**Program 3 Analysis**

*Kushagr Gupta (kgupta7) Rayan Dasoriya(rdasori)*

Let us assume that the number of matrices is n.

**Naïve Approach:**

When we are following the naïve approach to find the number of scaler multiplications while performing a matrix chain multiplication, we have to multiply the matrices from left to right. If there are two matrices of size m\*n and n\*o, then the number of scaler multiplications will be m\*n\*o. So, if the number of matrices increases, we have to sum up this product for all the subsequent pair. This will lead to traversing all the matrices once. Hence, the run time complexity in naïve approach will be O(n).

**Greedy Approach:**

In Greedy approach, we have to find a greedy solution. This can be found out by multiplying the matrices with the least dimensions. It will help to minimize the cost. In this, we will be making a recursive call to the matrices on the left and right of the smallest found matrices and then we will be summing it up. This approach is similar to Quick sort. The worst-case scenario in each step might that one of the recursive calls end up in the extreme end of the matrices and the other end up being empty. In such cases, the complexity will be O(n) + O(n-1) + … + O(1), which is equivalent to summing up n elements, i.e., O(n2).

**Dynamic Approach:**

In the bottom-up dynamic programming approach, we are initializing the array to store the results. Since the size of the array is n\*n, the space complexity turns out to be O(n2).

Let M[i,j] denotes the minimum number of scaler multiplications required to compute the matrix *Ai* x *Ai+1* x *Aj*. Our goal is to find M[1][n]. We will initialize M[i][i] = 0. Then, we will look for different possible ways to split *Ai* through *Aj* into two parts. We will compare the cost of all such splits by adding up the least cost of the product of the two parts and the cost of multiplying the two products. The recursive formula can be given by:

*M(i,j) = mink(M(i,k) + M(k+1,j) + di-1dkdj )*

The subproblem can be solved by going along the diagonal and starting just above it. The end/goal will be the upper right corner.

The code will consist of two for loops to traverse through each element of the array M and one more for loop which will help us to find the multiplications of all the possible subsequence. This will give us the run time complexity of the program to be O(n3).

The run time of naïve, greedy and dynamic approach for matrix chain multiplication for input-5.txt in MacBook-Pro 2.3 GHz Intel Core i5 is demonstrated below:

A picture containing indoor

Description automatically generated