DAA lab 4

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

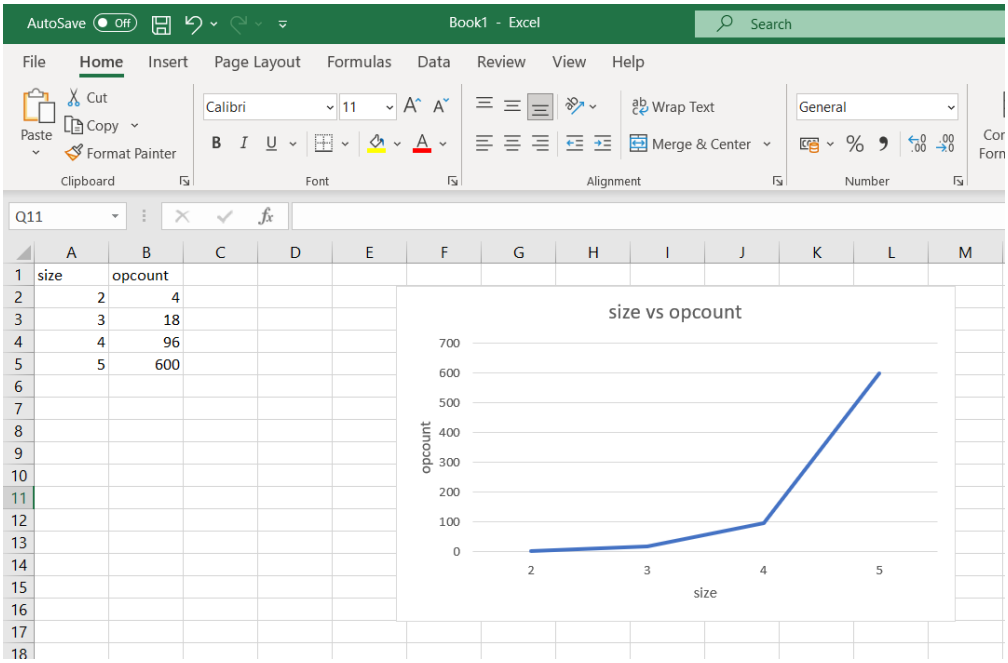
Code –

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

int ans[1000], min = INT_MAX, opcount;

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

void permuter(int l, int r, int arr[][r + 1], int per[])
{
   int i;
   if (l == r)
   {
      int sum = 0;
      for (i = 0; i <= r; i++)
      {
         opcount++;
         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++)
        {
            swap((per + l), (per + i));
            permuter(l+1, r, arr, per);
            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;
    permuter(0, n - 1, arr, per);
    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", opcount);
    return 0;

}
```

Execution –



size	opcount
2	4
3	18
4	96
5	600

Analysis –

The order of growth of this algorithm is seen to be O(n*n!). It is seen by the values in the graph when plotted. Basic operation taken is finding the sum of costs for all n! cases.

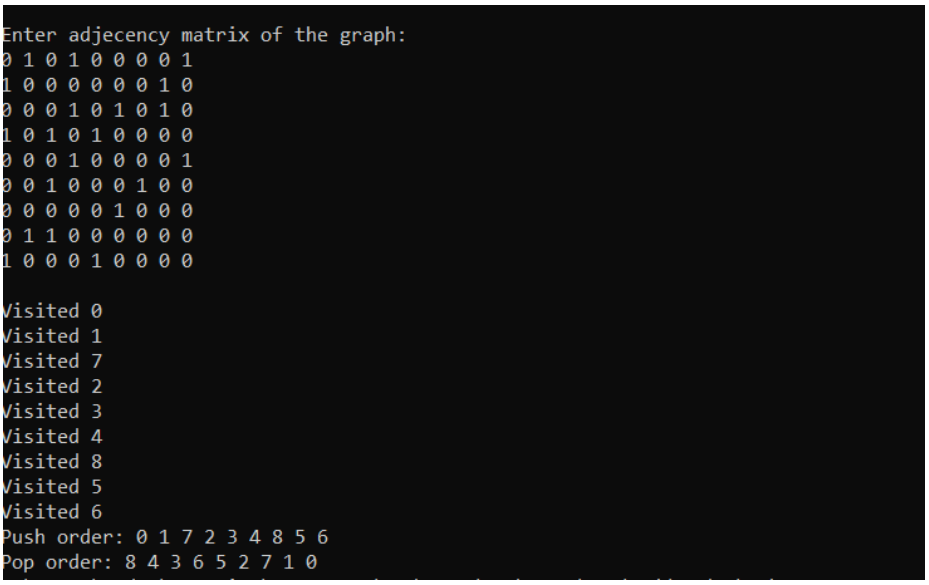
2). Write a program for depth-first search of a graph. Identify the push and pop order of vertices.

Code –

