Matrix Chain Multiplication :-

Given an array **arr[]** which represents **the** dimensions of a sequence of matrices where the i^{th} matrix has the dimensions **(arr[i-1] x arr[i])** for i>=1, find the most efficient way to multiply these matrices together. The efficient way is the one that involves the least number of multiplications.

Examples:

Input: arr[] = {2, 1, 3, 4}
Output: 20
Explanation: There are 3 matrices of dimensions 2 × 1, 1 × 3, and 3 × 4, Let this 3 input matrices be M1, M2, and M3. There are two ways to multiply: ((M1 x M2) x M3) and (M1 x (M2 x M3)), note that the result of (M1 x M2) is a 2 × 3 matrix and result of (M2 x M3) is a 1 × 4 matrix.
((M1 x M2) x M3) requires (2 x 1 x 3) + (2 x 3 x 4) = 30
(M1 x (M2 x M3)) requires (1 x 3 x 4) + (2 x 1 x 4) = 20.
The minimum of these two is 20.

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 this 4 input matrices be M1, M2, M3 and M4. The minimum number of multiplications are obtained by ((M1 x M2) x M3) x M4). The minimum number is (1 x 2 x 3) + (1 x 3 x 4) + (1 x 4 x 3) = 30.

Input: arr[] = {3, 4}
Output: 0
Explanation: As there is only one matrix so, there is no cost of multiplication.

Constraints:
2 ≤ arr.size() ≤ 100
1 ≤ arr[i] ≤ 200

Try more examples

arr[] = [2, 1, 3, 4] → size = n

matA = m × n (n columns) → (n-1) matrices

dimension(A) = m × n

dimension(mat[i]) = arr[i-1] × arr[i]

i = 1

dim(mat[1]) = arr[0] × arr[1]

= 2 × 1

mat[2] = 1 × 3

mat[3] = 3 × 4

mat[1] × mat[2] × mat[3] = (mat[1] × mat[2]) × mat[3]

= mat[1] × (mat[2] × mat[3])

arr[i-1] × arr[j]

The Brute Force Approach :-

(mat[1] × mat[2]) × (mat[3] × (mat[4] ... mat[n]))

a × b × c

i → j

a × b b × c

for(int k = i; k < j; k++)

ans = min(help(i,k) + help(k+1, j) + arr[i-1] × arr[k] × arr[j])

i × k × j

Base Case :-

if(i == j) return 0;

a × c

```
C++ (g++ 5.4)   Start Timer
1  //{ Driver Code Ends
7
8  class Solution {
9    public:
10     int matrixMultiplication(vector<int> &arr) {
11         // code here
12         int n=arr.size();
13         vector<vector<int>> dp(n, vector<int>(n, INT_MAX));
14         return help(0, n-1, arr, dp);
15     }
16     int help(int i, int j, vector<int> &arr, vector<vector<int>> &dp){
17         int n=arr.size();
18         if(dp[i][j]!=INT_MAX)return dp[i][j];
19         if(i+1==j)return dp[i][j]=0;
20         for(int k=i+1;k<j;k++){
21             dp[i][j]=min(dp[i][j], help(i, k, arr, dp)+help(k, j, arr, dp)+arr[i]*arr[k]*arr[j]);
22         }
23         return dp[i][j];
24     }
25 };
```