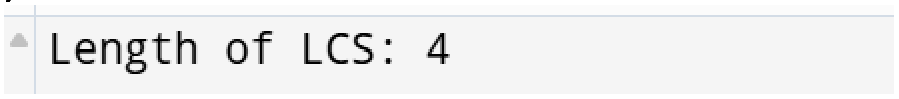
ASSIGNMENT 3

1)

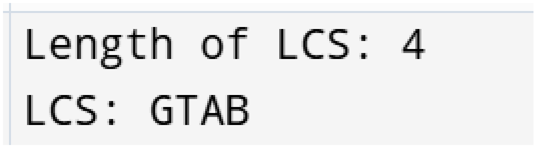
```
#include <bits/stdc++.h>
using namespace std;

int solve(string &s1, string &s2, int m, int n, vector<vector<int>> &memo) {
   if (m == 0 || n == 0)
      return 0;
   if (memo[m][n] != -1)
      return memo[m][n];
   if (s1[m - 1] == s2[n - 1])
      return memo[m][n] = 1 + solve(s1, s2, m - 1, n - 1, memo);
   return memo[m][n] = max(solve(s1, s2, m, n - 1, memo), solve(s1, s2, m - 1, n, memo));
}

int LCS(string &s1, string &s2) {
   int m = s1.length();
   int n = s2.length();
   vector<vector<int>> memo(m + 1, vector<int>(n + 1, -1));
   return solve(s1, s2, m, n, memo);
}

int main() {
   string s1 = "AGGTAB";
   string s2 = "GXTXAYB";
   int lcsLength = LCS(s1, s2);
   if (lcsLength == 0) {
      cout << "No LCS exists" << endl;
   } else {
      cout << "Length of LCS: " << lcsLength << endl;
   }
   return 0;
}
```



```
Length of LCS: 4
```

1*PRINT LCS ALSO

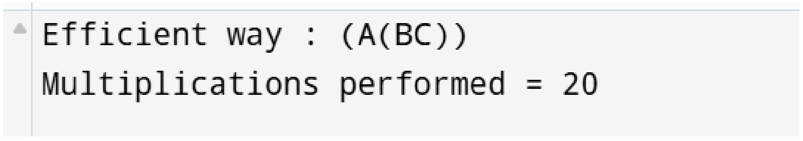
```
#include <bits/stdc++.h>
using namespace std;

string LCS(string &s1, string &s2) {
    int m = s1.length();
    int n = s2.length();
    vector<vector<int>> dp(m + 1, vector<int>(n + 1, 0));

    // Fill the dp table
    for (int i = 1; i <= m; ++i) {
        for (int j = 1; j <= n; ++j) {
            if (s1[i - 1] == s2[j - 1])
                dp[i][j] = 1 + dp[i - 1][j - 1];
            else
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
        }
    }
    // Trace back to get the LCS string
    int i = m, j = n;
    string lcs;
    while (i > 0 && j > 0) {
        if (s1[i - 1] == s2[j - 1]) {
            lcs += s1[i - 1];
            i--; j--;
        } else if (dp[i - 1][j] > dp[i][j - 1]) {
            i--;
        } else {
            j--;
        }
    }
    reverse(lcs.begin(), lcs.end()); // Reverse since we collected in reverse order
    return lcs;
}

int main() {
    string s1 = "AGGTAB";
    string s2 = "GXTXAYB";
    string lcs = LCS(s1, s2);

    if (lcs.empty()) {
        cout << "No LCS exists" << endl;
    } else {
        cout << "Length of LCS: " << lcs.length() << endl;
        cout << "LCS: " << lcs << endl;
    }
    return 0;
}
```



```
Length of LCS: 4
LCS: GTAB
```

