Matrices (Repetition)

- A matrix is a rectangle of numbers. For example:

  $$\begin{bmatrix} 1 & 6 & 4 \\ 2 & 5 & 7 \\ 9 & 1 & 1 \end{bmatrix}$$

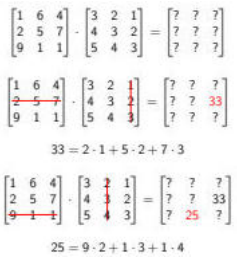
- The above is a 3 × 3 matrix. Today, all matrices are $n \times n$ square matrices, where n indicates the side length of the matrices.

Plus for Matrices

- Matrix addition involves adding corresponding elements of two matrices.
  $$\begin{bmatrix} 1 & 6 & 4 \\ 2 & 5 & 7 \\ 9 & 1 & 1 \end{bmatrix} + \begin{bmatrix} 3 & 2 & 1 \\ 4 & 3 & 2 \\ 5 & 4 & 3 \end{bmatrix} = \begin{bmatrix} 1+3 & 6+2 & 4+1 \\ 2+4 & 5+3 & 7+2 \\ 9+5 & 1+4 & 1+3 \end{bmatrix} =$$
- Time complexity for matrix addition: $\Theta(n^2)$. This is optimal since the output is of size n^2.

Matrix Multiplication

- Matrix Multiplication:

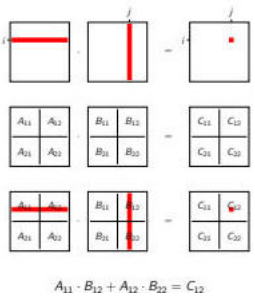


  The image illustrates matrix multiplication for 3x3 matrices, showing how each element in the resulting matrix is derived from the elements of the original matrices. Red lines and numbers highlight specific calculations, such as how 33 and 25 are calculated.

- Time complexity: $\Theta(n^3)$. Is this optimal? Are there alternative algorithms?

Recursive Algorithm for Multiplication?

- Recursive Algorithm for Matrix Multiplication:



  This image provides a visual representation of multiplying two 2x2 matrices. It breaks down the matrices into individual elements, labeled A_{11} to A_{22} and B_{11} to B_{22}, and demonstrates how these elements are combined to form the resulting matrix elements C_{11} to C_{22}.

- Matrix addition: $O(n^2)$
- Matrix multiplication: Recursive call to matrix multiplication on $n/2 \times n/2$ matrices. Base case: $n = 1 \implies$ multiplication of numbers.

$T(n) = 8T(n/2) + n^2$

Recursive Algorithm Analysis

- Given the recurrence relation:

  $T(n) = 8T(n/2) + n^2$

  We can analyze its time complexity using the Master Theorem.

- Master Theorem:
  - $a = 8, b = 2, f(n) = n^2$
  - $\alpha = \log_b(a) = \log_2(8) = 3$
  - $f(n) = n^2 = O(n^{\alpha - 0.1}) \implies$ Case 1
  - $T(n) = \Theta(n^\alpha) = \Theta(n^3)$

- This is the same as the standard algorithm. Exercise: Strassen [1969]

Strassen Algorithm [1969]

- Compute the following intermediate values:

  $S_1 = B_{12} - B_{22}$ $S_2 = A_{11} + A_{12}$ $S_3 = A_{21} + A_{22}$ $S_4 = B_{21} - B_{11}$ $S_5 = A_{11} + A_{22}$ $S_6 = B_{11} + B_{22}$ $S_7 = A_{12} - A_{22}$
  $S_8 = B_{21} + B_{22}$ $S_9 = A_{11} - A_{21}$ $S_{10} = B_{11} + B_{12}$

  Time: $O(n^2)$, as both addition and subtraction take this time.

- Compute the following products recursively:

  $P_1 = A_{11} \times S_1$ $P_2 = S_2 \times B_{22}$ $P_3 = S_3 \times B_{11}$ $P_4 = A_{22} \times S_4$ $P_5 = S_5 \times S_6$ $P_6 = S_7 \times S_8$ $P_7 = S_9 \times S_{10}$

  7 recursive calls to matrix multiplication on $n/2 \times n/2$ matrices.

- Verify that the following holds:

  $P_5 + P_4 - P_2 + P_6 = A_{11} \times B_{11} + A_{12} \times B_{21}$ $P_1 + P_2 = A_{11} \times B_{12} + A_{12} \times B_{22}$ $P_3 + P_4 = A_{21} \times B_{11} + A_{22} \times B_{21}$
  $P_5 + P_1 - P_3 - P_7 = A_{21} \times B_{12} + A_{22} \times B_{22}$

  I.e., the output can be computed in $O(n^2)$ time from $P_1, \ldots, P_7$, since:

  $A_{11} \times B_{11} + A_{12} \times B_{21} = C_{11}$ $A_{11} \times B_{12} + A_{12} \times B_{22} = C_{12}$ $A_{21} \times B_{11} + A_{22} \times B_{21} = C_{21}$ $A_{21} \times B_{12} + A_{22} \times B_{22} = C_{22}$

- Thus, $T(n) = 7T(n/2) + n^2$

Strassen Algorithm [1969] Time Complexity

- Given $T(n) = 7T(n/2) + n^2$

- Master theorem:
  - $a = 7, b = 2, f(n) = n^2$
  - $\alpha = \log_b(a) = \log_2(7) \approx 2.80735$
  - $f(n) = n^2 = O(n^{\alpha - 0.1}) \implies$ Case 1
  - $T(n) = \Theta(n^\alpha) = O(n^{2.81})$

- This is better than the standard algorithm!