



EXPERIMENT - 3

Algorithms Design and Analysis Lab

Aim

To implement Matrix Multiplication and analyse its time complexity.

Syeda Reeha Quasar

14114802719

4C7

EXPERIMENT – 3

Aim:

To implement Matrix Multiplication and analyse its time complexity.

Theory:

Given a sequence of matrices $A_1, A_2 \dots A_n$, insert parentheses so that the product of the matrices, in order, is unambiguous and needs the minimal number of multiplications

Matrix multiplication is associative: $A_1 (A_2 A_3) = (A_1 A_2) A_3$

A product is unambiguous if no factor is multiplied on both the left and the right and all factors are either a single matrix or an unambiguous product.

Multiplying an $i \times j$ and a $j \times k$ matrix requires ijk multiplications. Each element of the product requires j multiplications, and there are ik elements

$$\begin{array}{ccc}
 & \vec{b}_1 & \vec{b}_2 \\
 & \downarrow & \downarrow \\
 \begin{array}{l} \vec{a}_1 \rightarrow \\ \vec{a}_2 \rightarrow \end{array} & \begin{bmatrix} 1 & 7 \\ 2 & 4 \end{bmatrix} \cdot \begin{bmatrix} 3 & 3 \\ 5 & 2 \end{bmatrix} = \begin{bmatrix} \vec{a}_1 \cdot \vec{b}_1 & \vec{a}_1 \cdot \vec{b}_2 \\ \vec{a}_2 \cdot \vec{b}_1 & \vec{a}_2 \cdot \vec{b}_2 \end{bmatrix} \\
 A & B & C
 \end{array}$$

Number of Parenthesizations

Given the matrices A_1, A_2, A_3, A_4 . Assume the dimensions of $A_1 = d_0 \times d_1$, etc. Below are the five possible parenthesizations of these arrays, along with the number of multiplications:

1. $(A_1A_2)(A_3A_4):d_0d_1d_2+d_2d_3d_4+d_0d_2d_4$
2. $((A_1A_2)A_3)A_4:d_0d_1d_2+d_0d_2d_3+d_0d_3d_4$
3. $(A_1(A_2A_3))A_4:d_1d_2d_3+d_0d_1d_3+d_0d_3d_4$
4. $A_1((A_2A_3)A_4):d_1d_2d_3+d_1d_3d_4+d_0d_1d_4$
5. $A_1(A_2(A_3A_4)):d_2d_3d_4+d_1d_2d_4+d_0d_1d_4$

The number of parenthesizations is at least $T(n) \geq T(n-1) + T(n-1)$. Since the number with the first element removed is $T(n-1)$, which is also the number with the last removed. Thus the number of parenthesizations is $\Omega(2^n)$. The number is actually $T(n) = n-1 \sum_{k=1}^{n-1} T(k)T(n-k)$ which is related to the *Catalan numbers*. This is because the original product can be split into 2 sub products in k places. Each split is to be parenthesized optimally. This recurrence is related to the *Catalan numbers*.

Characterizing the Optimal Parenthesization

An optimal parenthesization of $A_1 \dots A_n$ must break the product into two expressions, each of which is parenthesized or is a single array. Assume the break occurs at position k . In the optimal solution, the solution to the product $A_1 \dots A_k$ must be optimal otherwise, we could improve $A_1 \dots A_n$ by improving $A_1 \dots A_k$ but the solution to $A_1 \dots A_n$ is known to be optimal. This is a contradiction. Thus the solution to $A_1 \dots A_n$ is known to be optimal.

Principle of Optimality

This problem exhibits the Principle of Optimality. The optimal solution to product $A_1 \dots A_n$ contains the optimal solution to two sub products. Thus we can use Dynamic Programming. Consider a recursive solution

Then improve its performance with memoization or by rewriting bottom up

Matrix Dimensions

Consider matrix product $A_1 \times \dots \times A_n$. Let the dimensions of matrix A_i be $d_{i-1} \times d_i$. Thus the dimensions of matrix product $A_i \times \dots \times A_j$ are $d_{i-1} \times d_j$.

