Joseph Williams, Joshua Chen

Iterative Algorithms:

Algorithm-1 (Naive Matrix Multiplication):

- **Description:** Multiplies two matrices using a triple nested loop.
- **Calculations and Derivations:**
  - **Outer loop**
    - **Runs from i = 0 to i = p – 1**
    - **So, loops run p times.**
  - **Middle loop**
    - **Runs from j = 0 to j = r –1**
    - **So, loops run r times.**
  - **Inner loop**
    - **Runs from k = 0 to k = q – 1**
    - **So, loop runs q times.**
  - **Each iteration of the inner loops performs a constant amount of work.**
  - **Innermost Operation is simple multiplication and addition, so O(1).**
  - **Total number of operations is p x r x q**
- **Polynomial**
  - **For this iterative algorithm, the polynomial T(n) is**

    T(n) = 2* p x r x q = 2n^3
- **Complexity Order**
  - **The complexity Order is O(p x r x q). If p = q = r = n then it simplifies to O(n^3).**

Algorithm-2 (Blocked Matrix Multiplication):

- **Description:** Uses block matrix multiplication to improve cache performance.
- **Calculations and Derivations:**
  - **Outer loop (I)**
    - **Runs from I= 0 to I < p in steps of T**
    - **So, loops run p/T times.**
  - **Second Outer loop (J)**
    - **Runs from J= 0 to J < r in steps of T**
    - **So, loops run r/T times.**
  - **Third Outer loop (K-loop):**
    - **Runs from K = 0 to K < q in steps of T**
    - **So, loop runs q/T times.**

- **Inner Loop (i-loop):**
  - **Runs from i = I to i < min(I + T,p).**
  - **So, loops run at most T times, if T divides p perfectly.**
- **Second Inner Loop (i-loop):**
  - **Runs from j = J to j < min(J + T,r).**
  - **So, loops run at most T times.**
- **Innermost Loop (i-loop):**
  - **Runs from k = K to k < min(K + T,q).**
  - **So, loops run at most T times.**
- **Total number of iterations for the outermost loops is about: p/T x r/T x q/T**
- **Total number of iterations for the inner loops is about**
- **Total number of operations is**

[p/T x r/T x q/T] x [T x T x T] = [p x r x q]/T^3 * T^3 = p x r x q

- **Polynomial**
  - **For this iterative algorithm, the polynomial T(n) is**

    T(n) = 2* p x r x q = 2n^3
- **Complexity Order**
  - **The complexity Order is O(p x r x q), If p = q = r = n then it simplifies to O(n^3).**

Recursive Algorithms:

Algorithm-3 (Recursive Matrix Multiplication):

- **Description:** Uses a divide-and-conquer approach to split the matrices and solve smaller subproblems.
- Analysis
  - **Base Case**
    - **If any dimension is 1, the algorithm calls 'algorithm_1', which has a time complexity of O(p x q x r).**
  - **Recursive Case**
    - **Algorithm 3 splits the matrix depending on the largest dimension:**
    - **If p is largest, splits A horizontally.**
    - **If r is largest, splits B vertically.**
    - **If same splits both.**
- **Recurrence Relation**
  - **If p or q or r = 1, then T(1) = 0**

- **When p is the largest dimension:**
  - **T(p, q, r) = 2T (p/2, q, r) + O(p x q x r)**
- **When r is the largest dimension:**
  - **T(p, q, r) = 2T(p,q,T/2) = O(p x q xr)**
- **When neither p nor r is the largest dimension, both are split:**
  - **T(p, q, r) = 4T (p/2, q, r/2) + O(p x q x r)**

- **Master Theorem**
  - **P is the larger dimension**
    - **The recurrence relation is**
      T(p, q, r) = 2T(p/2, q, r) + O(p x q x r)

    T(n) = 2* p x r x q = 2n^3

    - **Matches the form T(n) = aT(n/b) + f(n) with a =a and b= 2, and f(n) = O(n^3).**
    - **Since f(n) is larger than n^log$_b$a = n^ log$_2$2 = n,w e fall into case 3 of the Master Theorem: T(n) = O(f(n)) = O(n^3)**
  - **r is the larger Dimensions**
    - **Matches the form T(n) = aT(n/b) + f(n) with a =a and b= 2, and f(n) = O(n^3).**
    - **Since f(n) is larger than n^log$_b$a = n^ log$_2$2 = n,w e fall into case 3 of the Master Theorem again: T(n) = O(f(n)) = O(n^3)**
  - **Both Dimensions**
    - **a=4, b = 2 and f(n) = O(n^3). Compare f(n) with n^log$_b$a = n^log$_2$4 = n^2**
    - **F(n) = O(n^3) is larger than n^2**
      - **O(p x q x r) when p = q = r, T(n) = O(f(n)) = O(n^3).**

