

# 1 Math Fundamentals

## Formulae

**Asymptotics** Let  $f$  and  $g$  be non-negative functions, then  $f(n)$  is...

	...if and only if...	if $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)}$
$O(g(n))$	$\exists c, N : f(n) \leq c \cdot g(n) \text{ for } n \geq N$	$\neq \infty$
$o(g(n))$	$\forall c, \exists N : f(n) \leq c \cdot g(n) \text{ for } n > N$	$= 0$
$\Omega(g(n))$	$\exists c, N : f(n) \geq c \cdot g(n) \text{ for } n \geq N$	$\neq 0$
$\omega(g(n))$	$\forall c, \exists N : f(n) \geq c \cdot g(n) \text{ for } n > N$	$= \infty$
$\Theta(g(n))$	$f(n) \text{ is } O(g(n)) \text{ and } g(n) \text{ is } O(f(n))$	$\neq 0, \infty$

Implications	Equivalences
$f = o(g) \Rightarrow f = O(g)$	$f = O(g) \Leftrightarrow g = \Omega(f)$
$f = \omega(g) \Rightarrow f = \Omega(g)$	$f = o(g) \Leftrightarrow g = \omega(f)$
	$f = O(g) \text{ and } f = \Omega(g) \Leftrightarrow f = \Theta(g)$

**Generalizations:** For all  $a, b > 0, k > 1$ , and  $n \geq 1$

$$\begin{cases} (\log n)^b = o(n^a) \\ n^b = o((1+a)^n) \\ a^{\sqrt{\log n}} = o(n^b) \\ k^n > n! > n^b > n^a > \log n > n^{1/k} > c \approx \text{comparisons} \\ n! \approx n^n \cdot e^{-n} \cdot \sqrt{2\pi n} \end{cases} \quad \text{Stirling's Formula}$$

**Master Theorem:** Let  $T(n) = aT(n/b) + cn^k$  for  $a \geq 1, b \geq 2, c, k \geq 0$ . Then  $T(n)$  is

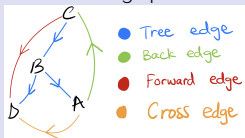
$$\begin{cases} \Theta(n^k) & \text{if } a < b^k \\ \Theta(n^k \log n) & \text{if } a = b^k \\ \Theta(n^{\log_b a}) & \text{if } a > b^k \end{cases}$$

# 2 Graph Algorithms

## Depth First Search Algorithm

**DFS Algorithm** **Runtime:**  $O(|V| + |E|)$

- Goes as deep as possible in the graph then backtrack.
- (In-class algorithm) marks preorder and postorder numbers of each node.
- In a directed graphs, DFS will label edges as follows. No cross edges in undirected graphs.



**Applications:**

Detecting cycles  
Topological Sort on DAGs  
Strongly Connected Components

$\exists (u, v): \text{postorder}(u) < \text{postorder}(v)$   
Decreasing post-order  
Hint: DFS on reversed Graph.

## Single Source Shortest Paths

**BFS Algorithm** **SSSP for unweighted graphs only.**

- Goes level by level in the graph.
- Uses a queue to process nodes.

**Runtime:**  $O(|V| + |E|)$

Let **update(u, v)** be defined as: if  $\text{Dist}[u] + \text{length}(u, v) < \text{Dist}[v]$ , update:  $\text{Prev}[v] = u$  and  $\text{Dist}[v] = \text{Dist}[u] + \text{length}(u, v)$ .

**Dijkstra's Algorithm:** **only +ve weighted graphs**

- Let  $\text{Dist}[v] = \infty$  and  $\text{Prev}[v] = \text{NIL}$  for all vertices  $v$ .
- Let  $\text{Dist}[s] = 0$  and initialize MinHeap with  $(s, 0)$ .
- While heap isn't empty, keep popping the vertex (say  $u$ ) with the smallest distance.
- For each neighbor  $v$  of  $u$  with weight  $w$ , **update(u, v)**.

**Runtime:**  $O(|V| * \text{popMin} + |E| * \text{Insert})$

**Bellman-Ford Algorithm:** **Only + and -ve weighted graphs**

- Let  $\text{Dist}[v] = \infty$  and  $\text{Prev}[v] = \text{NIL}$  for all vertices  $v$ .
- Let  $\text{Dist}[s] = 0$ .
- For  $|V| - 1$  times: For each edge  $(u, v)$ , **update(u, v)**
- Checking for negative weight cycles: repeat the above step once. If any distance can still be improved, a negative weight cycle exists. Hence, return **Inconclusive**

**Runtime:**  $O(|V| * |E|)$

**Linear Runtime:** **only for Directed Acyclic Graphs (DAGs)**

- Run DFS to get topological sort.
- For every edge  $(u, v)$ , in topological sort, **update(u, v)**.

