K. J. Somaiya College of Engineering

(A Constituent College of Somaiya Vidyavihar University)

Batch:- B-2	**Roll No:-** 16010122151

Experiment No. 06

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 achieved:

CO 2 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**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein," Introduction to algortihtms",2nd Edition ,MIT press/McGraw Hill,2001**
3. **http://www.lsi.upc.edu/~mjserna/docencia/algofib/P07/dynprog.pdf**
4. **http://www.geeksforgeeks.org/travelling-salesman-problem-set-1/**
5. **http://www.mafy.lut.fi/study/DiscreteOpt/tspdp.pdf**
6. **https://class.coursera.org/algo2-2012-001/lecture/181**
7. **http://www.quora.com/Algorithms/How-do-I-solve-the-travelling-salesman-problem-using-Dynamic-programming**
8. **www.cse.hcmut.edu.vn/~dtanh/download/Appendix_B_2.ppt**
9. **www.ms.unimelb.edu.au/~s620261/powerpoint/chapter9_4.ppt**

Pre Lab/ Prior Concepts:

Data structures, Concepts of algorithm analysis

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 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:

$$m[i,j] = \begin{cases} 0 & i = j, \\ \min_{i \le k < j} (m[i,k] + m[k+1,j] + p_{i-1} p_k p_j) & i < j \end{cases}$$

Code:-

```c
#include <stdio.h>
#include <stdlib.h>

int minimum_cost(int matrix[], int t) {
    int x, small;
    small = matrix[0];
    for (x = 1; x < t; x++) {
        if (matrix[x] < small)
            small = matrix[x];
    }
    return small;
}

int main() {
    int limit, i, j, k, l, t;
    int matrix[30], multiplier[10][10] = {0}, columns[10], rows[10], temp[10];

    printf("\nEnter Total Number of Matrices: ");
    scanf("%d", &limit);

    // Input matrix dimensions and validate
    for (i = 0; i < limit; i++) {
        printf("\nEnter Number of Rows of Matrix %d: ", i + 1);
        scanf("%d", &rows[i]);

        if (i > 0 && rows[i] != columns[i - 1]) {
            printf("Error: Number of rows of Matrix %d does not match the number of
columns of Matrix %d.\n", i + 1, i);
            return 1; // Exit with error code
        }

        printf("Enter Number of Columns of Matrix %d: ", i + 1);
        scanf("%d", &columns[i]);
    }

    printf("\n\n\n");

    // Initialize temporary array
    for (i = 0; i < limit; i++)
        temp[i] = rows[i];
    temp[i] = columns[i - 1];
```

```
    // Calculate minimum multiplications
    for (l = 2; l <= limit; l++) {
        for (i = 1; i <= limit - l + 1; i++) {
            j = i + l - 1;
            t = 0;
            printf("\nCalculating multipliers for matrices %d to %d:\n", i, j);
            for (k = i; k < j; k++) {
                matrix[t] = (multiplier[i][k] + multiplier[k + 1][j]) + (temp[i - 1]
* temp[k] * temp[j]);
                printf("Multiplier[%d][%d] + Multiplier[%d][%d] + (%d * %d * %d) =
%d\n",
                       i, k, k + 1, j, temp[i - 1], temp[k], temp[j], matrix[t]);
                t++;
            }
            multiplier[i][j] = minimum_cost(matrix, t);
            printf("Minimum cost for matrices %d to %d: %d\n", i, j,
multiplier[i][j]);
        }
    }

    printf("\nMinimum Multiplications Needed: %d", multiplier[1][limit]);

    return 0;
}
```

Output:-

```
Enter Total Number of Matrices: 4

Enter Number of Rows of Matrix 1: 5
Enter Number of Columns of Matrix 1: 4

Enter Number of Rows of Matrix 2: 4
Enter Number of Columns of Matrix 2: 6

Enter Number of Rows of Matrix 3: 6
Enter Number of Columns of Matrix 3: 2

Enter Number of Rows of Matrix 4: 2
Enter Number of Columns of Matrix 4: 7
```

```
Calculating multipliers for matrices 1 to 2:
Multiplier[1][1] + Multiplier[2][2] + (5 * 4 * 6) = 120
Minimum cost for matrices 1 to 2: 120

Calculating multipliers for matrices 2 to 3:
Multiplier[2][2] + Multiplier[3][3] + (4 * 6 * 2) = 48
Minimum cost for matrices 2 to 3: 48

Calculating multipliers for matrices 3 to 4:
Multiplier[3][3] + Multiplier[4][4] + (6 * 2 * 7) = 84
Minimum cost for matrices 3 to 4: 84

Calculating multipliers for matrices 1 to 3:
Multiplier[1][1] + Multiplier[2][3] + (5 * 4 * 2) = 88
Multiplier[1][2] + Multiplier[3][3] + (5 * 6 * 2) = 180
Minimum cost for matrices 1 to 3: 88
```