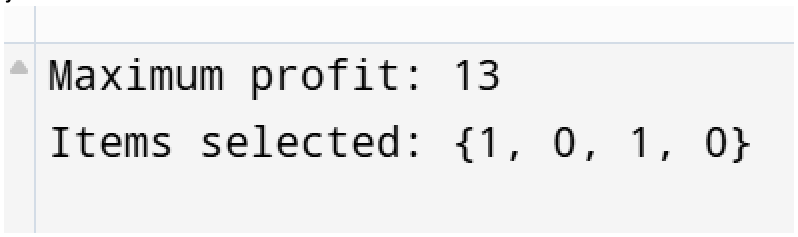
2)

```
#include <iostream>
#include <vector>
using namespace std;

// Function to print the optimal parenthesization
void printOptimalParenthesis(vector<vector<int>> &bracket, int i, int j, char &name) {
    if (i == j) {
        cout << name++;
        return;
    }
    cout << "(";
    printOptimalParenthesis(bracket, i, bracket[i][j], name);
    printOptimalParenthesis(bracket, bracket[i][j] + 1, j, name);
    cout << ")";
}
// Matrix Chain Multiplication function
void matrixChainOrder(int arr[], int n) {
    vector<vector<int>> m(n, vector<int>(n, 0));
    vector<vector<int>> bracket(n, vector<int>(n, 0));

    // Using a very large value instead of INT_MAX
    const int INF = 1e9;

    for (int L = 2; L < n; ++L) { // L is chain length
        for (int i = 1; i < n - L + 1; ++i) {
            int j = i + L - 1;
            m[i][j] = INF;
            for (int k = i; k < j; ++k) {
                int cost = m[i][k] + m[k+1][j] + arr[i-1] * arr[k] * arr[j];
                if (cost < m[i][j]) {
                    m[i][j] = cost;
                    bracket[i][j] = k;
                }
            }
        }
    }
    // Print the result
    cout << "Efficient way : ";
    char name = 'A';
    printOptimalParenthesis(bracket, 1, n - 1, name);
    cout << "\nMultiplications performed = " << m[1][n - 1] << endl;
}

int main() {
    int arr[] = {2, 1, 3, 4}; // Input
    int n = sizeof(arr) / sizeof(arr[0]);
    matrixChainOrder(arr, n);
    return 0;
}
```



```
Efficient way : (A(BC))
Multiplications performed = 20
```

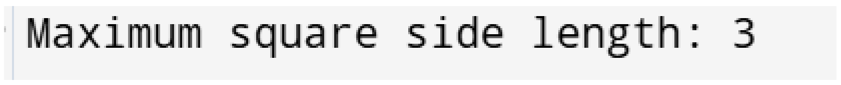
3)
```
#include <iostream>
#include <vector>
using namespace std;

void knapsack(int N, int W, vector<int>& profit, vector<int>& weight) {
    vector<vector<int>> dp(N + 1, vector<int>(W + 1, 0));

    // Build DP table
    for (int i = 1; i <= N; ++i) {
        for (int w = 1; w <= W; ++w) {
            if (weight[i - 1] <= w) {
                dp[i][w] = max(dp[i - 1][w], profit[i - 1] + dp[i - 1][w - weight[i - 1]]);
            } else {
                dp[i][w] = dp[i - 1][w];
            }
        }
    }

    // Maximum profit
    cout << "Maximum profit: " << dp[N][W] << endl;

    // Find selected items
    vector<int> selected(N, 0);
    int w = W;
    for (int i = N; i > 0 && w > 0; --i) {
        if (dp[i][w] != dp[i - 1][w]) {
            selected[i - 1] = 1; // Mark item as selected
            w -= weight[i - 1];
        }
    }
    // Print selected items vector
    cout << "Items selected: {";
    for (int i = 0; i < N; ++i) {
        cout << selected[i];
        if (i < N - 1) cout << ", ";
    }
    cout << "}" << endl;
}

int main() {
    int N = 4;
    int W = 7;
    vector<int> profit = {5, 3, 8, 6};
    vector<int> weight = {2, 3, 4, 5};

    knapsack(N, W, profit, weight);
    return 0;
}
```



```
Maximum profit: 13
Items selected: {1, 0, 1, 0}
```

4)

```
 #include <iostream>
#include <vector>
using namespace std;

int maxSquareLength(vector<vector<int>>& mat) {
    int n = mat.size();
    int m = mat[0].size();
    int maxSide = 0;

    vector<vector<int>> dp(n, vector<int>(m, 0));

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            if (i == 0 || j == 0) {
                dp[i][j] = mat[i][j]; // Copy first row and column
            } else if (mat[i][j] == 1) {
                dp[i][j] = 1 + min(min(dp[i - 1][j], dp[i][j - 1]), dp[i - 1][j - 1]);
            } else {
                dp[i][j] = 0;
            }
            if (dp[i][j] > maxSide)
                maxSide = dp[i][j];
        }
    }
    return maxSide;
}

int main() {
    vector<vector<int>> mat = {
        {0, 1, 1, 0, 1},
        {1, 1, 0, 1, 0},
        {0, 1, 1, 1, 0},
        {1, 1, 1, 1, 0},
        {1, 1, 1, 1, 1},
        {0, 0, 0, 0, 0}
    };
    int result = maxSquareLength(mat);
    cout << "Maximum square side length: " << result << endl;
    return 0;
}
```



```
Maximum square side length: 3
```