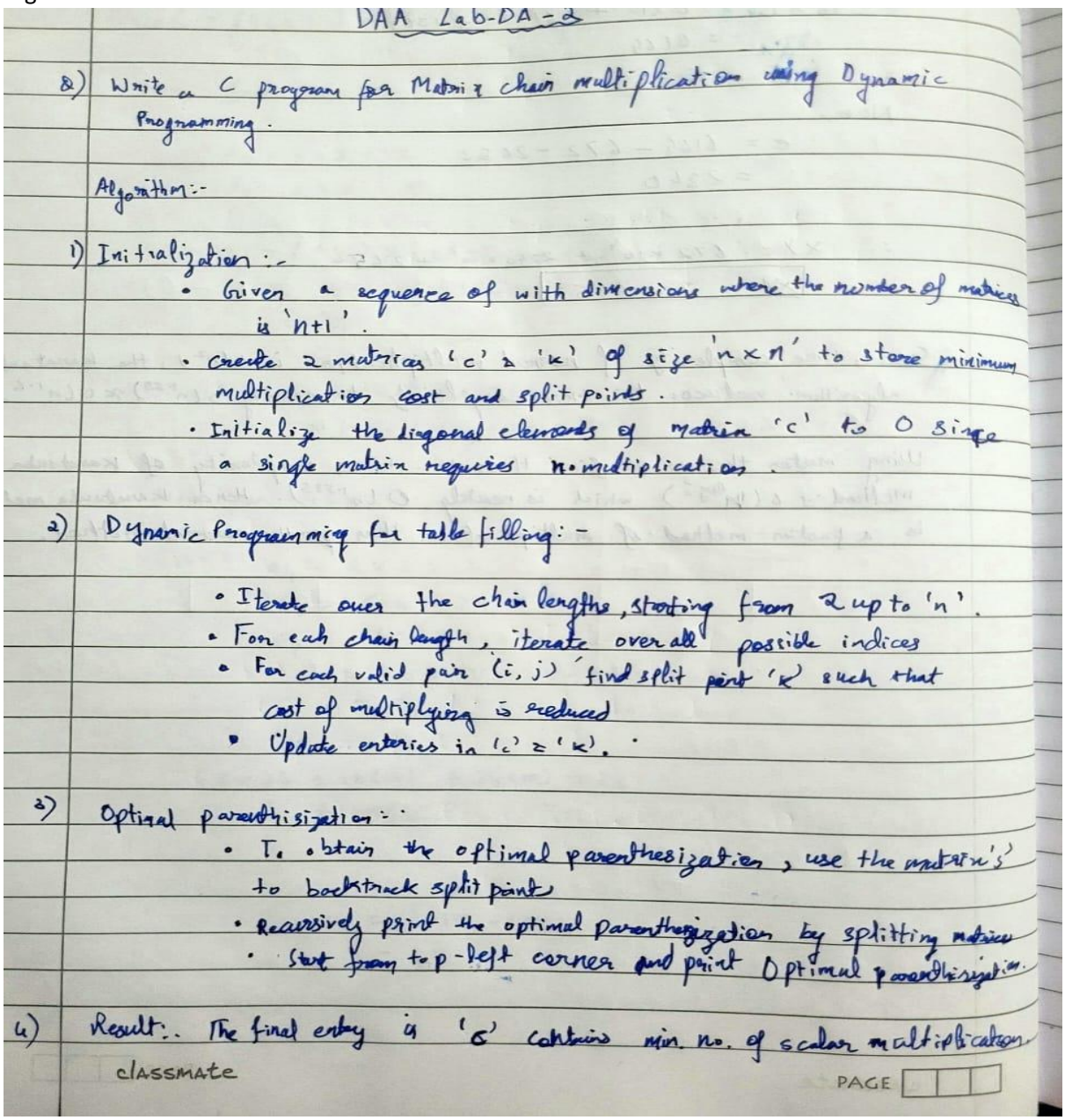
DAA LAB DA-2

NAME: AMAR MADREWAR

REGISTRATION NUMBER: 22BCE0827

Q1) Write a C program for Matrix Chain Multiplication using Dynamic Programming.

Algorithm:



DAA Lab-DA-2

8) Write a C program for Matrix chain multiplication using Dynamic Programming.

