Matrix Chain Multiplication using Dynamic Programming:

Dynamic programming solves matrix chain multiplication problem by building a 2D table where **dp[i][j]** stores the minimum number of scalar multiplications needed to compute the matrix product of matrices from **i** to **j**.

Algorithm:

1. Create a 2D array **dp[n][n]** where **n** is the number of matrices.
2. Initialize the array with zeros.
3. Iterate through the matrices, filling up the **dp** table based on optimal subproblem solutions.

Code:

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

int matrixChainMultiplication(int p[], int n) {
   int dp[n][n];
   for (int i = 1; i < n; i++)
      dp[i][i] = 0;

   for (int chainLen = 2; chainLen < n; chainLen++) {
      for (int i = 1; i < n - chainLen + 1; i++) {
         int j = i + chainLen - 1;
         dp[i][j] = INT_MAX;
         for (int k = i; k <= j - 1; k++) {
            int cost = dp[i][k] + dp[k + 1][j] + p[i - 1] * p[k] * p[j];
            if (cost < dp[i][j])
               dp[i][j] = cost;
         }
      }
   }
   return dp[1][n - 1];
}

int main() {
   int p[] = {10, 30, 5, 60};
   int n = sizeof(p) / sizeof(p[0]);
   cout << "Minimum number of multiplications: " << matrixChainMultiplication(p, n);
   return 0;
}
```

Output:

```
Minimum number of multiplications: 4500
```

Space complexity: _____

Justification:_____
_____

Time complexity:

Best case time complexity: _____

Justification:_____
_____

Worst case time complexity: _____

Justification:_____