Recursive Solution

Let $M[i,j]$ represent the number of multiplications required for matrix product $A_i \times \dots \times A_j$. For $1 \leq i \leq j \leq n$: $M[i,i] = 0$ since no product is required

The optimal solution of $A_i \times A_j$ must break at some point, k , with $i \leq k < j$

Thus, $M[i,j] = M[i,k] + M[k+1,j] + d_{i-1}d_kd_j$

Thus, $M[i,j] = \begin{cases} 0 & \text{if } i=j \\ \min_{i \leq k < j} \{M[i,k] + M[k+1,j] + d_{i-1}d_kd_j\} & \text{if } i < j \end{cases}$

Pseudo code:

// Matrix A_i has dimension $p[i-1] \times p[i]$ for $i = 1..n$

MATRIX-CHAIN-ORDER (p)

$n \leftarrow \text{length}[p] - 1$

for $i \leftarrow 1$ to n

do $m[i,i] \leftarrow 0$

for $l \leftarrow 2$ to n

do for $i \leftarrow 1$ to $n-l+1$

do $j \leftarrow i+l-1$

$m[i,j] \leftarrow \infty$

for $k \leftarrow i$ to $j-1$

do $q \leftarrow m[i,k] + m[k+1,j] + p_{i-1}p_kp_j$

if $q < m[i,j]$

then $m[i,j] \leftarrow q$

$s[i,j] \leftarrow k$

return m and s

PRINT-OPTIMAL-PARENS (s, i, j)

```

if i=j
then print "Ai"
    else print " ( "
        PRINT-OPTIMAL-PARENS (s, i, s[i,j])
        PRINT-OPTIMAL-PARENS (s, s[i,j]+1, j)
    print " ) "

```

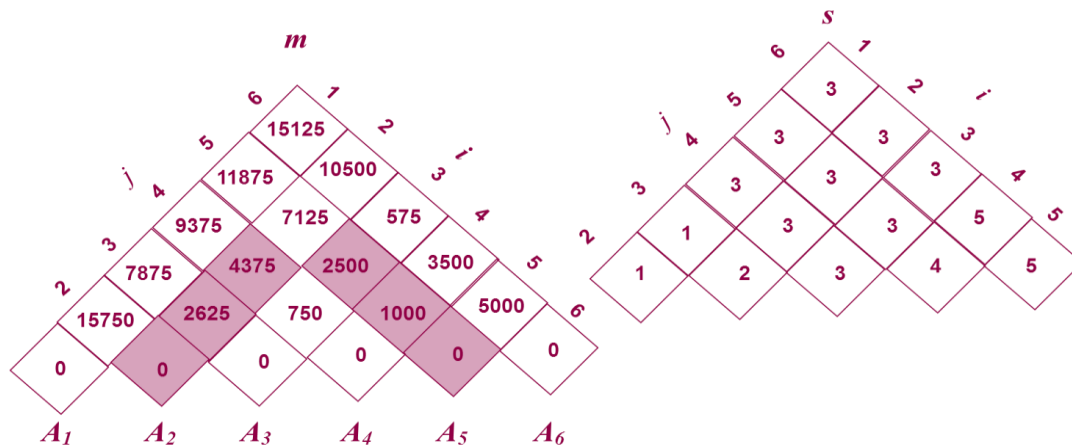
Conditions:

1. The main condition of matrix multiplication is that the number of columns of the 1st matrix must equal to the number of rows of the 2nd one.
2. As a result of multiplication you will get a new matrix that has the same quantity of rows as the 1st one has and the same quantity of columns as the 2nd one.
3. For example if you multiply a matrix of 'n' x 'k' by 'k' x 'm' size you'll get a new one of 'n' x 'm' dimension.

