NAME:	Eshan Bhuse
UID:	2021300013
SUBJECT	Design and analysis of algorithm
EXPERIMEN TNO:	4
AIM:	**Experiment to implement matrix chain multiplication.**
ALGORITHM:	**MATRIX-CHAIN-ORDER (p)** 1. n length[p]-1 2. for i ← 1 to n 3. do m [i, i] ← 0 4. for l ← 2 to n // l is the chain length 5. do for i ← 1 to n-l + 1 6. do j ← i+ l -1 7. m[i,j] ← ∞ 8. for k ← i to j-1 9. do q ← m [i, k] + m [k + 1, j] + pi-1 pk pj 10. If q < m [i,j] 11. then m [i,j] ← q 12. s [i,j] ← k 13. return m and s. **PRINT-OPTIMAL-PARENS (s, i, j)** 1. if i=j 2. then print "A" 3. else print "(" 4. PRINT-OPTIMAL-PARENS (s, i, s [i, j]) 5. PRINT-OPTIMAL-PARENS (s, s [i, j] + 1, j) 6. print ")"

THEORY:

MATRIX CHAIN MULTIPLICATION

Dynamic Programming is a technique in computer programming that helps to efficiently solve a class of
problems that have overlapping sub-problems and optimal substructure property. If any problem can be divided into
sub-problems, which in turn are divided into smaller sub-problems, and if there are overlapping among these sub-
problems, then the solutions to these sub-problems can be saved for future reference. The approach of solving
problems using dynamic programming algorithm has following steps:

1. Characterize the structure of an optimal solution.
2. Recursively define the value of an optimal solution.
3. Compute the value of an optimal solution, typically in a bottom-up fashion.
4. Construct an optimal solution from computed information.

Matrix chain multiplication (or the **matrix chain ordering problem**) is an optimization problem concerning the most efficient way to multiply a given sequence of matrices. The problem is not actually to perform the multiplications, but merely to decide the sequence of the matrix multiplications involved. The problem may be solved using dynamic programming.

The aim of this experiment is two-fold. First, it finds the efficient way of multiplying a sequence of k matrices (called Matrix Chain Multiplication) using Dynamic Programming. The chain of multiplication M1 x M2 x M3 x M4 x...x Mk may be computed in $(2N)!/((N+1)!\,N!) = (2N/N)\ /\ (N+1)$ ways due to associative property where $N = k - 1$ of matrix multiplication. Second, it compares regular matrix multiplication which has complexity of (n^3) and Strassen's Matrix Multiplication which has complexity of $(n^{2.81})$ using Divide and Conquer.

Consider the optimization problem of efficiently multiplying a randomly generated sequence of 10 matrices (M1, M2, M3, M4 ,..., M10) using Dynamic programming approach. The dimension of these matrices are stored in an array p[i] for i = 0 to 9, where the dimension of the matrix Mi is (p[i-1] x p[i]). All p[i] are randomly generated and they are between 15 and 46. For example, [0. .10] = (23, 20, 25, 45, 30, 35, 40, 22, 15, 29, 21). All ten matrices are generated randomly and each matrix value can be between 0 and 1. Determine following values of Matrix Chain Multiplication (MCM) using Dynamic Programming:

1) m[1..10][1..10] = Two dimension matrix of optimal solutions (No. of multiplications) of all possible matrices M1... M10

2) c[1..9][2..10] = Two dimension matrix of optimal solutions (parenthesizations) of all combinations of matrices M1...M10

3) the optimal solution (i.e.parenthesization) for the multiplication of all ten matrices M1x M2x M3xM4 x...x M10

Find the running time of 10 matrices using regular matrix multiplication and Strassen’s Matrix Multiplication as a trivial sequence i.e. (((((((((M1x M2)x M3) xM4 x...x M10) and the sequence of matrix multiplication suggested by Matrix Chain Multiplication in Step No. 3