Algorithm-4 (Strassen's Algorithm):

- **Description:** Uses a more advanced divide-and-conquer approach to multiply matrices, reducing the number of multiplications required.
- **Analysis**
  - **Base case**
    - **If the matrix size n is 1 or less, it performs elementwise multiplication and addition.**
  - **Recursive Case**
    - **The matrix is divided into submatrices of size n/2.**

        - **Severall matrices S1 to S10 are computed using addition and subtraction of submatrices.**
        - **Seven recursive multiplications are used to compute P1 to P7.**
        - **Its result is combined to form the final submatrices.**
- **Recurrence Relation**
    - **If p or q or r = 1, then T(1) = 0**
    - **When p is the largest dimension:**
        - **T(n) = 7T(n/2) + O(n^2)**
- **Master Theorem**
    - **For T(n) = 7T(n/2) + O(n^2)**
    - **A = 7 b = 2 f(n) = O(n^2)**
    - **Log$_b$a = log$_2$7 = 2.81**
    - **F(n) = O(n^2) and n^log$_b$a = n^2.81**
    - **Since f(n) = O(n^2) is polynomially smaller than n^2.81, its case 1.**

      T(n) = O(n^log$_b$a) = O(n^log$_2$7) = O(n^2.81)
    - **S1 to S10**
        - **Size O((n/2)^2) bc of matrix add or sub.**
    - **P1 to P7**
        - **Recursive calculate on matrices size n/2**
    - **Combine**
        - **O((n/2)^2)**
- **Total**
    - **T(n) = 7T (n/2) + 10 * O((n/2)^2) + O(n^2)**
    - **T(n) = 7T(n/2) + O(n^2)**
- **Complexity:**
    - **7(x/2)^3**
    - **Using the Master Theorem, T(n) = O(n^log$_2$7) ≈ O(n^2.81)**

Algorithm-5 (Variation of Strassen's Algorithm):
- **Description:** Uses a more advanced divide-and-conquer approach to multiply matrices, reducing the number of multiplications required.
- **Analysis**
    - **Base case**
        - **If the matrix size n is less than 3, it calls algorith_1 and performs matrix multiplication with time complexity O(p x q x r)..**
    - **Recursive Case**
        - **The matrix is divided into submatrices of size n/2 x n/2.**

        - **7 recursive multiplications are performed.**
        - **Its result is combined using matrix addition and subtraction to form the final submatrices.**
- **Recurrence Relation**
    - **If p or q or r = 1, then T(1) = 0**
    - **When p is the largest dimension:**
        - **T(n) = 7T(n/2) + O(n^2)**
- **Master Theorem**
    - **For T(n) = 7T(n/2) + O(n^2)**
    - **A = 7 b = 2 f(n) = O(n^2)**
    - **Log$_b$a = log$_2$7 = 2.81**
    - **F(n) = O(n^2) and n^log$_b$a = n^2.81**
    - **Since f(n) = O(n^2) is polynomially smaller than n^2.81, its case 1.**

      T(n) = O(n^log$_b$a) = O(n^log$_2$7) = O(n^2.81)
- **Complexity:**
    - **Using the Master Theorem, T(n) = O(n^log$_2$7) ≈ O(n^2.81)**

Dynamic Programming Algorithms:

Matrix-Chain-Order Algorithm:

- **Description:** Uses dynamic programming to find the optimal order of matrix multiplications.
- Analysis
    - calculates the minimum multiplication cost for every subproblem of matrices of the matrix chain, updating the cost into an array m and split-point s whenever the most efficient order is calculated (minimum amount of operations).
- **Recurrence Relation**
    - $$m[i,j] = \begin{cases} 0 & \text{if } i = j \,, \\ \min\{m[i,k] + m[k+1,j] + p_{i-1}p_k p_j : i \le k < j\} & \text{if } i < j \,. \end{cases}$$
- **Complexity:**
    - The number of subproblems is n^2 and each subproblem takes n time.
      So, the overall complexity is O(n^3).

