertex));
   vertex->label = label;
   vertex->visited = false;
   lstVertices[vertexCount++] = vertex;
}
void addEdge(int start,int end) {
   adjMatrix[start][end] = 1;
   adjMatrix[end][start] = 1;
}
void displayVertex(int vertexIndex) {
   printf("%c\n",lstVertices[vertexIndex]->label);
}
int getAdjUnvisitedVertex(int vertexIndex) {
   int i;

   for(i = 0; i < vertexCount; i++) {
      if(adjMatrix[vertexIndex][i] == 1 && lstVertices[i]-
>visited == false) {
         return i;
      }
   }
   return -1;
}
void depthFirstSearch() {
   int i=0,j=1;
   lstVertices[0]->visited = true;
   displayVertex(0);
   push(0);
   pushstk[0] = peek();
   while(!isStackEmpty()) {
      int unvisitedVertex = getAdjUnvisitedVertex(peek());
      if(unvisitedVertex == -1) {
         popstk[i]=pop();
         i++;
      }
      else {
         lstVertices[unvisitedVertex]->visited = true;
         displayVertex(unvisitedVertex);
         push(unvisitedVertex);
         pushstk[j]=peek();
         j++;
      }
   }
   for(i = 0;i < vertexCount;i++) {
      lstVertices[i]->visited = false;
   }
}

int main() {
   int i, j;
   for(i = 0; i < MAX; i++){
      for(j = 0; j < MAX; j++)
         adjMatrix[i][j] = 0;
   }

   addVertex('0');
   addVertex('1');
   addVertex('2');
   addVertex('3');
   addVertex('4');

   addEdge(0, 1);
   addEdge(0, 2);
   addEdge(0, 3);
   addEdge(1, 4);
   addEdge(2, 4);
   addEdge(3, 4);

   printf("Depth First Search: \n ");
   depthFirstSearch();
   int size1 = 0;
   int size2 = 0;
   for(i=0;i<100;i++){
    if(pushstk[i]!=0)
        size1++;
   }
   for(i=0;i<100;i++){
    if(popstk[i]!=0)
        size2++;
   }
```

```
    printf("Push order - \n");
    for(i=0;i<size1;i++){
        printf("%d \t",pushstk[i]);
    }
    printf("\n");
    printf("Pop order - \n");
    for(i=0;i<size2;i++){
        printf("%d \t",popstk[i]);
    }
    return 0;
}
```

Output:



```
Depth First Search:
0
1
4
2
3
Push order:
0       1       4       2
Pop order:
2       3       4       1
Process returned 0 (0x0)   execution time : 1.957 s
Press any key to continue.
```

3). Write a program for breadth-first search of a graph.

Algorithm:

```
ALGORITHM BFS(G)
//Implements a breadth-first search traversal of a given graph
//Input: Graph G = V, E
//Output: Graph G with its vertices marked with consecutive
integers
// in the order they are visited by the BFS traversal
mark each vertex in V with 0 as a mark of being “unvisited”
count ←0
for each vertex v in V do
    if v is marked with 0
        bfs(v)


bfs(v)
//visits all the unvisited vertices connected to vertex v
//by a path and numbers them in the order they are visited
```

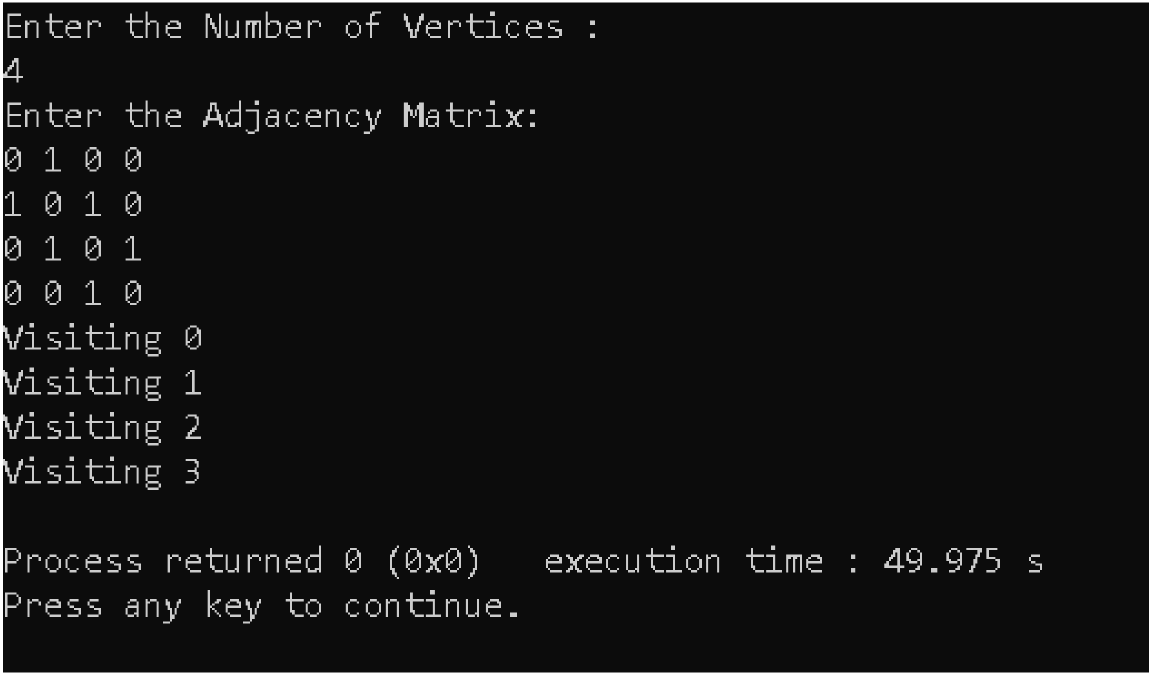
```
//via global variable count
count ←count + 1; mark v with count and initialize a queue
with v
while the queue is not empty do
     for each vertex w in V adjacent to the front vertex do
          if w is marked with 0
               count ←count + 1; mark w with count
               add w to the queue
      remove the front vertex from the queue
```

Program:

```c
#include <stdio.h>
#include <stdlib.h>

int g[100][100];
int V;
int visited[100];
int queue[100], f = 0, r = 0;

void enqueue(int v)
{
     queue[r++] = v;
}

int dequeue()
{
     if(f == r)
     {
          return -1;
     }

     return queue[f++];
}

void bfsv(int v)
{
     printf("Visiting %d\n", v);
     visited[v] = 1;

     int i;

     for(i = 0; i < V; ++i)
     {
          if(!visited[i] && g[v][i] && i != v)
          {
               enqueue(i);
          }
     }
}

void bfs()
```

```c
{
    int i, x;
    enqueue(0);

    do
    {
        x = dequeue();

        if(x != -1 && !visited[x])
        {
            bfsv(x);
        }
    }while (x != -1);
}

int main()
{
    printf("Enter the Number of Vertices : \n");
    scanf(" %d", &V);

    int i, j;

    printf("Enter the Adjacency Matrix: \n");

    for (i = 0; i < V; ++i)
    {
        for (j = 0; j < V; ++j)
        {
            scanf(" %d", &g[i][j]);
        }
    }

    bfs();

    return 0;
}
```

Output:



```
Enter the Number of Vertices :
4
Enter the Adjacency Matrix:
0 1 0 0
1 0 1 0
0 1 0 1
0 0 1 0
Visiting 0
Visiting 1
Visiting 2
Visiting 3

Process returned 0 (0x0)   execution time : 49.975 s
Press any key to continue.
```