

CS 170 Cheat Sheet

Big O notation

$f, g \in \mathbb{N}$, $f = O(g)$ means that f grows no faster than g if $\exists c > 0$
s.t. $F(n) \leq cg(n)$
 $f = \Theta(g)$ means $g = O(f)$
 $f = \Theta(g)$ IFF $f = O(g)$ & $g = \Theta(f)$

Master Theorem

Given: $T(n) = a \times T(\frac{n}{b}) + O(n^d)$

- a) $O(n^d)$ if $d > \log_b(a)$
- b) $O(n^d \log(n))$ if $d = \log_b(a)$
- c) $O(n^{\log_b(a)})$ if $d < \log_b(a)$

Graph Algorithms

DFS: $O(V + E)$

Guaranteed to visit every node reachable by v before returning from v . Can create topological sort of DAG.

BFS: $O(V + E)$

Used to find shortest path through an unweighted graph.

Dijkstras: $O((V + E) \log V)$

Like BFS but with priority queue, used to find shortest path between two nodes on a weighted graph.

Bellman Ford: $O(V E)$

Find shortest paths with negative edges as long as there are no negative cycles. Runs $V - 1$ updates on all E edges.

Kruskal: $O(E \log(V))$

Use the disjoint set trees to add edges in ascending order that don't complete a cycle. Used to find MST.

FFT

It is a black box which represents 2 polynomials as a list of points and then multiplies them together to create a new polynomial. Takes $O(N \log N)$ time. Uses roots of unity to determine where to multiply two polynomials together.

N^{th} Roots of Unity can be found by: $\cos(\frac{2\pi j}{n}) + i \cdot \sin(\frac{2\pi j}{n})$

P/NP Basics

P: search problem that can be solved in polynomial time.

NP: search problem that can be checked to be correct in polynomial time.

NP complete: search problem that is at least as hard as every other NP complete problem. Basically it reduces to circuit sat, could possibly be solved in polynomial time but we doubt it.

NP hard: there exists NP complete Y such that Y is reducible to X but can't go the other way around. Not actually in NP.

Reductions

A search problem is NP-complete if all other search problems reduce to it.

To reduce X to Y means to find a solution for X using Y .