Chain-Matrix-Multiplication Algorithm:

- **Description:** The chain matrix multiplication algorithm multiplies matrix chains by minimizing the number of multiplications using dynamic programming recursion from the matrix order chain outputs.
- Analysis
    - **Base Case**
        - If there is only one matrix to be multiplied it returns that matrix
    - **Recursive Case**
        - Split the matrices into smaller ones using s table from matrix order chain algorithm to have optimal calculations
        - Multiplies those matrices
- **Recurrence Relation**
    - $$m[i,j] = \begin{cases} 0 & \text{if } i = j \,, \\ \min\{m[i,k] + m[k+1,j] + p_{i-1}p_k p_j : i \le k < j\} & \text{if } i < j \,. \end{cases}$$
- **Time complexity**
    - **O(n^3)**

Iterative Algorithm:

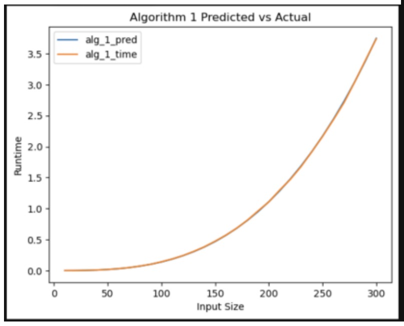
Sequential_matrix_mulitplication Algorithm:

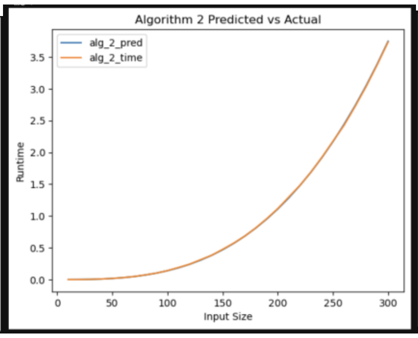
- **Description:** Uses dynamic programming to find the optimal order of matrix multiplications.
- **Analysis**
    - Sets product to the first matrix in matrix chain which will become the first result through the process of loops multiplication
    - While in the loop is records the dimensions of the product and then multiplies it using algorthim_1 to finally be stored in the result until all the matrices have been multiplied using the next variable
- **Polynomial**
    - **This iterative algorithm has the same polynomial as algorithm 1 along with a more constants to go through each matrix in the chain, so the general polynomial is T(n) is 2(p x r x q) = 2n^3**
- **Complexity**
    - So, the overall complexity O(n^3)

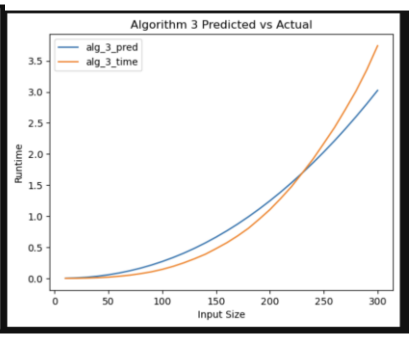
Experiment 1:

Did the actual/empirical time match each algorithm’s predicted/theoretical complexity as the input size increased? Why or why not?

In experiment 1, the provided graph compares the performance of sequential multiplication (blue line) and an optimal chain matrix multiplication algorithm (orange line). Both algorithms show an increasing trend in computation time as the matrix size grows. The actual and predicted complexity matched as input size increases. The actual times closely followed the theoretical predictions because the operations of the algorithms are consistent with their theoretical implementations. Slight variations may occur due to differences in computational speed. These factors can cause minor discrepancies compared to the idealized theoretical limits, especially as the input size increases.



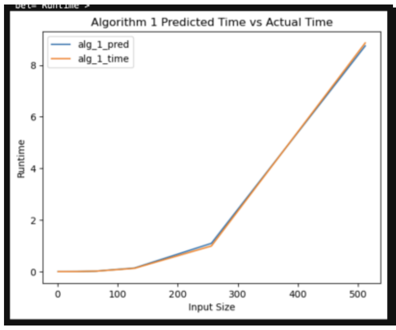


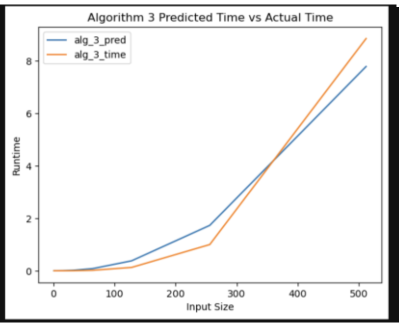


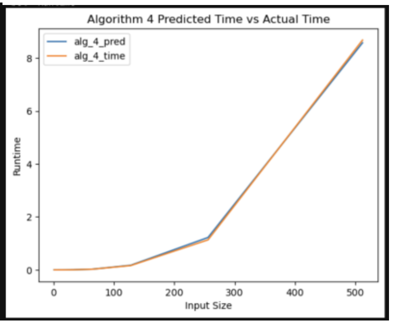
Experiment 2:

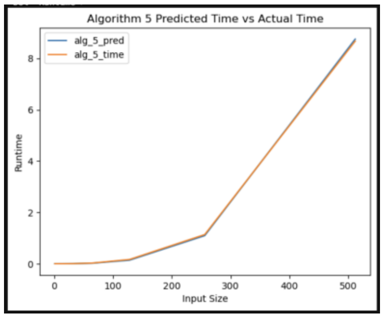
Did the actual/empirical time match each algorithm’s predicted/theoretical complexity as the input size increased? Why or why not?

In experiment 2, the provided graph compares the performance of sequential multiplication (blue line) and an optimal chain matrix multiplication algorithm (orange line). Both algorithms show an increasing trend in computation time as the matrix size grows. The actual and predicted either exactly matched or were very close with caveat that there is some more variation than in experiment 1. The actual times closely followed the theoretical predictions because the operations of the algorithms are consistent with their theoretical implementations. Slight variations may occur due to differences in computational speed. These factors can cause minor discrepancies compared to the idealized theoretical limits, especially as the input size increases.





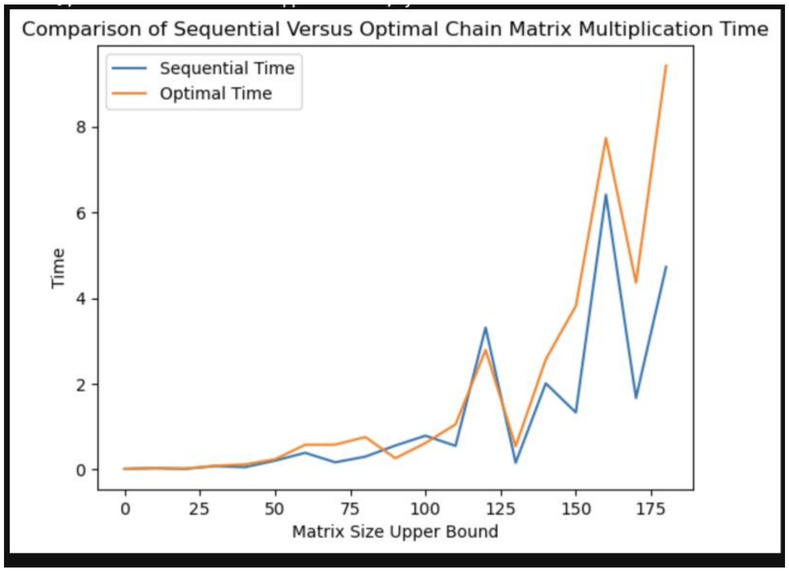




Matrix chain problem:

In the chain-matrix multiplication problem, does the optimal solution outperform the sequential multiplication, and by how much?

While the theoretical optimal solution should generally provide better performance for large matrix chains by reducing the number of multiplications, the unusual performance could be attributed to factors such as not a large enough input size to see the optimal time, input variability due to the randomness of the values, maybe overhead in the complexity of the optimal algorithm versus the simplicity in the sequential in smaller input sizes, and the potential for coding errors could also lead to the observed variations in empirical performance, despite thorough checking of the actual correct algorithm from the textbook. Since we don't have the optimal solution outperforming the sequential approach on average, we can't quantify the extent of its superiority except for isolated, uncommon points. The graph shows that the optimal solution (orange line) does not consistently outperform the sequential solution (blue line), with the performance varying significantly and the sequential solution sometimes being faster; this indicates that the theoretical benefits of the optimal solution are not consistently realized in practice again maybe due to factors like overhead and implementation details.


Comparison of Sequential Versus Optimal Chain Matrix Multiplication Time
Sequential Time
Optimal Time
Time
Matrix Size Upper Bound