Sample Example:

matrix	dimension
A_1	30 x 35
A_2	35 x 15
A_3	15 x 5
A_4	5 x 10
A_5	10 x 20
A_6	20 x 25

$$\begin{aligned}
 m[2,5] &= \min \begin{cases} m[2,2] + m[3,5] + p_1 p_2 p_5 = 0 + 2500 + 35 \cdot 15 \cdot 20 = 13000 \\ m[2,3] + m[4,5] + p_1 p_3 p_5 = 2625 + 100 + 35 \cdot 5 \cdot 20 = 7125 \\ m[2,4] + m[5,5] + p_1 p_4 p_5 = 4375 + 0 + 35 \cdot 10 \cdot 20 = 11375 \end{cases} \\
 &= (7125)
 \end{aligned}$$



An optimal solution can be constructed from the computed information stored in the table $s[1 \dots n, 1 \dots n]$. The earlier matrix multiplication can be computed recursively.

$$\begin{aligned}
 p1 &= a(f - h) \\
 p3 &= (c + d)e \\
 p5 &= (a + d)(e + h) \\
 p7 &= (a - c)(e + f)
 \end{aligned}$$

$$\begin{aligned}
 p2 &= (a + b)h \\
 p4 &= d(g - e) \\
 p6 &= (b - d)(g + h)
 \end{aligned}$$

The $A \times B$ can be calculated using above seven multiplications.

Following are values of four sub-matrices of result C

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} p5 + p4 - p2 + p6 & p1 + p2 \\ p3 + p4 & p1 + p5 - p3 - p7 \end{bmatrix}$$

A B C

A, B and C are square matrices of size $N \times N$

a, b, c and d are submatrices of A, of size $N/2 \times N/2$

e, f, g and h are submatrices of B, of size $N/2 \times N/2$

$p1, p2, p3, p4, p5, p6$ and $p7$ are submatrices of size $N/2 \times N/2$

Result and Analysis

Clearly, the space complexity of this procedure is $O(n^2)$. Since the tables m and s require $O(n^2)$ space. As far as the time complexity is concern, a simple inspection of the for-loop(s) structures gives us a running time of the procedure. Since, the three for-loops are nested three deep, and each one of them iterates at most n times (that is to say indices L , i , and j takes on at most $n - 1$ values). Therefore, the running time of this procedure is $O(n^3)$.

Addition and Subtraction of two matrices takes $O(N^2)$ time. So time complexity can be written as

$$T(N) = 7T(N/2) + O(N^2)$$

From Master's Theorem, time complexity of this method is

$O(N^{\log_2 7})$ which is approximately $O(N^{2.8074})$

Matrix Multiplication:

Source Code:

```
#include <bits/stdc++.h>
#include <chrono>

using namespace std;

void printMatrix(int arr[10][10], int r, int c)
{
    for (int i = 0; i < r; ++i)
        for (int j = 0; j < c; ++j)
        {
            cout << " " << arr[i][j];
            if (j == c - 1)
                cout << endl;
        }
}

int main()
{
    int a[10][10], b[10][10], multipliedMatrix[10][10], r1, c1, r2, c2, i, j, k;

    cout << "Enter rows and columns for first matrix: ";
    cin >> r1 >> c1;
    cout << "Enter rows and columns for second matrix: ";
    cin >> r2 >> c2;

    while (c1 != r2) // matrix multiplication validation check
    {
        cout << "Error! column of first matrix not equal to row of second.";

        cout << "Enter rows and columns for first matrix: ";
        cin >> r1 >> c1;

        cout << "Enter rows and columns for second matrix: ";
        cin >> r2 >> c2;
    }

    cout << endl
        << "\n Enter elements of matrix 1:" << endl;
```



```
for (i = 0; i < r1; ++i)
    for (j = 0; j < c1; ++j)
    {
        cin >> a[i][j];
    }

cout << endl
    << "\n Enter elements of matrix 2:" << endl;
for (i = 0; i < r2; ++i)
    for (j = 0; j < c2; ++j)
    {
        cin >> b[i][j];
    }

// Initializing elements of matrix multipliedMatrix to 0.
for (i = 0; i < r1; ++i)
    for (j = 0; j < c2; ++j)
    {
        multipliedMatrix[i][j] = 0;
    }

auto start = chrono::high_resolution_clock::now();
// unsync the I/O of C and C++.
ios_base::sync_with_stdio(false);


for (i = 0; i < r1; ++i) // multiplication
    for (j = 0; j < c2; ++j)
        for (k = 0; k < c1; ++k)
        {
            multipliedMatrix[i][j] += a[i][k] * b[k][j];
        }

auto end = chrono::high_resolution_clock::now();

cout << "multiply of the matrix=\n";
printMatrix(multipliedMatrix, r1, c2);

double time_taken = chrono::duration_cast<chrono::nanoseconds>(end -
start).count();
time_taken *= 1e-9;
cout << "Time difference is: " << time_taken << setprecision(6) << endl;

return 0;
}
```