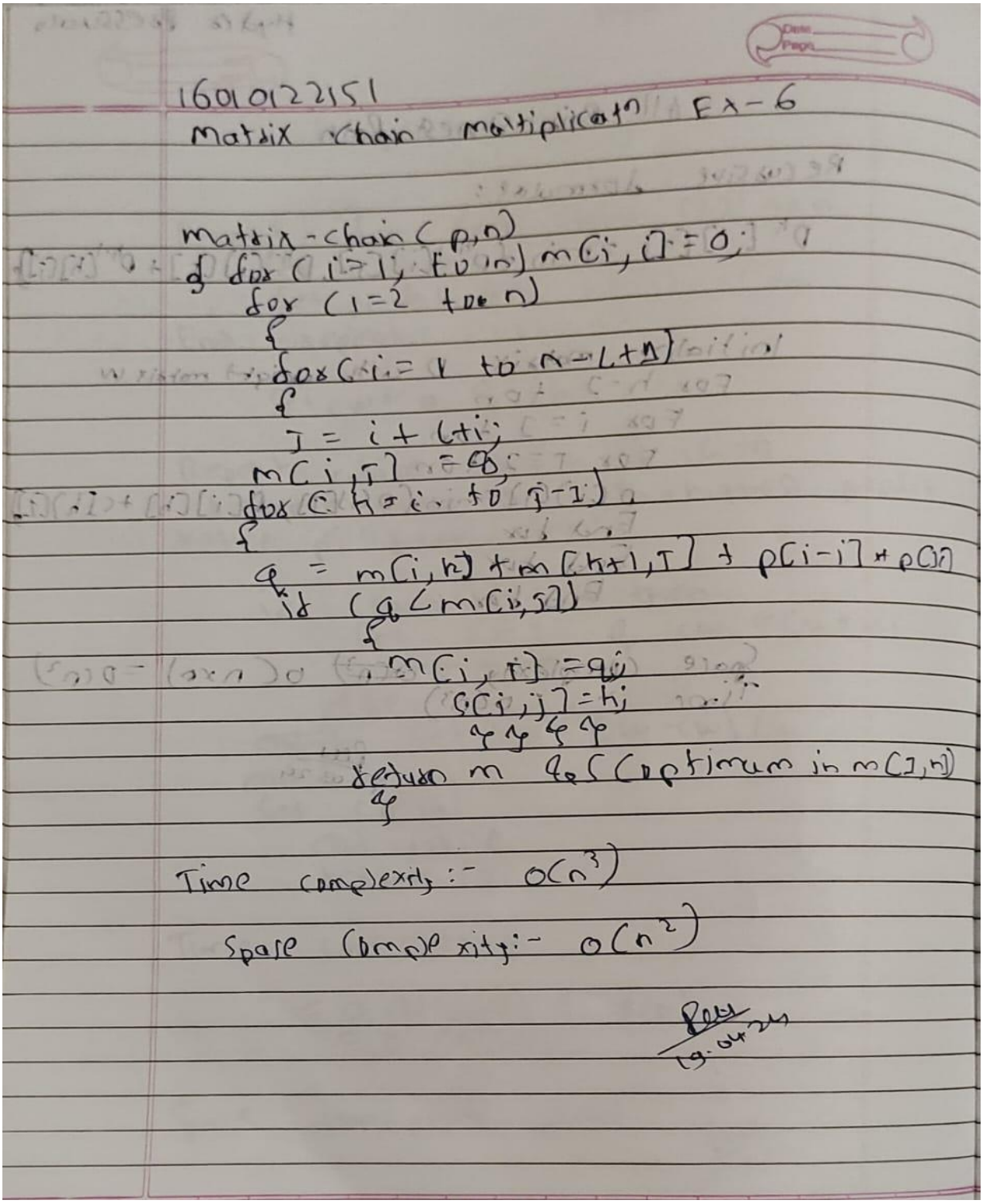
```
Calculating multipliers for matrices 2 to 4:
Multiplier[2][2] + Multiplier[3][4] + (4 * 6 * 7) = 252
Multiplier[2][3] + Multiplier[4][4] + (4 * 2 * 7) = 104
Minimum cost for matrices 2 to 4: 104

Calculating multipliers for matrices 1 to 4:
Multiplier[1][1] + Multiplier[2][4] + (5 * 4 * 7) = 244
Multiplier[1][2] + Multiplier[3][4] + (5 * 6 * 7) = 414
Multiplier[1][3] + Multiplier[4][4] + (5 * 2 * 7) = 158
Minimum cost for matrices 1 to 4: 158

Minimum Multiplications Needed: 158

=== Code Execution Successful ===
```

