**Batch: D2**

**Roll No.: 16010122323**

**Experiment No. 07**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**



**Title: Implementation Matrix Chain Multiplication of Dynamic Programming**

**Objective:** To learn Matrix chain multiplication using Dynamic Programming Approach

|  |  |
| --- | --- |
| **CO to be**  CO 2 | **achieved:**  Describe various algorithm design strategies to solve different problems and analyse  Complexity. |



## Books/ Journals/ Websites referred:

1. **Ellis horowitz, Sarataj Sahni, S.Rajsekaran,” Fundamentals of computer algorithm”, University Press**

## T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein,” Introduction to algortihtms”,2nd Edition ,MIT press/McGraw Hill,2001

1. [**http://www.lsi.upc.edu/~mjserna/docencia/algofib/P07/dynprog.pdf**](http://www.lsi.upc.edu/~mjserna/docencia/algofib/P07/dynprog.pdf)
2. [**http://www.geeksforgeeks.org/travelling-salesman-problem-set-1/**](http://www.geeksforgeeks.org/travelling-salesman-problem-set-1/)
3. [**http://www.mafy.lut.fi/study/DiscreteOpt/tspdp.pdf**](http://www.mafy.lut.fi/study/DiscreteOpt/tspdp.pdf)
4. [**https://class.coursera.org/algo2-2012-001/lecture/181**](https://class.coursera.org/algo2-2012-001/lecture/181)
5. [**http://www.quora.com/Algorithms/How-do-I-solve-the-travelling-salesman-**](http://www.quora.com/Algorithms/How-do-I-solve-the-travelling-salesman-problem-using-Dynamic-programming)[**problem-using-Dynamic-programming**](http://www.quora.com/Algorithms/How-do-I-solve-the-travelling-salesman-problem-using-Dynamic-programming)

## [www.cse.hcmut.edu.vn/~dtanh/download/Appendix\_B\_2.ppt](http://www.cse.hcmut.edu.vn/~dtanh/download/Appendix_B_2.ppt)

1. [**www.ms.unimelb.edu.au/~s620261/powerpoint/chapter9\_4.ppt**](http://www.ms.unimelb.edu.au/~s620261/powerpoint/chapter9_4.ppt)

## Pre Lab/ Prior Concepts:

Data structures, Concepts of algorithm analysis

Page **1** of **9** AOA Sem IV Jan-May 2022

## Historical Profile:

Dynamic Programming (DP) is used heavily in optimization problems (finding the maximum and the minimum of something). Applications range from financial models and operation research to biology and basic algorithm research. So the good news is that understanding DP is profitable. However, the bad news is that DP is not an algorithm or a data structure that you

can memorize. It is a powerful algorithmic design technique.

## New Concepts to be learned:

Application of algorithmic design strategy to any problem, dynamic Programming method of problem-solving Vs other methods of problem solving, optimality of the solution, Optimal Binary Search Tree Problems and their applications

## Theory:

**Problem definition:**

Given a sequence of N matrices, the matrix chain multiplication problem is to find the most efficient way to [multiply these matrices](https://en.wikipedia.org/wiki/Matrix_multiplication) by minimizing the number of computations involved during multiplications.

**Optimal Substructure:** parenthesization/ select the subgroup of matrices that will result in least number of computations.

For multiplication of matrix series Ai to Aj, choose Ak such that multiplication of matrices through Ai..k and Ak+1…j will incur least number of computations for any k such that i<=k<j.

## Recursive Formula:



**Implementations and output :**#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

#include <math.h> // Include math.h for fmin function

int \*\*dp;

int matrixChain(int p[], int i, int j) {

    if (i == j) {

        return 0;

    }

    if (dp[i][j] != -1) {

        return dp[i][j];

    }

    dp[i][j] = INT\_MAX;

    for (int k = i; k < j; k++) {

        dp[i][j] = fmin(

                dp[i][j], matrixChain(p, i, k)

                        + matrixChain(p, k + 1, j)

                        + p[i - 1] \* p[k] \* p[j]);

    }

    return dp[i][j];

