Matrix Chain Multiplication

Given the dimension of a sequence of matrices in an array **arr[]**, where the dimension of the **i^{th}** matrix is **(arr[i-1] * arr[i])**, the task is to find the most efficient way to multiply these matrices together such that the total number of element multiplications is minimum.

Examples:

Input: *arr[] = {40, 20, 30, 10, 30}*
Output: *26000*
Explanation:*There are 4 matrices of dimensions 40×20, 20×30, 30×10, 10×30.*
Let the input 4 matrices be A, B, C and D.
The minimum number of multiplications are obtained by
putting parenthesis in following way (A(BC))D.
*The minimum is 20*30*10 + 40*20*10 + 40*10*30*
Input: *arr[] = {1, 2, 3, 4, 3}*
Output: *30*
Explanation: *There are 4 matrices of dimensions 1×2, 2×3, 3×4, 4×3.*
Let the input 4 matrices be A, B, C and D.
The minimum number of multiplications are obtained by
putting parenthesis in following way ((AB)C)D.
*The minimum number is 1*2*3 + 1*3*4 + 1*4*3 = 30*
Input: *arr[] = {10, 20, 30}*
Output: *6000*
Explanation: *There are only two matrices of dimensions 10×20 and 20×30.*
*So there is only one way to multiply the matrices, cost of which is 10*20*30*

Recommended Problem
Matrix Chain Multiplication

Matrix Chain Multiplication using Recursion:

We can solve the problem using recursion based on the following facts and observations:

*Two matrices of size **m*n** and **n*p** when multiplied, they generate a matrix of size **m*p** and the number of multiplications performed are **m*n*p**.*
*Now, for a given chain of **N** matrices, the first partition can be done in **N-1** ways. For example, sequence of matrices A, B, C and D can be grouped as (A)(BCD), (AB)(CD) or (ABC)(D) in these 3 ways.*
*So a range **[i, j]** can be broken into two groups like **{[i, i+1], [i+1, j]}, {[i, i+2], [i+2, j]}, . . . , {[i, j-1], [j-1, j]}**.*

- *Each of the groups can be further partitioned into smaller groups and we can find the total required multiplications by solving for each of the groups.*
- *The minimum number of multiplications among all the first partitions is the required answer.*

Follow the steps mentioned below to implement the approach:

- Create a recursive function that takes **i** and **j** as parameters that determines the range of a group.

    - Iterate from **k = i to j** to partition the given range into two groups.
    - Call the recursive function for these groups.
    - Return the minimum value among all the partitions as the required minimum number of multiplications to multiply all the matrices of this group.
- The minimum value returned for the range **0** to **N-1** is the required answer.

Below is the implementation of the above approach.

```
// C++ code to implement the

// matrix chain multiplication using recursion



#include <bits/stdc++.h>

using namespace std;



// Matrix Ai has dimension p[i-1] x p[i]

// for i = 1 . . . n

int MatrixChainOrder(int p[], int i, int j)

{

    if (i == j)

        return 0;

    int k;

    int mini = INT_MAX;

    int count;



    // Place parenthesis at different places
```

```
    // between first and last matrix,
    // recursively calculate count of multiplications
    // for each parenthesis placement
    // and return the minimum count
    for (k = i; k < j; k++)
    {
        count = MatrixChainOrder(p, i, k)
                + MatrixChainOrder(p, k + 1, j)
                + p[i - 1] * p[k] * p[j];

        mini = min(count, mini);
    }

    // Return minimum count
    return mini;
}

// Driver Code
int main()
{
    int arr[] = { 1, 2, 3, 4, 3 };
    int N = sizeof(arr) / sizeof(arr[0]);
```

```
    // Function call
    cout << "Minimum number of multiplications is "
         << MatrixChainOrder(arr, 1, N - 1);
    return 0;
}

// This code is contributed by Shivi_Aggarwal
```

