**Experiment 2.1**

**Student Name: Sahul Kumar Parida UID: 20BCS4919**

**Branch: CSE Section/Group: WM-904/B**

**Semester: 5th Date of Performance: 27/08/2022**

**Subject Name: Design and Analysis Algorithm Lab**

**Subject Code: 20CSP-312**

1. **Aim/Overview of the practical:**

Code and analyze to find an optimal solution to matrix chain multiplication using dynamic programming.

1. **Task to be done/which logistics used:**

Using Dynamic Programming

Dynamic Programming is a method for solving a complex problem by breaking it down into a collection of simpler sub problems, solving each of those sub problems just once, and storing their solutions using a memory-based data structure (array, map, etc).

1. **Algorithm/Flowchart:**

1.Start by placing the parenthesis in all feasible locations, calculating the cost of each placement, and returning the lowest value.

2.Next, you will position the first set of parentheses in n-1 ways in a chain of matrices of size n.

3.When you put a set of parentheses around a problem, you divide it into smaller sub problems.

4.As a result, the problem has an optimal substructure and can be solved quickly using recursion.

5.The least number of n-1 placements required to multiply a chain of size n.

1. **Steps for experiment/practical/Code:**

#include <bits/stdc++.h>

using namespace std;

int MatrixChainOrder(int p[], int i, int j)

{

if (i == j)

return 0;

int k;

int min = INT\_MAX;

int count;

for (k = i; k < j; k++)

{

count = MatrixChainOrder(p, i, k)+ MatrixChainOrder(p, k + 1, j)+ p[i - 1] \* p[k] \* p[j];

if (count < min)

min = count;

}

return min;

}

int main()

{

int arr[] = { 1, 2, 3, 4, 3 };

int n = sizeof(arr) / sizeof(arr[0]);

cout<<"20BCS4919"<<endl;

cout<<"Sahul"<<endl;

cout << "Minimum number of multiplications is "<< MatrixChainOrder(arr, 1, n - 1);

}

1. **Observations/Discussions/ Complexity Analysis:**

If there are n number of matrices we are creating a table contains [(n) (n+1)] / 2 cells that is in worst case total number of cells n\*n = n2 cells we need calculate = **O (n2)**

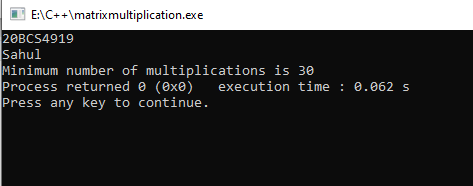
For each one of entry we need find minimum number of multiplications taking worst (it happens at last cell in table) that is Table [1,4] which equals to **O (n)** time.

Finally, **O (n2) \* O (n) = O (n3)**is time complexity.

**Space Complexity**

We are creating a table of n x n so space complexity is **O (n2).**

1. **Output:**



**Learning outcomes (What I have learnt):**

1. Dynamic Programming
2. To implement problems based on different algorithm design techniques.
3. To learn the importance of designing an algorithm in an effective way by considering space and time complexity.