**Report of Mini-Project 2**

Section-03

Group-6

**Submitted by:**

|  |  |
| --- | --- |
| Name | Student ID |
| Prinom Mojumder | (2021-2-60-098) |
| Ismail Mahmud Nur | (2021-2-60-052) |
| Tithi Paul | (2021-2-60-057) |

**Submitted to:**

Md. Mozammel Huq Azad Khan,

Ph.D. Professor, Department of Computer Science and Engineering

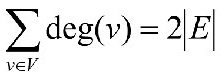
**Introduction:**

An undirected graph consists of a set of nodes and links connecting them. Each node is defined as a vertex, and each link is known as an edge, which links two vertices. The sequence in which the two connected vertices are connected is unimportant. An undirected graph is formed of a finite number of vertices and edges.

Adjacency Matrix is a 2D array of size n x n where n is the number of vertices in a graph.

Handshaking Theorem:

An undirected graph with vertex set V and edge set E. Then



**Code Analysis:**

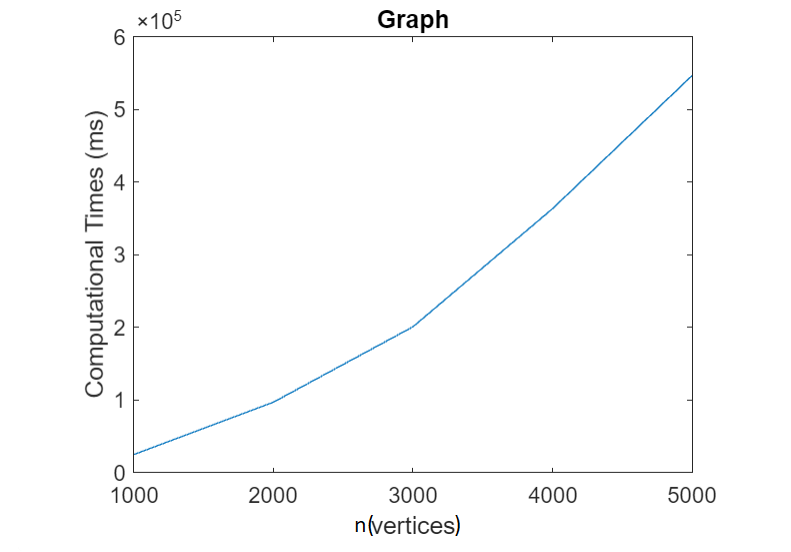
In our mini project-2, we have used some library functions in the code. Then we took an array[5000][5000], some integers, and srand to create a random number. We have also taken a function to measure computational time. “The clock\_t clock()” function from the C library returns the number of clock ticks from starting to the end of the program. The function “Divide CLOCKS\_PER\_SEC by CLOCKS\_PER\_SEC” to get the number of seconds used by the CPU.

We have used 4 nested loops in the program. In the 1st nested for loop, we are indexing the array for random numbers within 0 and 1. Then we used the 2nd nested for loop for printing the array which will allow us to show our matrix output. Through the 3rd nested for loop, we can calculate the edge of the matrix and for the 4th nested for loop, we can calculate the degree of the matrix. Then we can test whether the handshaking theorem works in the matrix, using the formula of the handshaking theorem. For this program, we have converted the computational time from seconds to milliseconds.

**Outputs of the C Code:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| n(vertices) | 1000 | 2000 | 3000 | 4000 | 5000 |
| Computational Times (ms) | 24878 | 97032 | 200957 | 363724 | 547104 |

By using MATLAB, we draw a graph that shows computational time vs n(vertices).



For n = [1000, 2000, 3000, 4000, 5000] and computational time = [24878, 97032, 200957, 363724, 547104] (ms) putting the values of vertex(n) and time in the graph, we can get our desired graph. Seeing the graph, we can assume that it is a (y=x2) curve.

**Time Complexity:**

The time complexity of an algorithm is an approximation of just how much time it would take to solve a task of a specific size. Also, the time complexity of an algorithm may be represented as the number of operations performed by the algorithm when the input is of a certain size.

According to the directions for our project, we created a graph of processing time vs n-vertices and compared it to the Big O notation graph. As an outcome, we determined Big O's estimated time complexity (n2). In the theory, we implemented three nested loops and a couple of extra functions to correctly build the entire program.

For the 1st nested for loop,

f1 = (n+1) (n-i+1)

= n2 - i.n + n + n – i + 1

= n2+2n-i.n-i+1

and for the 2nd nested for loop,

f2= (n+1) (n+1)

= n2 + 2n + 1

In, f3 the outer for loop will execute n time. So, for the 2nd loop, one more comparison will be performed.

Now, f3 = (n+1) (n-i+1+1)

= (n+1) (n-i+2)

= n2 -i.n+2n+n-i + 2

= n2+3n-i.n-i+2

Then for f4 ,the outer for loop will be executed n times. As it is a nested loop, the 2nd for loop will be performed two more comparisons.

f4 = (n+1) (n+1+1+1)

=(n+1) (n+3)

= n2 + 4n + 3

Therefore, the total time complexity is:

f(n ) = f1+f2+f3+f4

= n2 + 2n - i.n – i + 1 + n2 + 2n + 1 + n2+3n - i.n – i +2+ n2 + 4n + 3

= 4n2+12n - 2i.n - 2i+7

Hence, the time complexity of our program is: O(n)=n2

So, we can see that the graph’s time complexity and the program’s time complexity which we determined have been matched.