```
#include<stdio.h>

int G[10][10], visited[10], n, pushed[10], popped[10], k1=0, k2=0;

void DFS(int i)
{
   int j;
   pushed[k1++] = i;
         printf("\nVisited %d",i);
   visited[i]=1;

         for(j=0;j<n;j++)
     if(!visited[j]&&G[i][j]==1){
        DFS(j);
```

```
    }
  popped[k2++] = i;
}

int main()
{
  int i,j;
  printf("Enter number of vertices:");

        scanf("%d",&n);

        printf("\nEnter adjecency matrix of the graph:\n");

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

 for(i=0;i<n;i++)
     visited[i]=0;

  DFS(0);
  printf("\nPush order: ");
  for(i=0;i<n;++i){
     printf("%d ",pushed[i]);
  }
  printf("\nPop order: ");
  for(i=0;i<n;++i){
     printf("%d ",popped[i]);
  }
  return 0;
}
```

Execution –



Analysis –

We use stack for DFS. A graph with n vertices is considered here. Time spent for insertion and deletion of items from stack is O(1) for each item. Since n vertices, therefore O(n). To check adjacency, we can iterate over n vertices for n vertices therefore $O(|n|^2)$ order of growth. For adjacency list it's O(|n| + |e|) where e is number of edges.

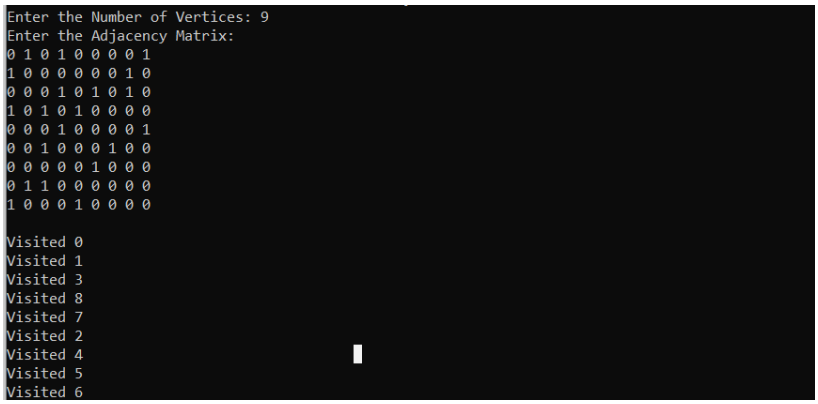
3). Write a program for breadth-first search of a graph.

Code –

```
#include <stdio.h>

#include <stdlib.h>


int g[100][100];

int n;

int visited[100];

int queue[100], f = 0, r = 0;


void enqueue(int v)

{

   if(f == -1)
```

```
        f = 0;
            queue[r++] = v;
}

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

            return queue[f++];
}

void bfs()
{
            int i, v;
            enqueue(0);
            do
            {
                        v = dequeue();

                        if(v != -1 && !visited[v])
                        {
                                    printf("\nVisited %d", v);
                  visited[v] = 1;

                        int i;

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

```
            if(!visited[i] && g[v][i] && i != v)
            {
               enqueue(i);
            }
         }
      }
         }while (v != -1);
}

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

         int i, j;

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

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

         bfs();

         return 0;
}
```



Analysis –

We use queue for BFS. A graph with n vertices is considered here. Time spent for insertion and deletion of items from stack is O(1) for each item. Since n vertices, therefore O(n). To check adjacency, we can iterate over n vertices for n vertices therefore $O(|n|^2)$ order of growth. For adjacency list it's O(|n| + |e|) where e is number of edges.