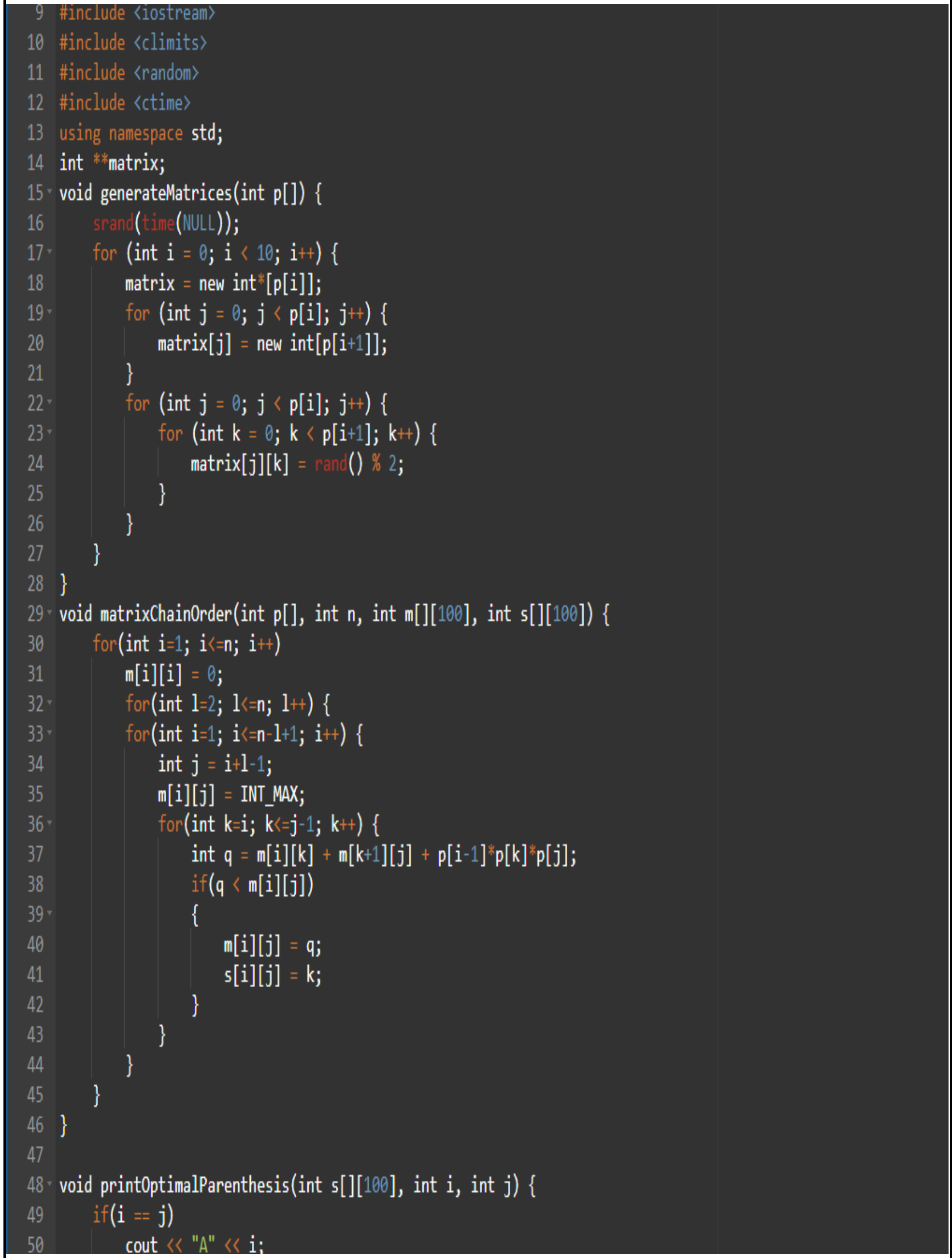
Input :

All p[i] for i=0 to 9 are randomly generated and they are between 15 and 46. 2) All ten matrices are generated randomly and each matrix value can be between 0 and 1.