Algorithm:-

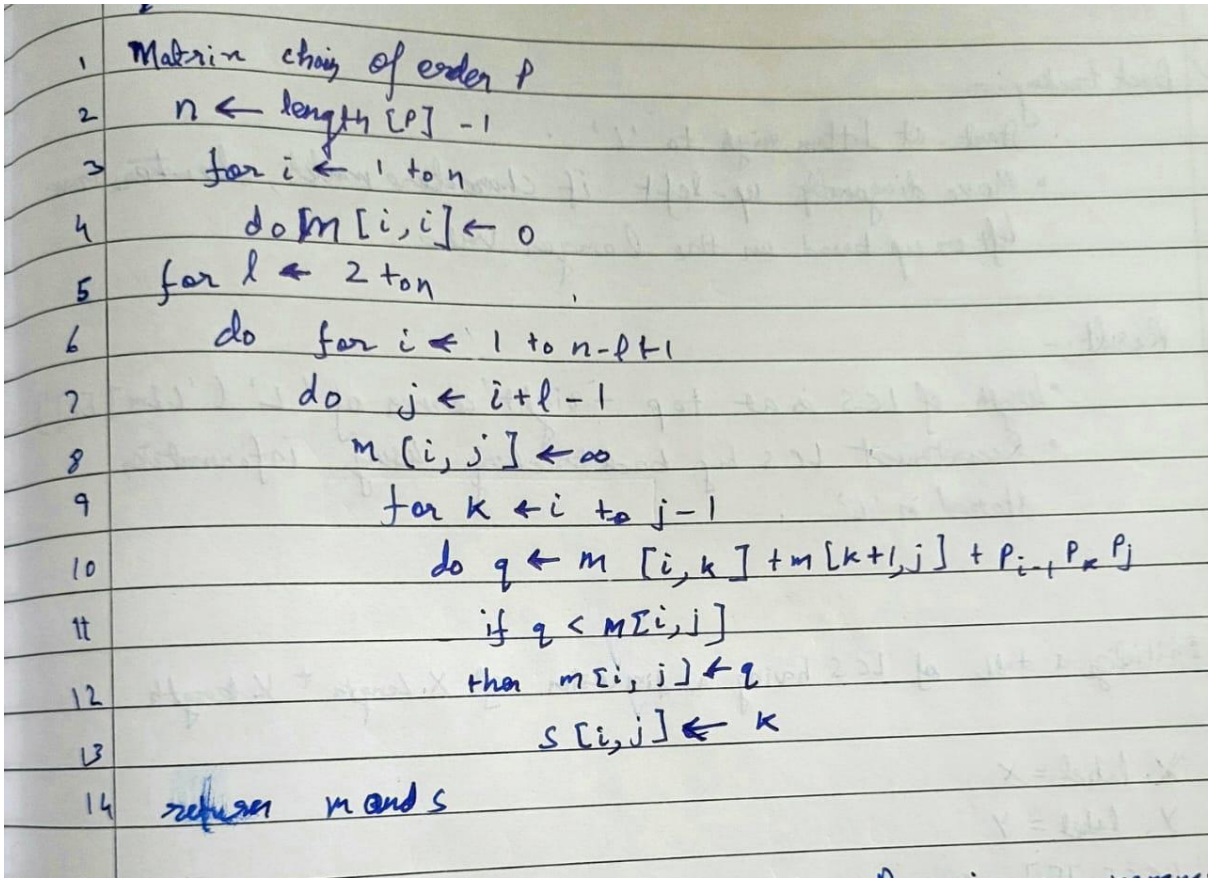
1) Initialization :-
   - Given a sequence of with dimensions where the number of matrices is 'n+1'.
   - Create 2 matrices 'c' & 'k' of size 'n x n' to store minimum multiplication cost and split points.
   - Initialize the diagonal elements of matrix 'c' to 0 since a single matrix requires no multiplications.

2) Dynamic Programming for table filling :-
   - Iterate over the chain lengths, starting from 2 up to 'n'.
   - For each chain length, iterate over all possible indices
   - For each valid pair (i, j) find split point 'k' such that cost of multiplying is reduced
   - Update enteries in 'c' & 'k'.

3) Optimal parenthisization :-
   - To obtain the optimal parenthesization, use the matrix's' to backtrack split point
   - Recursively print the optimal parenthesization by splitting matrices
   - Start from top-left corner and print Optimal parenthisization.

4) Result:- The final entry is 'c' contains min. no. of scalar multiplications

classmate PAGE



Code: #include <stdio.h>

#include <limits.h>

int matrixChainMultiplication(int dims[], int n) {

int dp[n][n];

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

dp[i][j] = 0;

}

}

for (int len = 2; len < n; len++) {

```
        for (int i = 1; i < n - len + 1; i++) {
            int j = i + len - 1;
            dp[i][j] = INT_MAX;
            for (int k = i; k <= j - 1; k++) {
                int cost = dp[i][k] + dp[k + 1][j] + dims[i - 1] * dims[k] * dims[j];
                if (cost < dp[i][j]) {
                    dp[i][j] = cost;
                }
            }
        }
    }

    return dp[1][n - 1];
}

int main() {

    int dims[] = {3, 4, 5, 2, 4, 2, 5};
    int n = sizeof(dims) / sizeof(dims[0]);

    printf("Minimum number of scalar multiplications: %d\n", matrixChainMultiplication(dims, n));

    return 0;
}
```