Output:

```
Enter rows and columns for first matrix: 2 2
Enter rows and columns for second matrix: 3 3
Error! column of first matrix not equal to row of second.
Enter rows and columns for first matrix: 2 2
Enter rows and columns for second matrix: 2 2

Enter elements of matrix 1:
3 4
2 1

Enter elements of matrix 2:
1 5
3 7
multiply of the matrix=
15 43
5 17
Time difference is: 4.7856e-05

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

```
multiply of the matrix=
15 43
5 17
Time difference is: 4.7856e-05

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

Strassen algorithm:

Source Code:

```
#include <bits/stdc++.h>
#include <chrono>

using namespace std;

void printMatrix(int arr[2][2], int r, int c)
{
    for (int i = 0; i < r; ++i)
        for (int j = 0; j < c; ++j)
        {
            cout << " " << arr[i][j];
            if (j == c - 1)
                cout << endl;
        }
}

int main()
{
    int a[2][2], b[2][2], mult[2][2];
    int m1, m2, m3, m4, m5, m6, m7, i, j;

    cout << "Matrix Multiplication Strassen's method: \n";
    cout << "Enter the elements of 2x2 Matrix 1:\n";
    for (i = 0; i < 2; i++)
    {
        for (j = 0; j < 2; j++)
        {
            cin >> a[i][j];
        }
    }

    cout << "Enter the elements of 2x2 Matrix 2:\n";
    for (i = 0; i < 2; i++)
    {
        for (j = 0; j < 2; j++)
        {
            cin >> b[i][j];
        }
    }
}
```

```
}

auto start = chrono::high_resolution_clock::now();
// unsync the I/O of C and C++.
ios_base::sync_with_stdio(false);

m1 = (a[0][0] + a[1][1]) * (b[0][0] + b[1][1]);
m2 = (a[1][0] + a[1][1]) * b[0][0];
m3 = a[0][0] * (b[0][1] - b[1][1]);
m4 = a[1][1] * (b[1][0] - b[0][0]);
m5 = (a[0][0] + a[0][1]) * b[1][1];
m6 = (a[1][0] - a[0][0]) * (b[0][0] + b[0][1]);
m7 = (a[0][1] - a[1][1]) * (b[1][0] + b[1][1]);

mult[0][0] = m1 + m4 - m5 + m7;
mult[0][1] = m3 + m5;
mult[1][0] = m2 + m4;
mult[1][1] = m1 - m2 + m3 + m6;


auto end = chrono::high_resolution_clock::now();

cout << "\nProduct of matrices is: \n";
printMatrix(mult, 2, 2);

double time_taken = chrono::duration_cast<chrono::nanoseconds>(end -
start).count();
time_taken *= 1e-9;

cout << "Time difference is: " << time_taken << setprecision(6) << endl;

return 0;
}
```

Output:


```
Matrix Multiplication Strassen's method:
Enter the elements of 2x2 Matrix 1:
3 4
2 1
Enter the elements of 2x2 Matrix 2:
1 5
3 7

Product of matrices is:
 15 43
  5 17
Time difference is: 7.2175e-05

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