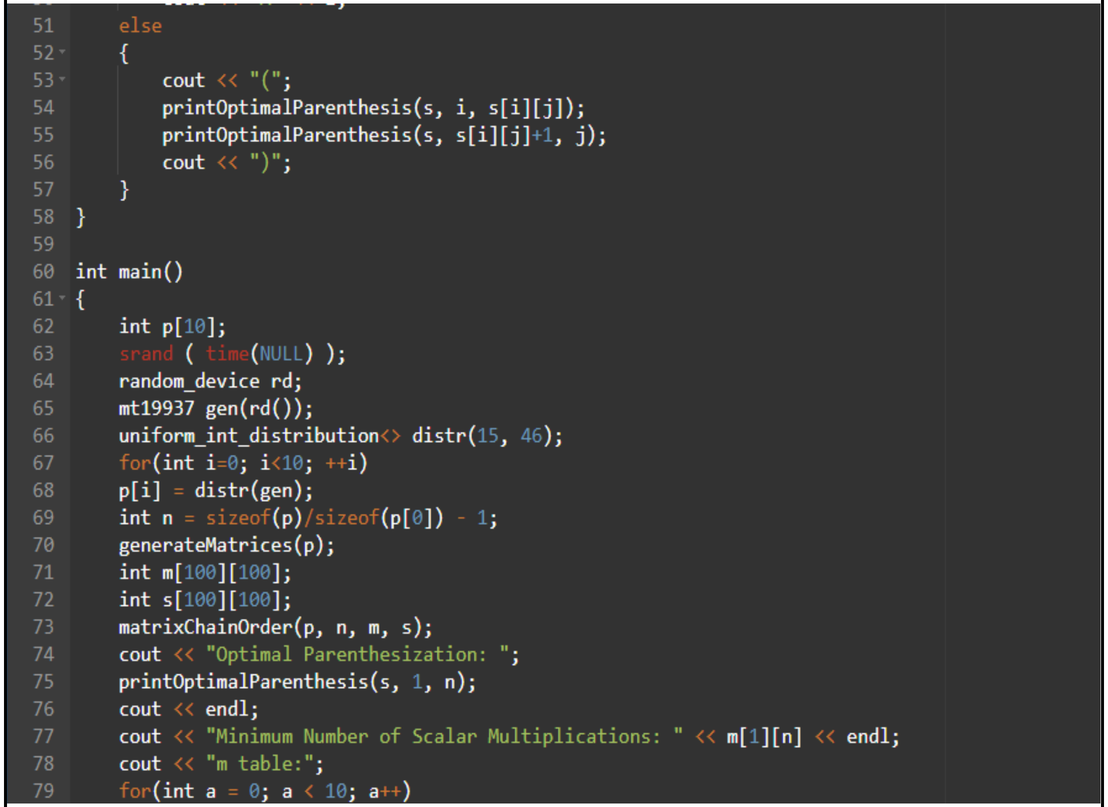
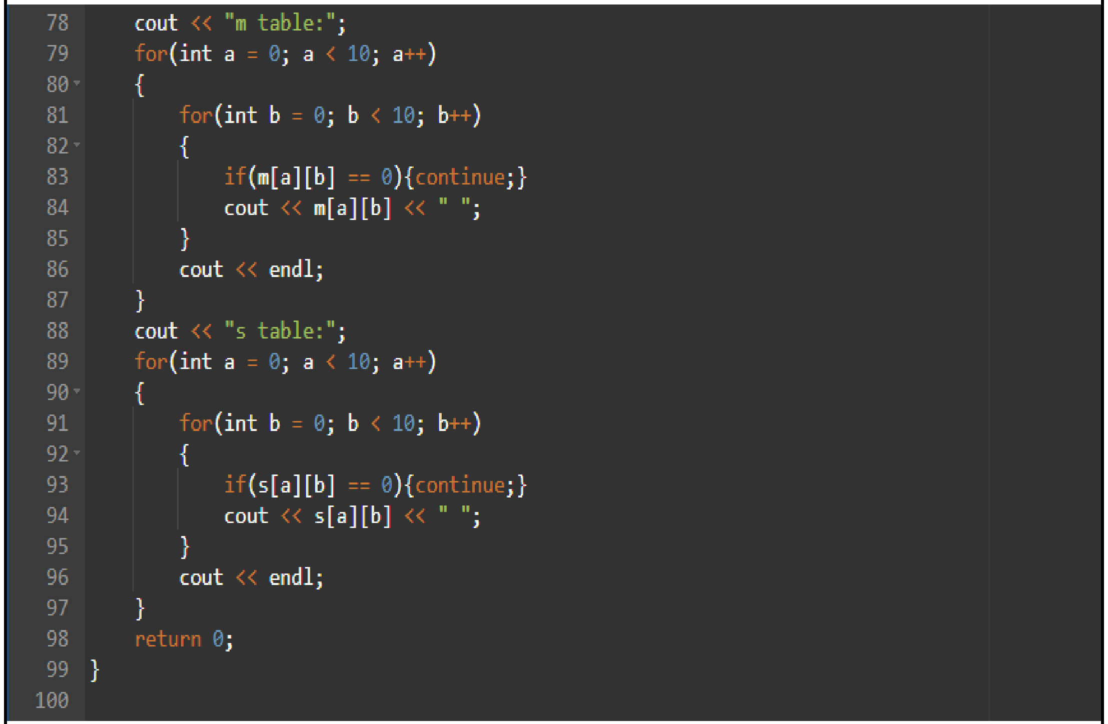
Output:

1) m[1..10][1..10] = Two dimension matrix of optimal solutions (No. of multiplications) of all possible matrices M1... M10
2) c[1..9][2..10] = Two dimension matrix of optimal solutions (parenthesizations) of all combinations of matrices M1...M10
3) The optimal solution (i.e.parenthesization) for the multiplication of all ten matrices M1x M2x M3xM4 x...x M10 4) Print the time required to multiply ten matrices using four combinations as discussed above.