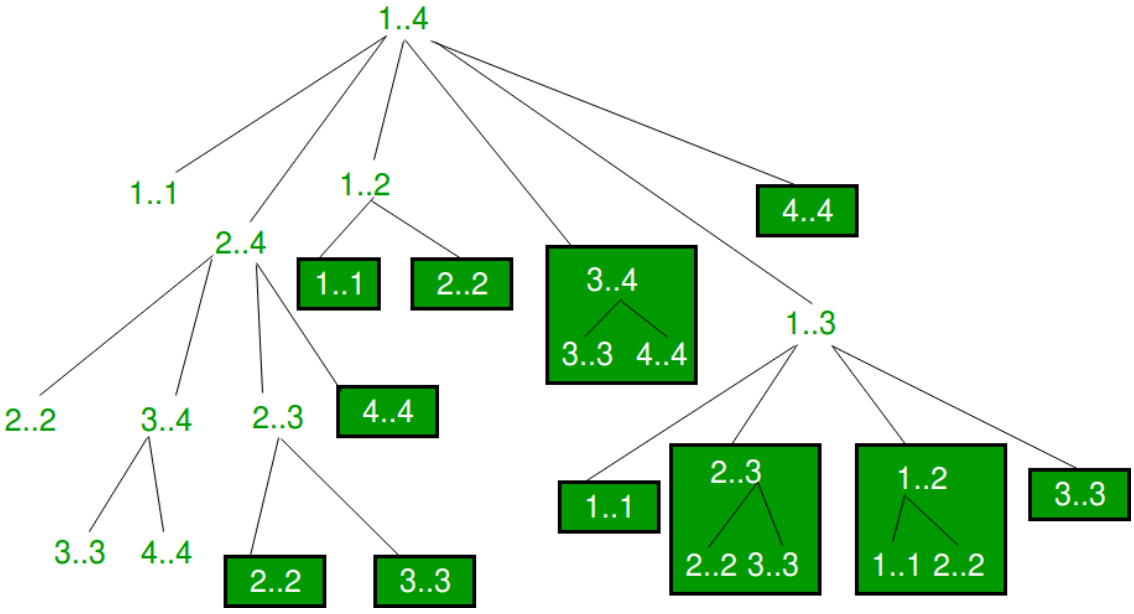
Output

```
Minimum number of multiplications is 30
```

The time complexity of the solution is exponential
Auxiliary Space: O(1)

Dynamic Programming Solution for Matrix Chain Multiplication using Memoization:

Below is the recursion tree for the 2nd example of the above recursive approach:



If observed carefully you can find the following two properties:

1) Optimal Substructure: *In the above case, we are breaking the bigger groups into smaller subgroups and solving them to finally find the minimum number of*

multiplications. Therefore, it can be said that the problem has optimal substructure property.
2) Overlapping Subproblems: *We can see in the recursion tree that the same subproblems are called again and again and this problem has the Overlapping Subproblems property.*
So Matrix Chain Multiplication problem has both properties of a dynamic programming problem. So recomputations of same subproblems can be avoided by constructing a temporary array ***dp[][]*** *in a bottom up manner.*

Follow the below steps to solve the problem:

- Build a matrix **dp[][]** of size **N*N** for memoization purposes.
- Use the same recursive call as done in the above approach:
    - When we find a range **(i, j)** for which the value is already calculated, return the minimum value for that range (i.e., **dp[i][j]**).
    - Otherwise, perform the recursive calls as mentioned earlier.
- The value stored at **dp[0][N-1]** is the required answer.