**Runtime:**  $O(|V| + |E|)$

## Min Heaps

**Representation:** Visually a complete tree. Implementationwise, let  $A[0..n-1]$  be a list where  $A[0]$  is the root and for any  $i$ th element, its parent, left, and right children are at  $\lfloor i/2 \rfloor, 2i, 2i+1$  respectively.

**Heap Property:** The parent element is smaller than its children.

Operations	Description
$\text{Insert}(a)$	let $A[n] = a$ and $\text{HeapifyUp}(n)$
$\text{PopMin}$	let $A[0] = A[n-1]$ and $\text{HeapifyDown}(0)$
$\text{HeapifyUp}(i)$	repeatedly swap $A[i]$ with its parent until the heap property is restored
$\text{HeapifyDown}$	repeatedly swap $A[i]$ with its smallest child until the heap property is restored

**Heaps Operations and Runtimes:** Both  $O(\log n)$  with binary heaps.

**Note:** Can do better with Fibo-Heaps (amortized  $O(1)$  for PopMin.)

## Minimum Spanning Trees

**Basic Properties:**

- a Tree** is connected, acyclic, and has  $|V| - 1$  edges (any two implies the third).
- Cut Property** states that for any cut of a connected, undirected graph, the minimum weight edge that crosses the cut belongs to the MST.
- Only for connected, undirected, and weighted (non-negative) graphs.

**Prim's Algorithm:**

- Start with a single vertex and greedily add closest vertices.
- Similar to Dijkstra's algorithm, but  $\text{dist}[v]$  is the weight of the edge connecting  $v$  to the MST instead of the distance from  $s$ .

**Runtime:**  $O(|E| \log |V|)$  with Fibo-heaps

**Kruskal's Algorithm:**

- Sort edges in ascending order of weight.
- Repeatedly add the lightest edge that does not create a cycle until we have  $|V| - 1$  edges.

**Runtime:**  $nT(\text{Union}) + mT(\text{Find}) + T(\text{Sort } m \text{ Edges})$ .

**Notes:** Implemented using a union-find data structure.

## Disjoint Forest Data Structure

Maintain disjoint sets that can be combined ("unioned") efficiently. Operations  $\text{MakeSet}(x)$ ,  $\text{Find}(x)$ , and  $\text{Union}(a, b)$

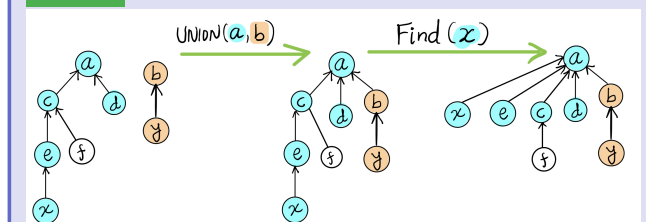
**Runtime:** An sequence of  $m$  UNIONS and  $n$  FINDs operations take  $O((m+n) \log^* n)$ .

**Note:**  $\log^* n$  is the number of times we can  $\log_2 n$  until we get  $\leq 1$ .

**Heuristics:**

- Union by rank. When performing a union operation, we prefer to merge the shallower tree into the deeper tree.
- Path compression. After performing a find operation, attach all the nodes touched directly onto the root of the tree.

**Example:**



### 3 Greedy Algorithms

**Main idea:** At each step, make a locally optimal choice in hope of reaching the globally optimal solution.

#### Horn Formula

**Algorithm:** Set all variables to false and greedily set ones to be true when forced to.

**Runtime:** linear time in the length of the formula (i.e., the total number of appearances of all literals).

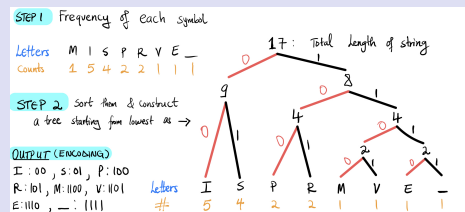
**Notes:** Only works for SAT instances where in each clause, there is at most one positive literal.

#### Huffman Coding

**Algorithm:** **Runtime:**  $O(n \log n)$

Find the best encoding by greedily combining the two least frequently items. Optimal in terms of encoding one character at a time.

