

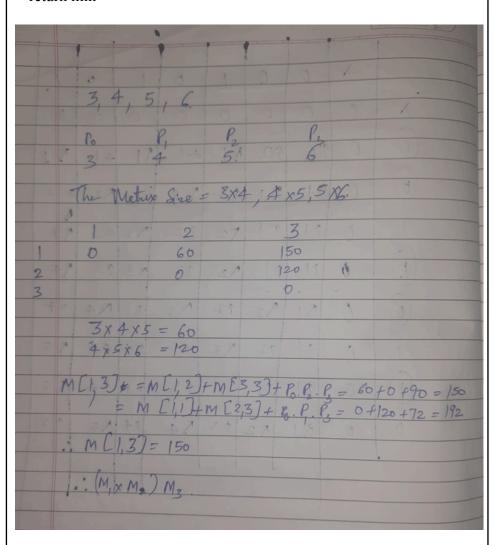
Bharatiya Vidya Bhavan's SARDAR PATEL INSTITUTE OF TECHNOLOGY

(Autonomous Institute Affiliated to University of Mumbai) Munshi Nagar, Andheri (W), Mumbai – 400 058. Department of Master of Computer Application

Experiment	3
Aim	To understand and implement Dynamic Programming Approach
Objective	Write Pseudocode for given problems and understanding the implementation of Dynamic Programming Solve Matrix Multiplication Problem using Dynamic Programming Calculating time complexity of the given problems
Name	Vaibhav Agarwal
UCID	2023510001
Class	FYMCA
Batch	D

Algorithm and	Pseudocode for Matrix Chaining Multiplication
Explanation of	MatrixChainOrder(p, i, j)
the technique	if $i == j$ return 0
used	
	min = INFINITY
	for k from i to j-1:
	count = MatrixChainOrder(p, i, k) + MatrixChainOrder(p, k + 1, j)
	+ p[i - 1] * p[k] * p[j]
	if count < min:
	min = count

return min



Time Complexity

The time complexity for Matrix Chaining Multiplication is $O(n^3)$.

```
Program(Code
```

```
public static void main(String args[])
                      int arr[] = new int[] {3, 4, 5,6};
                      int N = arr.length;
                      System.out.println("Minimum number of multiplications is
                  + MatrixChainOrder(arr, 1, N - 1));
Output
                  Minimum number of multiplications is 150
Justification of
                 Subproblem Time complexity - O(n^2)
                 Within each subproblem, the algorithm involves a loop that iterates over
the complexity
calculated
                 the possible positions to split the subchain. This loop runs in O(n) time.
                 For each subproblem, the algorithm calculates the cost of multiplying
                 matrices at each possible split point.
                 Multiplying these factors together, we get the overall time complexity:
                 O(n^2) * O(n) * O(1) = O(n^3) Therefore, the time complexity for
                 Matrix Chaining Multiplication is O(n^3)
Conclusion
                 Advantages:
                 Using dynamic programming we can store the solutions to these sub
                 problems so that it does not use extra space for the same sub problem.
                 Applications:
                 Optimization - Utilized in compilers to optimize code generation and
                 execution, particularly for mathematical operations, resulting in faster
                 and more efficient programs.
                 Data Compression - Used in various compression algorithms to
                 efficiently encode and decode data, reducing storage requirements and
                 transmission times.
```

Computer Graphics - Used for transformations like scaling, rotation, and

translation of images and objects in computer graphics rendering

engines.