Below is the implementation of the above approach

```
// C++ program using memoization

#include <bits/stdc++.h>

using namespace std;

int dp[100][100];



// Function for matrix chain multiplication

int matrixChainMemoised(int* p, int i, int j)

{

    if (i == j)

    {

        return 0;

    }
```

```
    if (dp[i][j] != -1)
    {
        return dp[i][j];
    }
    dp[i][j] = INT_MAX;
    for (int k = i; k < j; k++)
    {
        dp[i][j] = min(
            dp[i][j], matrixChainMemoised(p, i, k)
                    + matrixChainMemoised(p, k + 1, j)
                      + p[i - 1] * p[k] * p[j]);
    }
    return dp[i][j];
}
int MatrixChainOrder(int* p, int n)
{
    int i = 1, j = n - 1;
    return matrixChainMemoised(p, i, j);
}

// Driver Code
int main()
```

```
{

    int arr[] = { 1, 2, 3, 4 };

    int n = sizeof(arr) / sizeof(arr[0]);

    memset(dp, -1, sizeof dp);



    cout << "Minimum number of multiplications is "

         << MatrixChainOrder(arr, n);

}



// This code is contributed by Sumit_Yadav
```


Output

```
Minimum number of multiplications is 18
```

Time Complexity: $O(N^3)$
Auxiliary Space: $O(N^2)$ ignoring recursion stack space

Dynamic Programming Solution for Matrix Chain Multiplication using Tabulation (Iterative Approach):

In iterative approach, we initially need to find the number of multiplications required to multiply two adjacent matrices. We can use these values to find the minimum multiplication required for matrices in a range of length 3 and further use those values for ranges with higher lengths.

Build on the answer in this manner till the range becomes [0, N-1].

Follow the steps mentioned below to implement the idea:

- Iterate from **l = 2 to N-1** which denotes the length of the range:
    - Iterate from **i = 0 to N-1**:
        - Find the right end of the range (**j**) having **l** matrices.
        - Iterate from **k = i+1 to j** which denotes the point of partition.
            - Multiply the matrices in range (**i, k**) and (**k, j**).
            - This will create two matrices with dimensions **arr[i-1]*arr[k]** and **arr[k]*arr[j]**.

    - The number of multiplications to be performed to multiply these two matrices (say **X**) are **arr[i-1]*arr[k]*arr[j]**.
    - The total number of multiplications is **dp[i][k]+ dp[k+1][j] + X**.
- The value stored at **dp[1][N-1]** is the required answer.

Below is the implementation of the above approach.

```
// See the Cormen book for details of the

// following algorithm

#include <bits/stdc++.h>

using namespace std;



// Matrix Ai has dimension p[i-1] x p[i]

// for i = 1..n

int MatrixChainOrder(int p[], int n)

{



    /* For simplicity of the program, one

    extra row and one extra column are

    allocated in m[][]. 0th row and 0th

    column of m[][] are not used */

    int m[n][n];



    int i, j, k, L, q;
```

```
/* m[i, j] = Minimum number of scalar

multiplications needed to compute the

matrix A[i]A[i+1]...A[j] = A[i..j] where

dimension of A[i] is p[i-1] x p[i] */


// cost is zero when multiplying

// one matrix.

for (i = 1; i < n; i++)

    m[i][i] = 0;


// L is chain length.

for (L = 2; L < n; L++)

{

    for (i = 1; i < n - L + 1; i++)

    {

        j = i + L - 1;

        m[i][j] = INT_MAX;

        for (k = i; k <= j - 1; k++)

        {

            // q = cost/scalar multiplications

            q = m[i][k] + m[k + 1][j]
```

```
                    + p[i - 1] * p[k] * p[j];
                if (q < m[i][j])
                    m[i][j] = q;
            }
        }
    }

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

// Driver Code
int main()
{
    int arr[] = { 1, 2, 3, 4 };
    int size = sizeof(arr) / sizeof(arr[0]);

    cout << "Minimum number of multiplications is "
         << MatrixChainOrder(arr, size);

    getchar();
    return 0;
}
```

```
// This code is contributed

// by Akanksha Rai
```

Output

```
Minimum number of multiplications is 18
```

Time Complexity: $O(N^3)$
Auxiliary Space: $O(N^2)$