```
Product of matrices is:
 15 43
  5 17
Time difference is: 7.2175e-05

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



```
Matrix Multiplication Strassen's method:
Enter the elements of 2x2 Matrix 1:
2 4
3 6
Enter the elements of 2x2 Matrix 2:
9 3
2 3

Product of matrices is:
 26 18
 39 27
Time difference is: 6.3373e-05

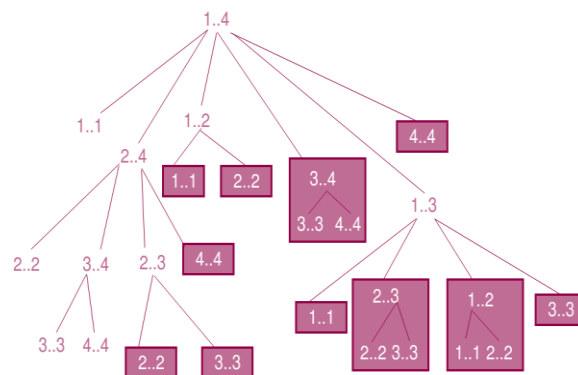
...Program finished with exit code 0
Press ENTER to exit console.
```

Given a sequence of matrices, find the most efficient way to multiply these matrices together. The problem is not actually to perform the multiplications, but merely to decide in which order to perform the multiplications.

$$(ABC)D = (AB)(CD) = A(BCD) = \dots$$
$$(AB)C = (10 \times 30 \times 5) + (10 \times 5 \times 60) = 1500 + 3000 = 4500 \text{ operations}$$

$$A(BC) = (30 \times 5 \times 60) + (10 \times 30 \times 60) = 9000 + 18000 = 27000 \text{ operations.}$$

Given an array $p[]$ which represents the chain of matrices such that the i th matrix A_i is of dimension $p[i-1] \times p[i]$. We need to write a function `MatrixChainOrder()` that should return the minimum number of multiplications needed to multiply the chain.



$$(AB)C = (10 \times 30 \times 5) + (10 \times 5 \times 60) = 1500 + 3000 = 4500 \text{ operations}$$

$$A(BC) = (30 \times 5 \times 60) + (10 \times 30 \times 60) = 9000 + 18000 = 27000 \text{ operations.}$$

Given an array $p[]$ which represents the chain of matrices such that the i th matrix A_i is of dimension $p[i-1] \times p[i]$. We need to write a function `MatrixChainOrder()` that should return the minimum number of multiplications needed to multiply the chain.

Source Code:

```
#include <bits/stdc++.h>
#include <chrono>

using namespace std;

int dp[100][100];

int matrixChainMemo(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], matrixChainMemo(p, i, k) +
matrixChainMemo(p, k + 1, j) + p[i - 1] * p[k] * p[j]);
    }
    return dp[i][j];
}

int MatrixChainOrder(int *p, int n){
    return matrixChainMemo(p, 1, n - 1);
}

int main(){
    int arr[] = {10, 20, 30, 40};
```



```
int n = sizeof(arr) / sizeof(arr[0]);
memset(dp, -1, sizeof dp);

auto start = chrono::high_resolution_clock::now();

ios_base::sync_with_stdio(false);

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

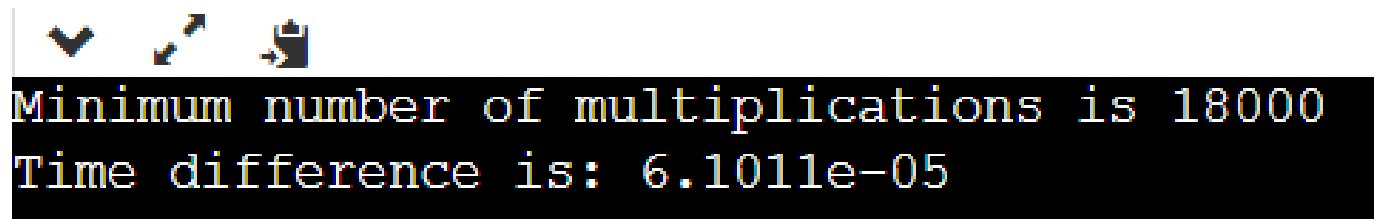
auto end = chrono::high_resolution_clock::now();

double time_taken = chrono::duration_cast<chrono::nanoseconds>(end -
start).count();
time_taken *= 1e-9;

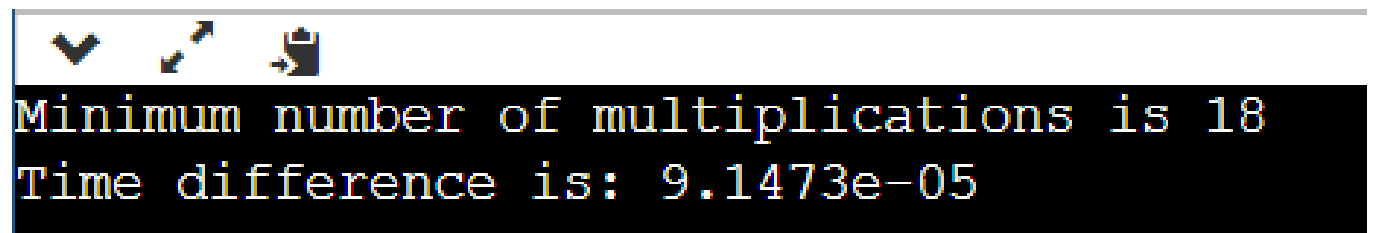
cout << "Time difference is: " << time_taken << setprecision(6) << endl;

return 0;
}
```