Algorithm:-



16010122151
Matrix chain multiplication Ex-6

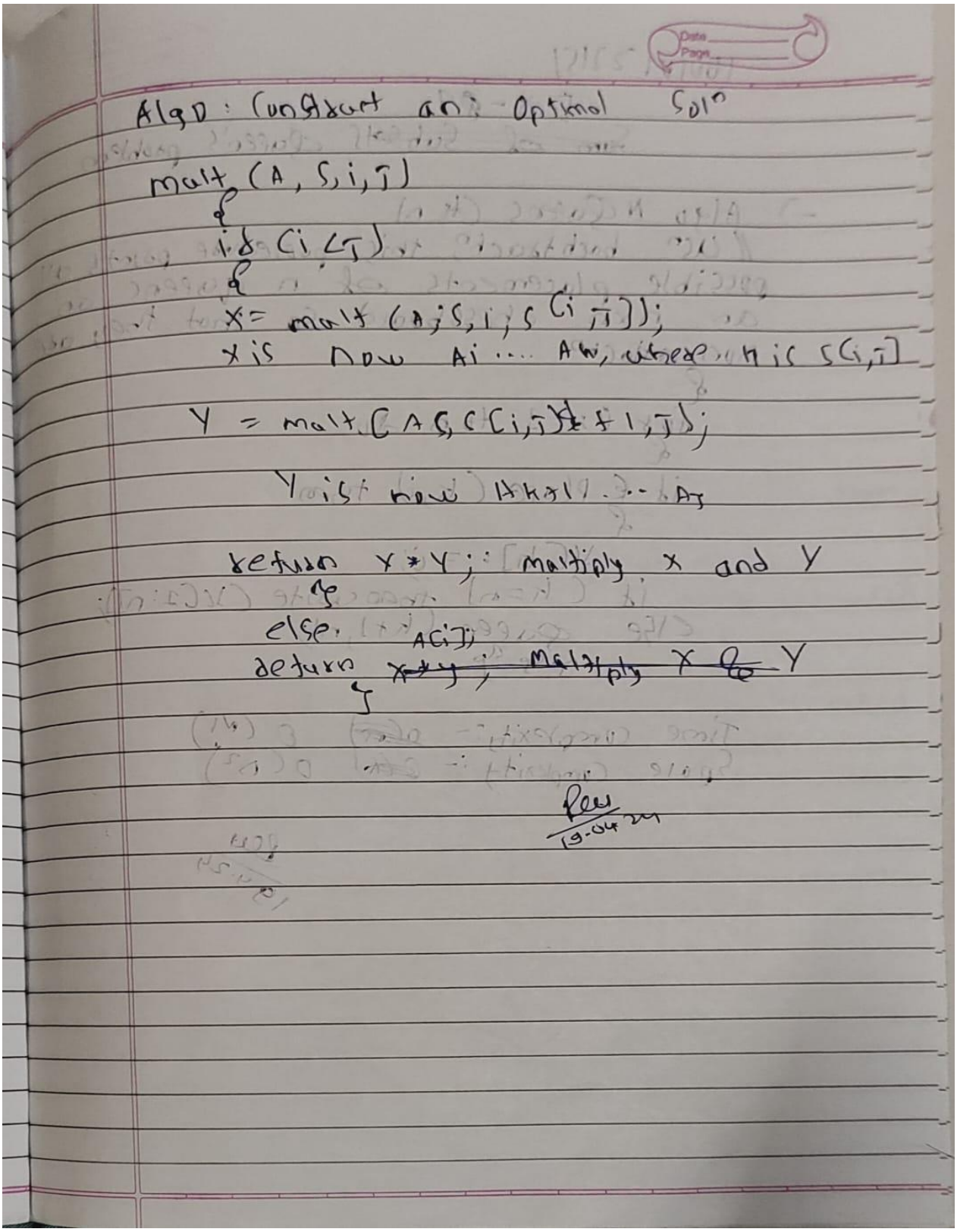
```
matrix-chain(p,n)
{ for(i=1; i<n; i++) m[i,i]=0;
  for(l=2 to n)
  {
    for(i=1 to n-l+1)
    {
      j = i+l-1;
      m[i,j] = ∞;
      for(k=i; k<j; k++)
      {
        q = m[i,k] + m[k+1,j] + p[i-1]*p[k]*p[j];
        if(q<m[i,j])
        {
          m[i,j] = q;
          s[i,j] = k;
        }
      }
    }
  }
  return m and s(Optimum in m[1,n])
}
```

Time complexity :- $O(n^3)$

Space complexity :- $O(n^2)$

K. J. Somaiya College of Engineering

(A Constituent College of Somaiya Vidyavihar University)



Algo: Construct an Optimal Soln

```
Mult (A, S, i, j)
{
if (i < j)
{
X = Mult (A, S, i, S[i, j]);
X is Now Ai ... Ak, where k is S[i, j]

Y = Mult (A, S, S[i, j] + 1, j);

Y is now Ak+1 ... Aj

return X * Y; : Multiply X and Y
}
else
return A[i];
}
```

Rev
19-04-22

Analysis of algorithm:

Total Complexity is: O (n^3)

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.

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 c language. Additionally, we have compared the time complexity of the program.