**Example:** A Huffman tree for string "Mississippi River"



#### Set Cover

Given  $X = \{x_1, \dots, x_n\}$ , and a collection of subsets  $S$  of  $X$  such that  $\bigcup_{S \in \mathcal{S}} S = X$ , find the subcollection  $\mathcal{T} \subseteq \mathcal{S}$  such that the sets of  $\mathcal{T}$  cover  $X$ .

**Algorithm:** **Runtime:**  $O(|U|)$

1. Greedily choose the set that covers the most number of the remaining uncovered elements at the given iteration.

**claim:** Let  $k$  be the size of the smallest set cover for the instance  $(X, \mathcal{S})$ . Then the greedy heuristic finds a set cover of size at most  $k \ln n$ .

**Note:**

Not always optimal; achieves  $O(\log n)$  approximation ratio.

### 4 Divide and Combine

**Main Idea:** Divide the problem into smaller pieces, recursively solve those, and then combine their results to get the final result.

#### Famous Examples w/ Runtimes

Mergesort	$O(n \log n)$
Min and Max on a line	$\frac{3}{2}n - 2$ comparisons; $O(n)$ runtime.
Closest Pair of Points	$O(n \log^2 n)$

#### $n$ -digit Integer Multiplication

standard Multiplication	$\Theta(n^2)$
3 products on $n/2$ digits	$\Theta(n^{\log_2 3}) = \Theta(n^{1.59})$
5 products on $n/3$ digits	$\Theta(n^{\log_3 5}) = \Theta(n^{1.46})$

#### $n \times n$ Matrix Multiplication

**Strassen's Algorithm:** **Runtime:**  $O(n^{\log_2 7})$

Divide into four submatrices, each of size  $n/2$  by  $n/2$ .

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

Find:  $P_1 = A(F - H)$ ,  $P_2 = (A + B)H$ ,  $P_3 = (C + D)E$ ,  $P_4 = D(G - E)$ ,  $P_5 = (A + D)(E + H)$ ,  $P_6 = (B - D)(G + H)$ ,  $P_7 = (C - A)(E + F)$ , then:

$AE + BG = -P_2 + P_4 + P_5 + P_6$  and  $AF + BH = P_1 + P_2$   
 $CE + DG = P_3 + P_4$  and  $CF + DH = P_1 - P_3 + P_5 + P_7$

### 5 Dynamic Programming

**Main Idea:** Maintain a lookup table of correct solutions to subproblems and build up this table towards the actual solution.

**Steps:**

1. Define subproblems and recurrence to solve subproblems.
2. Combine with **reuse**.
3. Runtime and space analysis.

#### Edit Distance

Find the minimum number of operations required to transform one string,  $A[1 \dots n]$ , into another,  $B[1 \dots m]$ .

**Algorithm:** **Runtime and Space:**  $O(nm)$

1. Subproblem: let  $D(i, j)$  represent the edit distance between  $A[1 \dots i]$  and  $B[1 \dots j]$ .
2. Recurrence is:  
Base cases:  $D(i, 0) = i$ ,  $D(0, j) = j$ .  
 $D(i, j) = \min[D(i - 1, j) + 1, D(i, j - 1) + 1, D(i - 1, j - 1) + (1 \text{ if } i = j, 0 \text{ otherwise})]$ .
3. return  $D(n, m)$ .

#### All Pairs Shortest Paths

Given a graph  $G$  with  $n$  vertices and  $m$  edges, calculate distances of the shortest paths between every pair of nodes.

**Floyd-Warshall Algorithm:** **Runtime:**  $O(n^3)$

1. Subproblem: let  $D_k[i, j]$  represent the shortest path between  $i$  and  $j$  using only nodes in  $[1 \dots k]$ .
2. Recurrence is:

$$\begin{cases} D_0[i, j] = d_{ij} \text{ if } i \text{ and } j \text{ are connected, } \infty \text{ otherwise.} \\ D_k[i, j] = \min(D_{k-1}[i, j], D_{k-1}(i, k) + D_{k-1}[k, j]). \end{cases}$$

3. return  $D(i, j, n)$ .

**Notes:** Does not work for cyclic graphs.

#### (in-class) DP Examples

String Reconstruction  
Longest Common Subsequence  
Longest Increasing Subsequence  
Edit Distance  
Knapsack

Link to leetcode Follow-up  
Word Break  
1143  
300  
72  
Coin Change