PROGRAM:

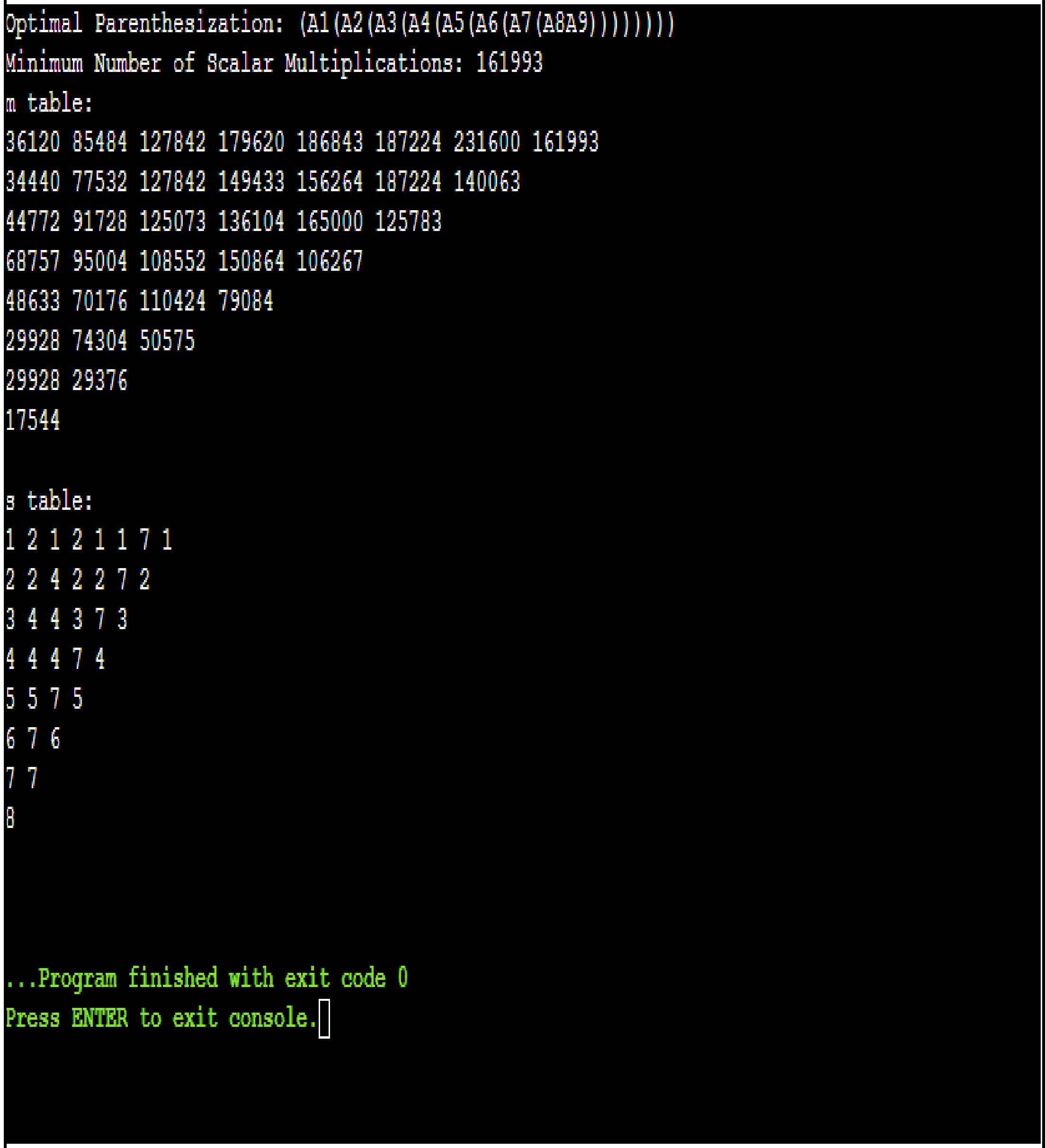


```
#include <iostream>
#include <climits>
#include <random>
#include <ctime>
using namespace std;
int **matrix;
void generateMatrices(int p[]) {
    srand(time(NULL));
    for (int i = 0; i < 10; i++) {
        matrix = new int*[p[i]];
        for (int j = 0; j < p[i]; j++) {
            matrix[j] = new int[p[i+1]];
        }
        for (int j = 0; j < p[i]; j++) {
            for (int k = 0; k < p[i+1]; k++) {
                matrix[j][k] = rand() % 2;
            }
        }
    }
}
void matrixChainOrder(int p[], int n, int m[][100], int s[][100]) {
    for(int i=1; i<=n; i++)
        m[i][i] = 0;
        for(int l=2; l<=n; l++) {
        for(int i=1; i<=n-l+1; i++) {
            int j = i+l-1;
            m[i][j] = INT_MAX;
            for(int k=i; k<=j-1; k++) {
                int 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;
                }
            }
        }
    }
}

void printOptimalParenthesis(int s[][100], int i, int j) {
    if(i == j)
        cout << "A" << i;
```

```cpp
    else
    {
        cout << "(";
        printOptimalParenthesis(s, i, s[i][j]);
        printOptimalParenthesis(s, s[i][j]+1, j);
        cout << ")";
    }
}

int main()
{
    int p[10];
    srand ( time(NULL) );
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<> distr(15, 46);
    for(int i=0; i<10; ++i)
    p[i] = distr(gen);
    int n = sizeof(p)/sizeof(p[0]) - 1;
    generateMatrices(p);
    int m[100][100];
    int s[100][100];
    matrixChainOrder(p, n, m, s);
    cout << "Optimal Parenthesization: ";
    printOptimalParenthesis(s, 1, n);
    cout << endl;
    cout << "Minimum Number of Scalar Multiplications: " << m[1][n] << endl;
    cout << "m table:";
    for(int a = 0; a < 10; a++)
```

```cpp
    cout << "m table:";
    for(int a = 0; a < 10; a++)
    {
        for(int b = 0; b < 10; b++)
        {
            if(m[a][b] == 0){continue;}
            cout << m[a][b] << " ";
        }
        cout << endl;
    }
    cout << "s table:";
    for(int a = 0; a < 10; a++)
    {
        for(int b = 0; b < 10; b++)
        {
            if(s[a][b] == 0){continue;}
            cout << s[a][b] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