}

int matrixChainOrder(int p[], int n) {

    dp = (int \*\*)malloc(n \* sizeof(int \*));

    for (int i = 0; i < n; i++) {

        dp[i] = (int \*)malloc(n \* sizeof(int));

        for (int j = 0; j < n; j++) {

            dp[i][j] = -1;

        }

    }

    int i = 1, j = n - 1;

    return matrixChain(p, i, j);

}

int main() {

    printf("Enter the number of matrices: ");

    int N;

    scanf("%d", &N);

    int arr[N + 1];

    printf("Enter the dimensions of matrices: ");

    for (int i = 0; i < N + 1; i++) {

        scanf("%d", &arr[i]);

    }

    printf("Minimum number of multiplications is %d\n", matrixChainOrder(arr, N + 1));

    // Free dynamically allocated memory

    for (int i = 0; i < N + 1; i++) {

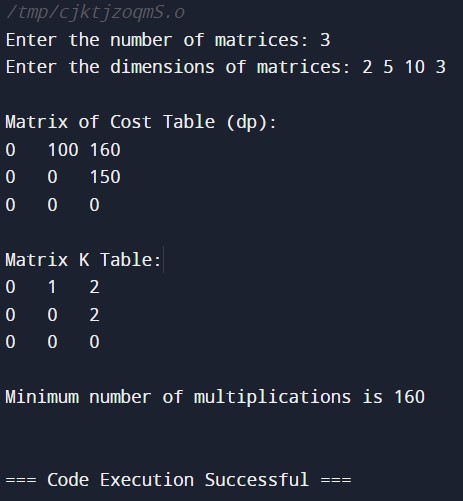
        free(dp[i]);

    }

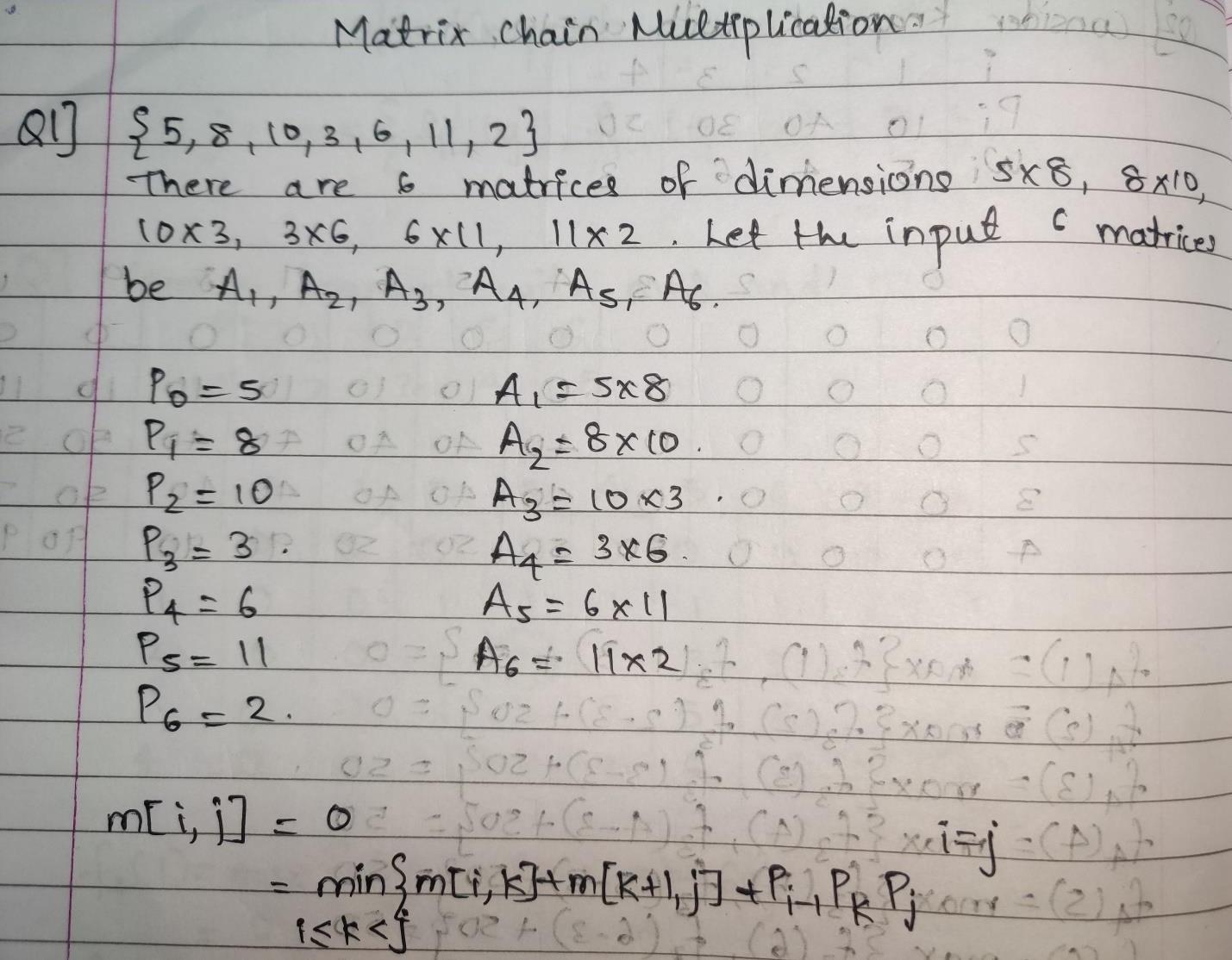
    free(dp);