Output:

A terminal window with a black background and yellow text. It shows the output of the program: "Minimum number of multiplications is 18000" and "Time difference is: 6.1011e-05".

```
Minimum number of multiplications is 18000
Time difference is: 6.1011e-05
```

A terminal window with a black background and yellow text. It shows the output of the program: "Minimum number of multiplications is 18" and "Time difference is: 9.1473e-05".

```
Minimum number of multiplications is 18
Time difference is: 9.1473e-05
```

Batch Analysis

Source Code

```
#include <bits/stdc++.h>
#include <chrono>

using namespace std;

int dp[100][100];

int matrixChainMemo(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], matrixChainMemo(p, i, k) + matrixChainMemo(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 matrixChainMemo(p, i, j);
}

void printArray(double arr[], int size)
{
    for (int i = 0; i < size; i++)
        cout << arr[i] << ", ";
    cout << endl;
}

void printArray(int arr[], int size)
{
    for (int i = 0; i < size; i++)
        cout << arr[i] << ", ";
    cout << endl;
}
```

```
}

void printArray(float arr[], int size)
{
    for (int i = 0; i < size; i++)
        cout << arr[i] << ", ";
    cout << endl;
}

int main(){
    double times[10];
    int ns[10];
    for (int x = 0; x < 10; x++)    {
        memset(dp, -1, sizeof dp);
        int n = rand() % 100;
        ns[x] = n;

        int arr[n];
        for (int i = 0; i < n; i++) {
            arr[i] = rand() % 100;
        }

        auto start = chrono::high_resolution_clock::now();
        ios_base::sync_with_stdio(false);

        MatrixChainOrder(arr, n);

        auto end = chrono::high_resolution_clock::now();

        double time_taken = chrono::duration_cast<chrono::nanoseconds>(end -
start).count();

        times[x] = time_taken;
    }

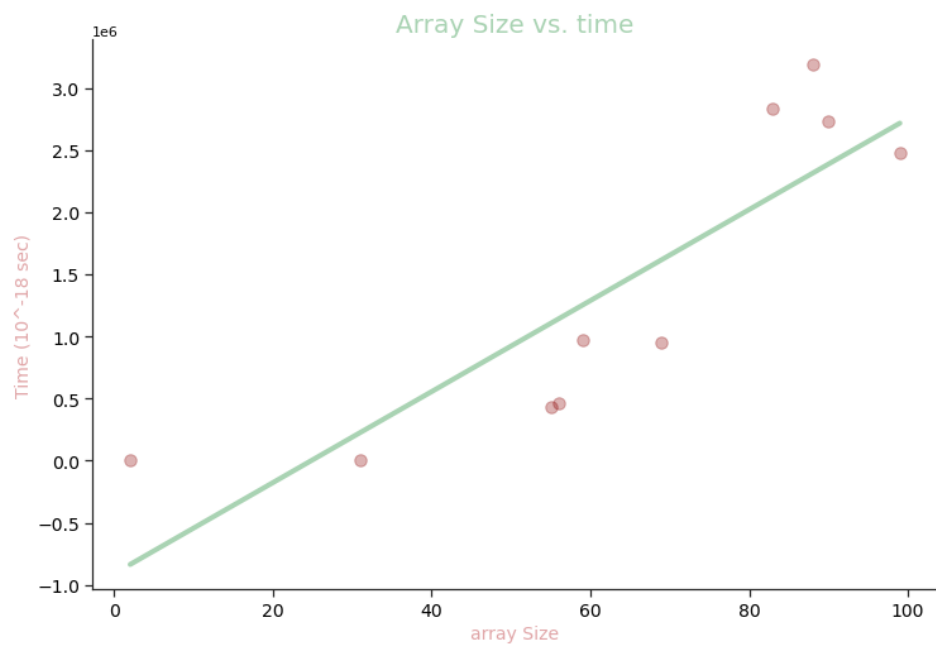
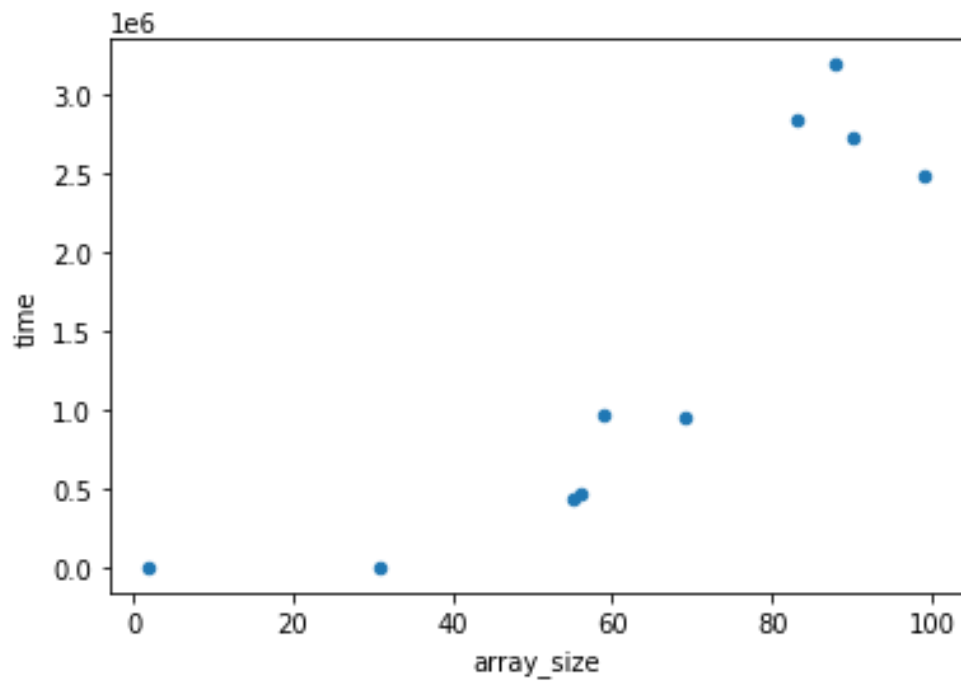
    cout << "value of n's: " << endl;
    printArray(ns, 10);

    cout << "time for each n: " << endl;
    printArray(times, 10);

    return 0;
}
```

Output:

```
input
value of n's:
83, 88, 56, 69, 90, 59, 2, 99, 55, 31,
time for each n:
2.83879e+06, 3.19062e+06, 460596, 953485, 2.72973e+06, 974584, 235, 2.48403e+06, 432541, 76248,
```



Viva Questions

1. What is the recurrence relation for MCM problem?

Ans.