```
Optimal Parenthesization: (A1(A2(A3(A4(A5(A6(A7(A8A9))))))))
Minimum Number of Scalar Multiplications: 161993
m table:
36120 85484 127842 179620 186843 187224 231600 161993
34440 77532 127842 149433 156264 187224 140063
44772 91728 125073 136104 165000 125783
68757 95004 108552 150864 106267
48633 70176 110424 79084
29928 74304 50575
29928 29376
17544

s table:
1 2 1 2 1 1 7 1
2 2 4 2 2 7 2
3 4 4 3 7 3
4 4 4 7 4
5 5 7 5
6 7 6
7 7
8



...Program finished with exit code 0
Press ENTER to exit console.
```

OBSERVATION:

Complexity Analysis:

Time Complexity - We are using three nested for loops, each of which is iterating roughly O (n) times. Hence, the overall time complexity is O(n3).

Space Complexity - We are using an auxiliary *dp* array of dimensions, $(n-1)\times(n-1)$ hence space complexity is $O(n2)$

Matrix chain multiplication (or the matrix chain ordering problem) is an optimization problem concerning the most efficient way to multiply a given sequence of matrices. The problem is not actually to perform the multiplications, but merely to decide the sequence of the matrix multiplications involved. The problem may be solved using dynamic programming.

There are many options because matrix multiplication is associative. In other words, no matter how the product is parenthesized, the result obtained will remain the same. For example, for four matrices A, B, C, and D, there are five possible options:

$$((AB)C)D = (A(BC))D = (AB)(CD) = A((BC)D) = A(B(CD)).$$

If *A* is a 10 × 40 matrix, *B* is a 40 × 5 matrix, and *C* is a 5 × 80 matrix, then

> computing (AB)C needs (10×40×5) + (10×5×80) = 2000 + 4000 = 6000 operations, while
> computing A(BC) needs (40×5×80) + (10×40×80) = 16000 + 32000 = 48000 operations.

Therefore I found out that finding an optimal way to multiply will significantly reduce running times especially with large matrix values.

CONCLUSION:	Successfully performed the experiment of Matrix Chain Multiplication and in the matrix chain multiplication problem, the minimum number of multiplication steps required to multiply a chain of matrices has been calculated.
