EAST WEST UNIVERSITY

Report of Mini Project

Discrete Mathematic, CSE-106
Sec 3, Group no 7(Odd)

Group member 1:
Name: Md.Ariful Islam Anik
Id: 2021-1-60-028
Group member 2:
Name: Al-Amin Sarker
Id: 2021-1-60-029
Group member 3:
Name: Anika Tahsin
Id: 2021-1-60-021

Submitted to:
Prof. Dr. Md. Mozammel Huq Azad Khan

Approach :

Adjacency Matrix is a 2D array of size n x n where n is the number of vertices in a graph. Let the 2D array be matrix[][], a slot matrix[i][j] = 1 indicates that there is an edge from vertex i to vertex j. According to our project instruction, we have to randomly generate a directed graph represented by adjacency matrix with n=1000 vertices using C program and determine in-degrees and out degrees of all vertices and show that the sum of in-degrees and sum of out-degrees are equal. Below are the steps:

1. In the C program, first, we enterd our necessary header files,2D matrix, and variables for coding the program. We also used the 'clock_t start, end' function to calculate the computing time.

```
#include<stdio.h> int main () { int
indeg,outdeg,total_in=0,total_out=0;
```

2. To generate directed adjacency matrix we have used nested for loop in 2d array where rand()%2 will generate the random values between 0,1. To generate new values every time we used srand(time(0)). Also, we have applied a condition of 'if (i==j) { matrix[i][j] = 0; }' to make the diagonal elements 0. Hence, we find the random adjacency matrix of a directed graph.

```
  for(int i=0; i<n; i++) {
      for(int j=0; j<n; j++){
          matrix[i][j]=rand()%2;
            if(i==j){
              matrix[i][j]=0;}}}
```

3. To calculate the In-degree and Out-degree, we used another nested loop with the condition of if (matrix[i][j]==1) then count In-degree and If(matrix[j][i]==1) then count Out-degree.

```
for(int i=0; i<n; i++) {
indeg=0;  outdeg=0;
for(int j=0; j<n; j++) {
if(matrix[i][j]==1) {
indeg++;
 total_in++; }
if(matrix[j][i]==1) {
outdeg++;}
```

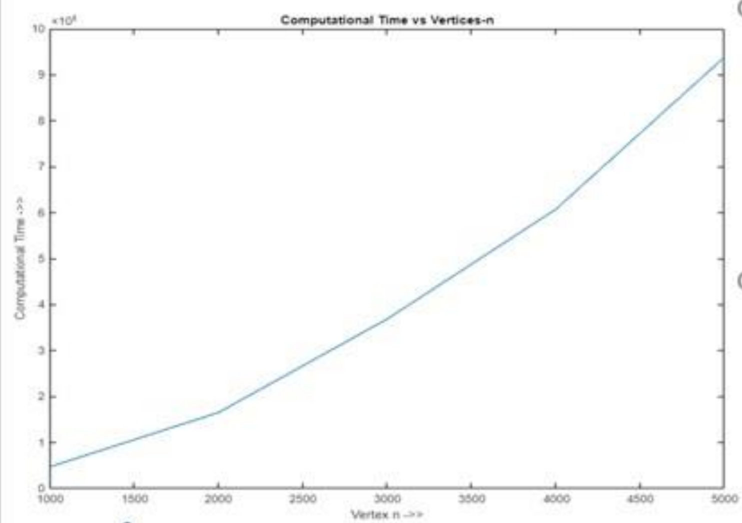
4. Lastly, we printed all the values of In-degree and Out-degree and the computational time in the millisecond and showed that the In-degree and Out-degree are equal. We repeated the same steps for the value of n= 2000, 3000, 4000,5000.

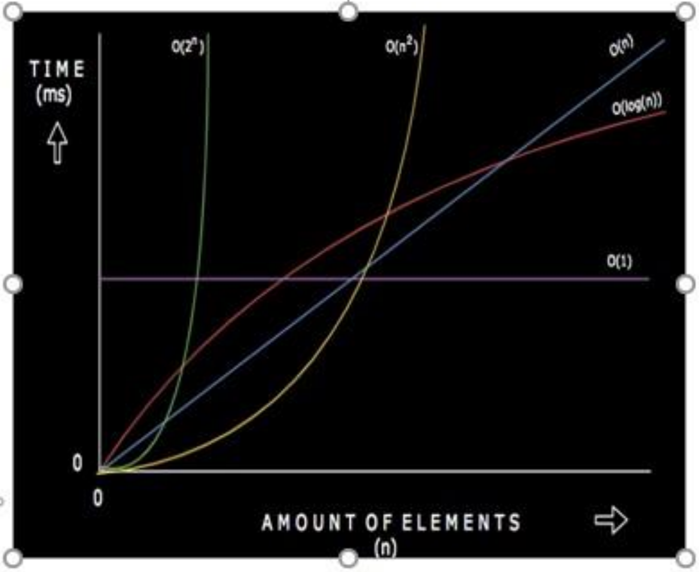
```
printf("\n\n %d \t\t %d \t\t %d",i+1,indeg,outdeg);

 printf("\n");
printf("\n\n Total In degree=%d",total_in);
printf("\n\n Total Out degree=%d",total_out);
```

Computational time vs vertices-n Graph

After finding the computational time when the vertex of n is = 1000, 2000, 3000,4000, 5000 we generate the graph by using MATLAB.

We have determined the approximate time complexity of the program as a function of n by comparing it with the graph of big-O notation.





We can see that the approximate time complexity is Big O(n^2).

Time Complexity

The time complexity of an algorithm is an estimate of the time required by the algorithm to solve a problem of a particular size. The time complexity of an algorithm can be expressed in terms of the number of operations used by the algorithm when the input has a particular size.

According to our mini project instruction, we have drawn a graph of computational time vs vertices-n and compared it with the Big O notation graph. As a result, we found the approximate time complexity of Big O(n2). In theoretical explanation, we used 3 nested loops and some other functions to generate the whole program properly.

```
for(int i=0; i<n; i++)
    {
      for(int j=0; j<n; j++)
      {
         matrix[i][j]=rand()%2;
         if(i==j)
         {
            matrix[i][j]=0;
         }
      }
    }
```

f_1

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

        printf(" %d",matrix[i][j]);
      }
      printf("\n");
    }
```

f_2

```
for(int i=0; i<n; i++)
    {
      indeg=0;
      outdeg=0;
      for(int j=0; j<n; j++)
      {
        if(matrix[i][j]==1)
        {
          indeg++;
          total_in++;
        }
        if(matrix[j][i]==1)
        {
          outdeg++;
          total_out++;
        }
      }
```

f_3

Every time the for loop will be executed for n times. In f_1 part there are two nested for loops. To exit the loop each time one more comparison is needed. For the first nested loop. One more comparison in 2nd for loop.

Number of Comparisons here,

$f_1(n) = (n+1)(n+2) = n^2 + 3n + 2$ And,

For the second nested loop, means the f_2 part,

$$f_2(n) = (n+1)(n+1) = n^2 + 2n + 1$$

In f_3 the for loop will executes n times. As it is a nested loop, In the 2nd for loop two more comparisons will be performed.

So, the number of comparisons in f_3 ,

$f_3 = (n+1)(n+3) = n^2 + 4n + 3$

So, total time complexity is: $f(n) = f_1 + f_2 + f_3$

$$= n^2 + 3n + 2 + n^2 + 2n + 1 + n^2 + 4n + 3$$

$$= 3n^2 + 9n + 5$$

Hence, time complexity of the program is : $O(n) = n^2$

In Big O notation, the time complexity of O(n2) is known as Quadratic time complexity.