    return 0;

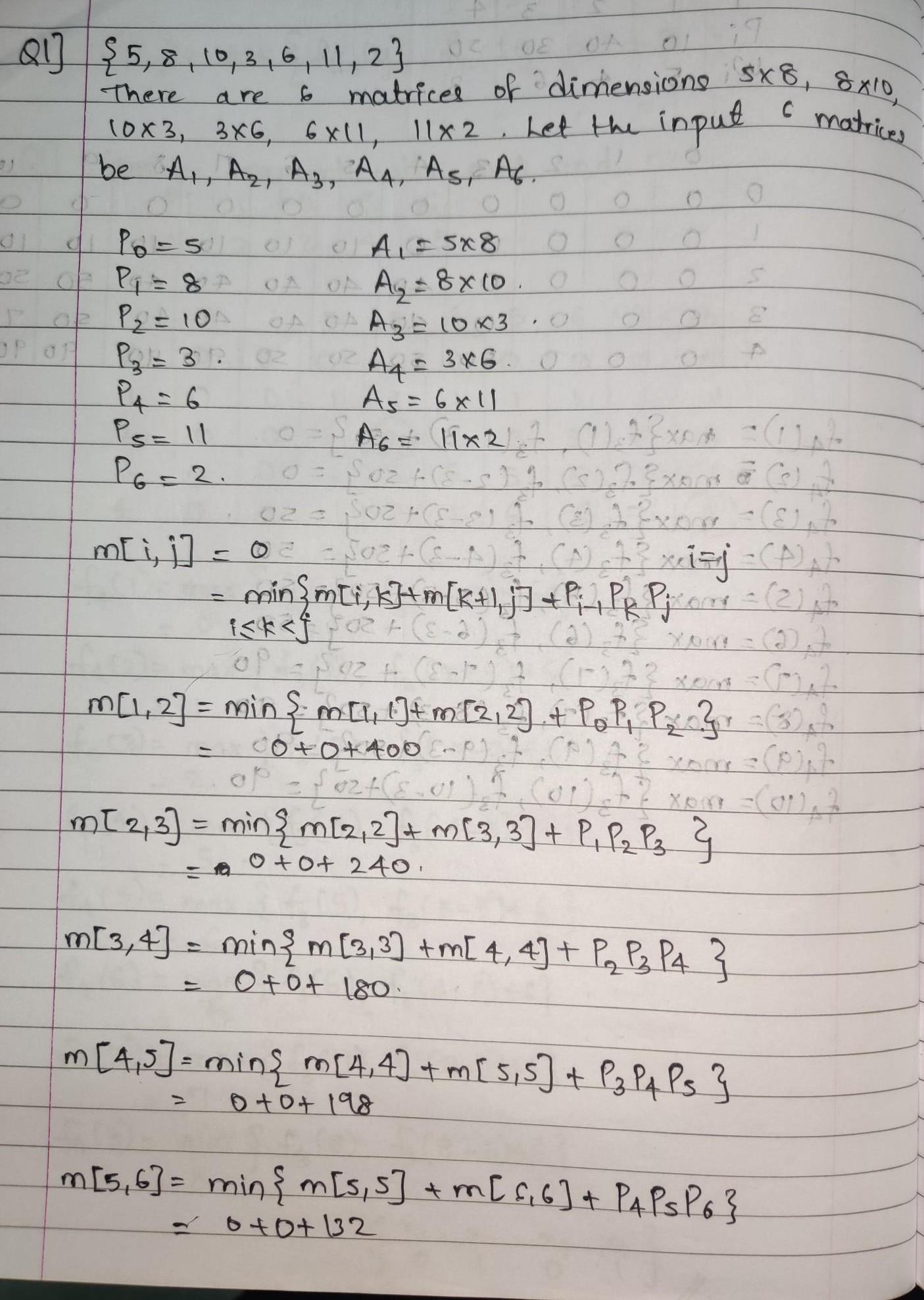