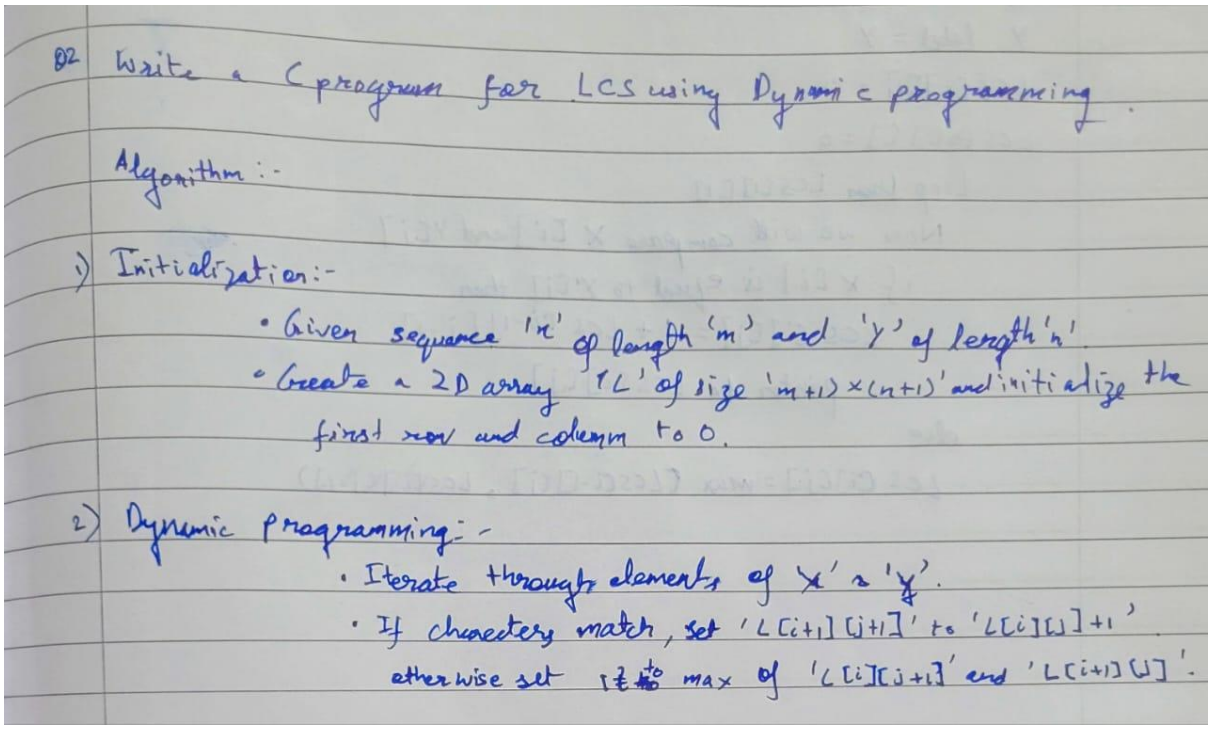
Output:

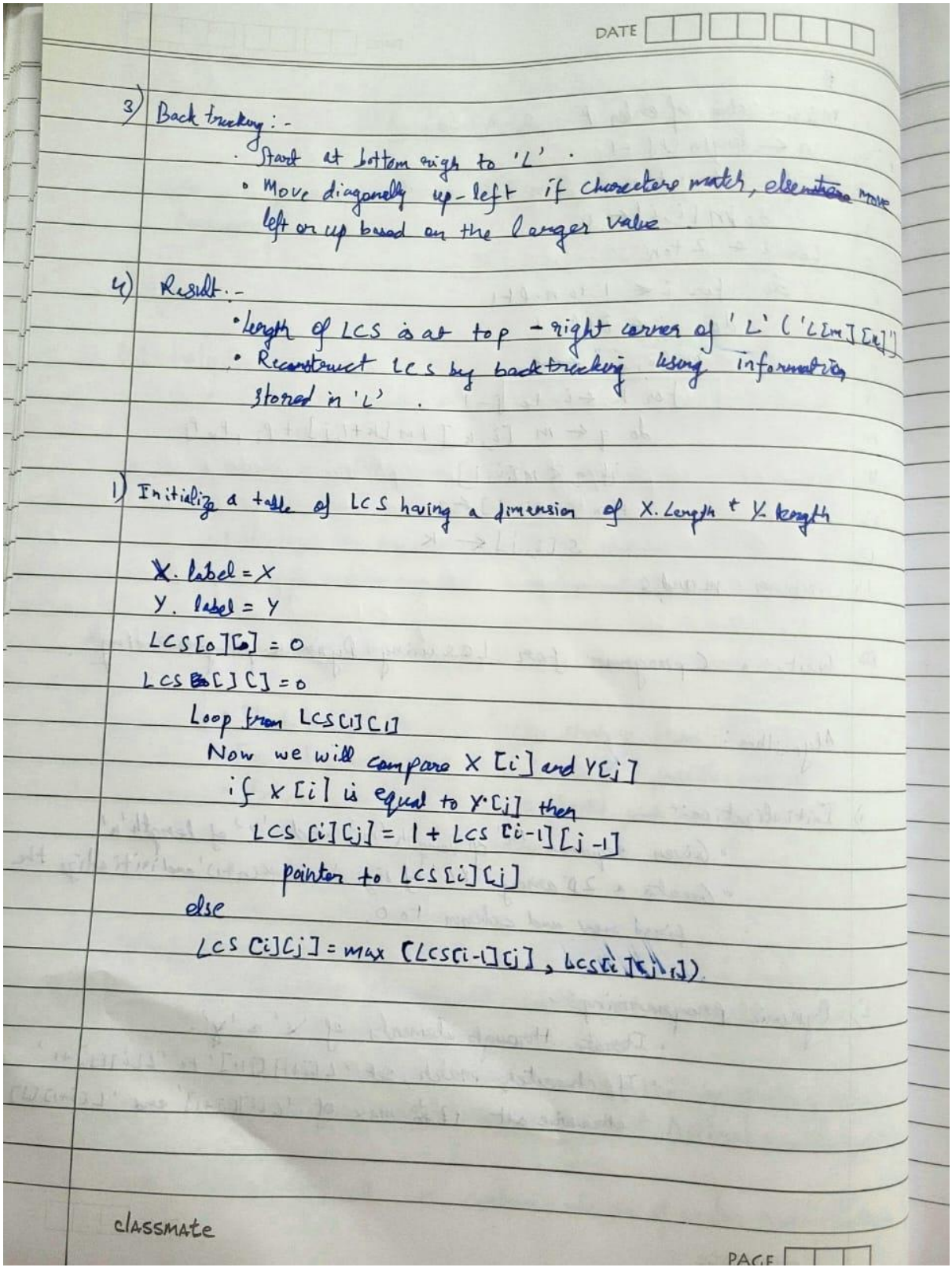
```
Minimum number of scalar multiplications: 122


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

Q2) Write a C program for LCS using Dynamic Programming.

Algorithm:





Code: #include <stdio.h>

#include <string.h>

int lcs(char X[], char Y[], int m, int n) {

int dp[m + 1][n + 1];

```
    for (int i = 0; i <= m; i++) {
        for (int j = 0; j <= n; j++) {
            if (i == 0 || j == 0)
                dp[i][j] = 0;
            else if (X[i - 1] == Y[j - 1])
                dp[i][j] = dp[i - 1][j - 1] + 1;
            else
                dp[i][j] = (dp[i - 1][j] > dp[i][j - 1]) ? dp[i - 1][j] : dp[i][j - 1];
        }
    }

    return dp[m][n];
}
void printLCS(char X[], char Y[], int m, int n) {
    int dp[m + 1][n + 1];

    for (int i = 0; i <= m; i++) {
        for (int j = 0; j <= n; j++) {
            if (i == 0 || j == 0)
                dp[i][j] = 0;
            else if (X[i - 1] == Y[j - 1])
                dp[i][j] = dp[i - 1][j - 1] + 1;
            else
                dp[i][j] = (dp[i - 1][j] > dp[i][j - 1]) ? dp[i - 1][j] : dp[i][j - 1];
        }
    }

    int index = dp[m][n];
    char lcs[index + 1];
    lcs[index] = '\0';
```

```
    int i = m, j = n;
    while (i > 0 && j > 0) {
        if (X[i - 1] == Y[j - 1]) {
            lcs[index - 1] = X[i - 1];
            i--;
            j--;
            index--;
        } else if (dp[i - 1][j] > dp[i][j - 1]) {
            i--;
        } else {
            j--;
        }
    }

    printf("Longest Common Subsequence: %s\n", lcs);
}

int main() {
    char X[] = "ABCBDAB";
    char Y[] = "BDCAB";

    int m = strlen(X);
    int n = strlen(Y);

    printf("Length of LCS: %d\n", lcs(X, Y, m, n));
    printLCS(X, Y, m, n);

    return 0;
}
```

Output:


Length of LCS: 4
Longest Common Subsequence: BDAB
...Program finished with exit code 0
Press ENTER to exit console.