Topological Sorting
Strongly Connected Components
Advanced Recurrences
Topological Sorting

• Dependency Graphs

• For well defined dependency graphs (no cycle, i.e a DAG), process nodes in reverse topological order
Strongly Connected Components

Example from lecture slides
Divide and Conquer Strategy

• Strategy is to break given problem into smaller sub-problems
• Combine the solutions to the sub-problems to get solution to the original problem (these sub-problem are usually $\left(\frac{1}{b}\right)^{th}$ of the original problem size)

• Binary Search
  • Find number $x$ in array $A$
  • Check middle element (say $M$)
  • If $x>M$, search in right sub-array
  • If $x<M$, search in left sub-array
Recurrences in Algorithm Analysis

• Let running time of an algorithm be $T(n)$
• Say size of original instance is $n$
• Break problem into $a$ instances of size $\frac{n}{b}$ each
• Combine solutions to smaller instances by doing $f(n)$ extra work

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$
Example: Merge Sort

- Let $T(n)$ be running time of Merge Sort
- Given the array $A[1, n]$
- Sort the sub-arrays $A[1, \frac{n}{2}], A[\frac{n}{2}, n]$ independently using merge sort
- Merge the two sorted sub-arrays
- Time taken: $T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + cn$ (linear time to merge)
Methods to Solve

- Iterative Method
  - Keep expanding the expression and solve algebraically

- Tree Method
  - Draw the recursion tree and find the time taken at each level and sum

- Master Theorem
Iterative Method

• Expand the recurrence repeatedly

• Example $T(n) = 2T(n/2) + 1$
Tree Method

• Make a recurrence tree of the function calls according to the recurrence equation

• Mark the nodes by amount of time taken in joining the solutions i.e. $f(n)$

• Sum across nodes at a particular height and then sum across all heights to find total time
\[ n / 2 + n / 2 \]

\[ n / 4 + n / 4 + n / 4 + n / 4 \]

\[ \log n \text{ terms} \]
Example

• $T(n) = 2T\left(\frac{n}{2}\right) + n$
Master Theorem

- $T(n) = aT\left(\frac{n}{b}\right) + f(n)$

- If $f(n) = O(n^c)$, $c < \log_b a \Rightarrow T(n) = \Theta(n^{\log_b a})$

- If $f(n) = \Theta(n^c \log^k n)$, $c = \log_b a \Rightarrow T(n) = \Theta(n^c (\log n)^{k+1})$

- If $f(n) = O(n^c)$, $c > \log_b a \Rightarrow T(n) = \Theta(f(n))$
  - $f(n)$ must satisfy $af\left(\frac{n}{b}\right) < c f(n) \forall n > n_0$ for some constants $c, n_0$
Examples:

• \( T(n) = 2T\left(\frac{n}{2}\right) + n\log n \)

• \( T(n) = 3T\left(\frac{n}{4}\right) + n \)

• \( T(n) = 3T\left(\frac{n}{2}\right) + n^2 \)
Master Theorem: Proof
Median/Selection
Matrix Multiplication
Master Theorem

\[ T(n) = aT \left( \frac{n}{b} \right) + O(n^d) \]

\[ T(n) = O(n^d) \quad \text{if } d > \log_b a \]

\[ T(n) = O(n^d \log(n)) \quad \text{if } d = \log_b a \]

\[ T(n) = O(n^{\log_b a}) \quad \text{if } d < \log_b a \]
Master Theorem: Proof

- Assume $n$ is a power of $b \rightarrow$ can ignore rounding in $\lceil n/b \rceil$
- Subproblem size decreases by a factor of $b$ with each level of recursion $\rightarrow$ reaches base case after $\log_b n$ levels

- Branching factor $a \rightarrow k^{th}$ level of tree has $a^k$ subproblems, each of size $n/b^k$
- Total work done at $k^{th}$ level: $a^k \times O\left(\frac{n}{b^k}\right)^d = O\left(n^d\right) \times \left(\frac{a}{b^d}\right)^k$
Master Theorem: Proof

• Total work done at $k^{th}$ level:
  \[ a^k \times O\left(\frac{n}{b^k}\right)^d = O\left(n^d\right) \times \left(\frac{a}{b^d}\right)^k \]

• As $k$ goes from 0 (root) to $\log_b n$ (leaves) → geometric series with ratio $\frac{a}{b^d}$

• $\frac{a}{b^d} < 1$ → sum is $O(n^d)$

• $\frac{a}{b^d} > 1$ → sum is given by last term, $O(n^{\log_b a})$

• $\frac{a}{b^d} = 1$ → sum is $O(\log(n))$ terms, all equal to $O(n^d)$
Median/Selection

• Input: A list of $n$ numbers $S$; an integer $k$

• Output: The $k$th smallest element of $S$

• Expected running time: $T(n) = T(3n/4) + O(n)$
Matrix Multiplication

• Input: $n \times n$ matrices $X$ and $Y$

• Output: $n \times n$ matrix $Z$
Matrix Multiplication

- $X = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$, $Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix}$

- $XY = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$

- Running time: $T(n) = 8T(n/2) + O(n^2)$
Matrix Multiplication

• $XY = \begin{bmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{bmatrix}$

• $P_1 = A(F - H); \quad P_2 = (A + B)H; \quad P_3 = (C + D)E$
  $P_4 = D(G - E); \quad P_5 = (A + D)(E + H)$
  $P_6 = (B - D)(G + H); \quad P_7 = (A - C)(E + F)$

• Running time: $T(n) = 7T(n/2) + O(n^2)$