}

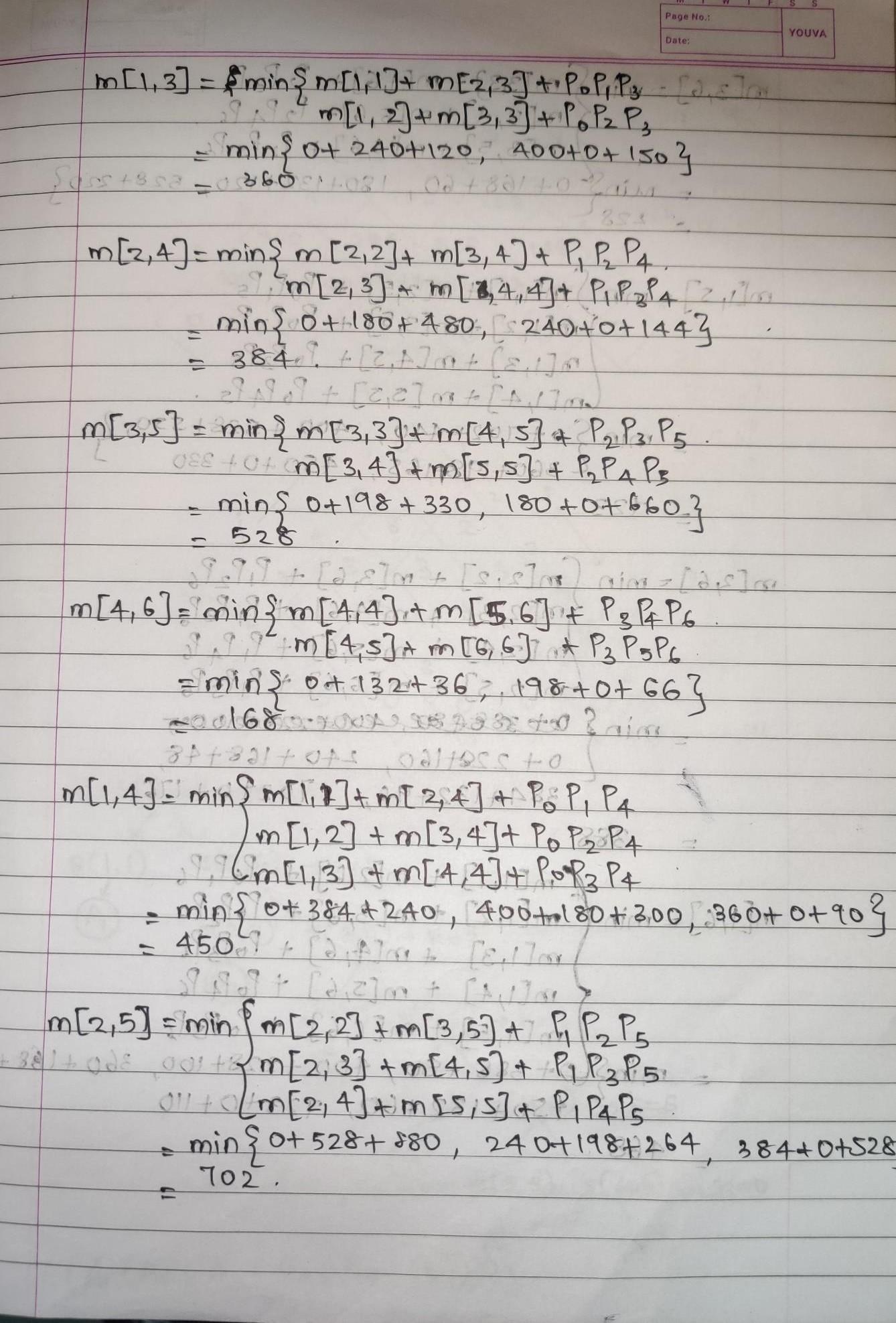