Matrix Chain Multiplication + Dynamic Programming + Recurrence Relation

Given a sequence of matrices, find the most efficient way to multiply these matrices together. The problem is not actually to perform the multiplications, but merely to decide in which order to perform the multiplications.

We have many options to multiply a chain of matrices because matrix multiplication is associative. In other words, no matter how we parenthesize the product, the result will be the same. For example, if we had four matrices A, B, C, and D, we would have:

$$(ABC)D = (AB)(CD) = A(BCD) = \dots$$

However, the order in which we parenthesize the product affects the number of simple arithmetic operations needed to compute the product, or the efficiency. For example, suppose A is a 10×30 matrix, B is a 30×5 matrix, and C is a 5×60 matrix. Then,

$$(AB)C = (10 \times 30 \times 5) + (10 \times 5 \times 60) = 1500 + 3000 = 4500 \text{ operations}$$

$$A(BC) = (30 \times 5 \times 60) + (10 \times 30 \times 60) = 9000 + 18000 = 27000 \text{ operations.}$$

Clearly the first parenthesization requires less number of operations.

Given an array $p[]$ which represents the chain of matrices such that the i th matrix A_i is of dimension $p[i-1] \times p[i]$. We need to write a function `MatrixChainOrder()` that should return the minimum number of multiplications needed to multiply the chain.

2. Let A_1, A_2, A_3, A_4, A_5 be five matrices of dimensions $2 \times 3, 3 \times 5, 5 \times 2, 2 \times 4, 4 \times 3$ respectively. Find the minimum number of scalar multiplications required to find the product $A_1 A_2 A_3 A_4 A_5$ using the basic matrix multiplication method?

Ans.

We have many ways to do matrix chain multiplication because matrix multiplication is associative. In other words, no matter how we parenthesize the product, the result of the matrix chain multiplication obtained will remain the same. Here we have four matrices A_1, A_2, A_3 , and A_4 , we would have:

$$((A_1 A_2) A_3) A_4 = ((A_1 (A_2 A_3)) A_4) = (A_1 A_2) (A_3 A_4) = A_1 ((A_2 A_3) A_4) = A_1 (A_2 (A_3 A_4)).$$

However, the order in which we parenthesize the product affects the number of simple arithmetic operations needed to compute the product, or the efficiency. Here, A_1 is a 10×5 matrix, A_2 is a 5×20 matrix, and A_3 is a 20×10 matrix, and A_4 is 10×5 .

If we multiply two matrices A and B of order $l \times m$ and $m \times n$ respectively, then the number of scalar multiplications in the multiplication of A and B will be $l \times m \times n$.

Then,

The number of scalar multiplications required in the following sequence of matrices will be :

$$A_1 ((A_2 A_3) A_4) = (5 \times 20 \times 10) + (5 \times 10 \times 5) + (10 \times 5 \times 5) = 1000 + 250 + 250 = 1500.$$

All other parenthesized options will require number of multiplications more than 1500.

3. Consider the two matrices P and Q which are 10×20 and 20×30 matrices respectively. What is the number of multiplications required to multiply the two matrices?

Ans.

The number of multiplications required is $10 \times 20 \times 30$.

4. Consider the matrices P, Q and R which are 10 x 20, 20 x 30 and 30 x 40 matrices respectively. What is the minimum number of multiplications required to multiply the three matrices?

Ans.

The minimum number of multiplications are 18000. This is the case when the matrices are parenthesized as $(P*Q)*R$.

5. Can this problem be solved by greedy approach. Justify?

Ans.

Unfortunately, **there is no good "greedy choice"** for Matrix Chain Multiplication, meaning that for any choice that's easy to compute, there is always some input sequence of matrices for which your greedy algorithm will not find the optimum parenthesization.

At each step only least value is selected among all elements of in the array $p[x, y]$, so that the multiplication cost is kept minimum at each step. This greedy approach ensures that the solution is optimal with least cost involved and the output is a fully parenthesized product of matrices.