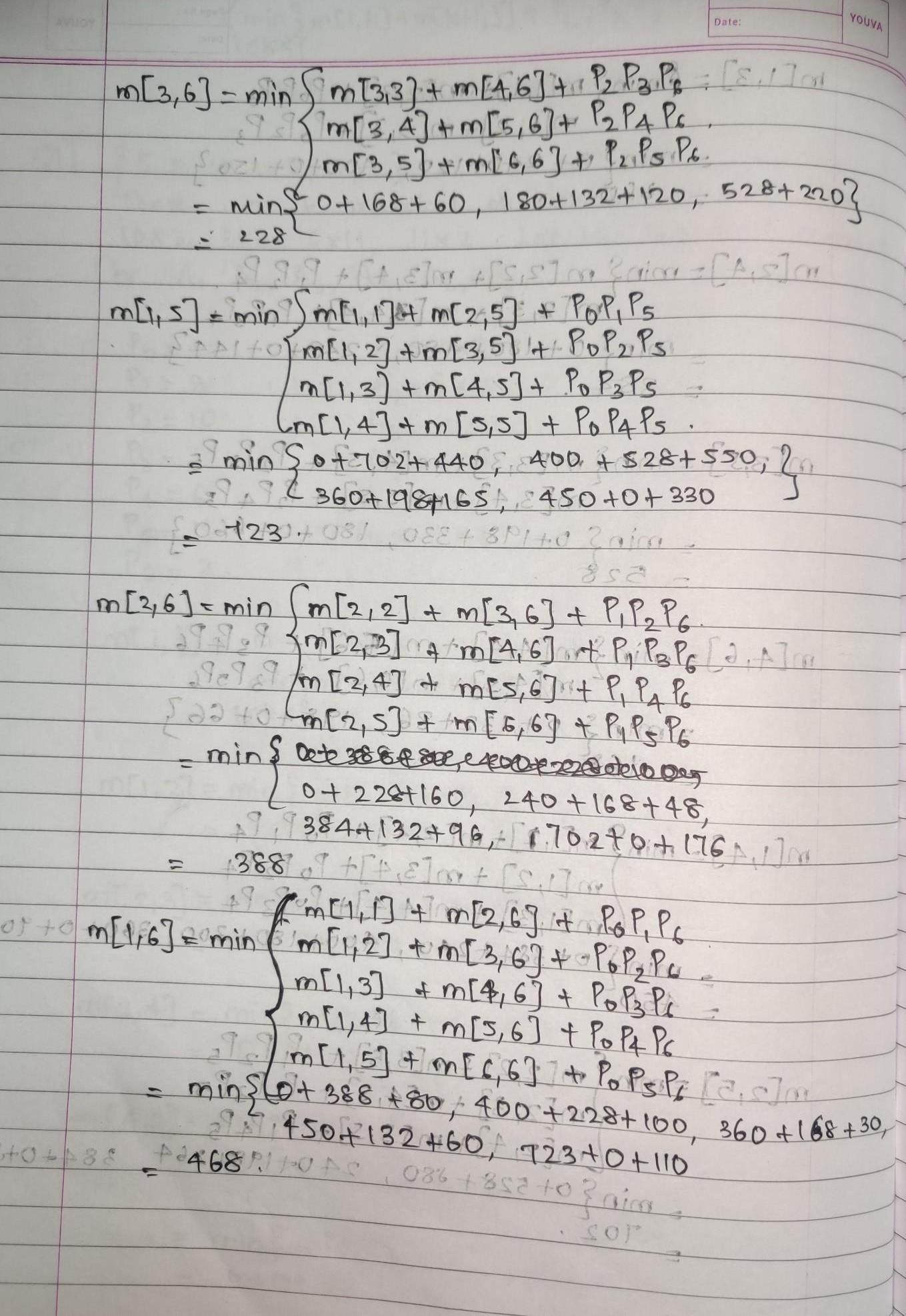
## Example:

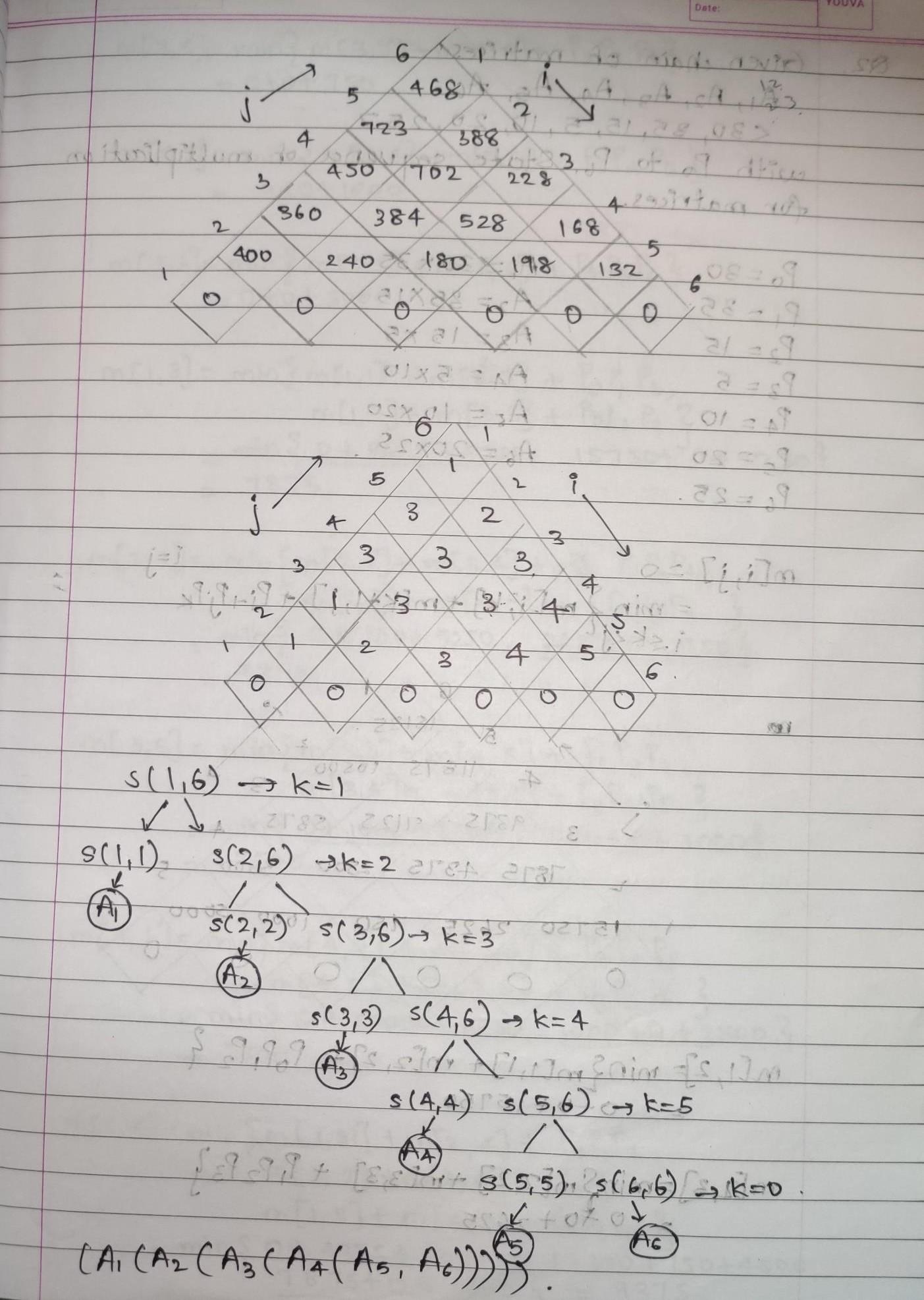


## Solution for the example:









## Analysis of algorithm:

**Total Complexity is: O (n3)**

There are three nested loops. Each loop executes a maximum n times.

1. l, length, O (n) iterations.
2. i, start, O (n) iterations.
3. k, split point, O (n) iterations Body of loop constant complexity **Space Complexity:**

O(n\*n) where n is the number present in the chain of the matrices. We create a DP matrix that stores the results after each operation.

**Code:**

import java.util.\*;

public class Main {

  static int[][] dp = new int[100][100];

  // Function for matrix chain multiplication

  static int matrixChainMemoised(int[] p, int i, int j)

  {

    if (i == j)

    {

      return 0;

    }

    if (dp[i][j] != -1)

    {

      return dp[i][j];

    }

    dp[i][j] = Integer.MAX\_VALUE;

    for (int k = i; k < j; k++)

    {

      dp[i][j] = Math.min(

        dp[i][j], matrixChainMemoised(p, i, k)

        + matrixChainMemoised(p, k + 1, j)

        + p[i - 1] \* p[k] \* p[j]);

    }

    return dp[i][j];

  }

  static int MatrixChainOrder(int[] p, int n)

  {

    int i = 1, j = n - 1;

    return matrixChainMemoised(p, i, j);

  }

  // Driver Code

  public static void main (String[] args)

  {

    Scanner sc=new Scanner(System.in);

    int N=sc.nextInt();

    int arr[];

    arr=new int[N];

    for(int i=0; i<N; i++){

        arr[i]=sc.nextInt();

    }

    // int arr[] = { 1, 2, 3, 4 };

    // int N= arr.length;

    for (int[] row : dp)

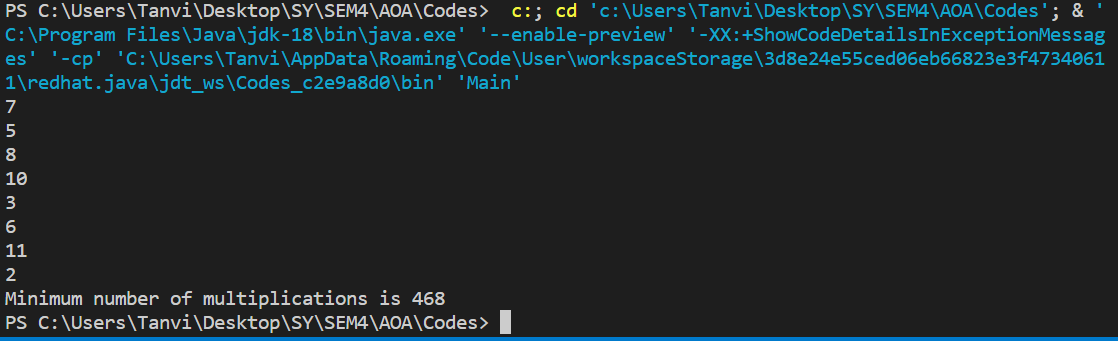
      Arrays.fill(row, -1);

    System.out.println("Minimum number of multiplications is " + MatrixChainOrder(arr, N));

  }

}

**Output**



## Conclusion:

In this experiment, we have learnt Implementation of Matrix Chain Multiplication by dynamic programming approach. We have also understood the algorithm and implemented the same on java. Additionally, we have compared the time complexity of the program.