1 Useful Algorithms

Name	Description	Runtime	Notes
Merge Sort	Recursively divide list in half, sort each half, and then merge	$O(n \log n)$	This is a divide and conquer algorithm. The iterative version uses a queue and the recursive version uses a stack.
Breadth-First Search (DFS)	Visits all vertices in increasing order of distance from source s. Maintain a queue of unvisited vertices, once v is popped from the queue, mark as visited and for all unvisited w reachable from v, set $dist(w) = dist(v) + 1$.	$O(\|E\| + \|V\|)$	One of the two basic graph search algorithms.
Depth-First Search (DFS)	Explore a graph using the stack as the basic data structure. Recursively search from a vertex v, exploring all unexplored vertices reachable from v.	$O(\|E\| + \|V\|)$	One of the two basic graph search algorithms. Can use pre- and post-visit numbers for various tasks like detecting back-edges in the graph.
Topological Sort	Let G be a DAG. Then topological sort gives an ordering $V = v_1, \ldots, v_n$ such that for all paths $p = (v_i, v_j)$, v_i comes before v_j in the ordering.	$O(\|E\| + \|V\|)$	Run DFS on DAG, process tasks in decreasing order of post-order number. If G is not a DAG, first detect all SCCs by performing DFS on G and G^R, then topologically sort the graph considering each SCC as a "mega-vertex."
Djikstra's Algorithm	Maintain a binary heap (other implementions work too) of unvisited nodes (and distances to these nodes) directly reachable from visited nodes. While the heap is non-empty, pop the vertex v with smallest distance from the heap, update distances for vertices w reachable from v and push onto heap if applicable.	$O(\|E\| \cdot insert + \|V\| \times deleteMin) = O(\|E\| \log \|V\|)$	Used to find shortest paths to all nodes v from source s. Does not work for graphs with negative weights.
Bellman-Ford Algorithm	Initialize $dist(s) = 0$ iterate through all edges up to $\|V\| - 1$ times, updating the distance to w if $dist(w) > dist(v) + w(v, w)$. If the $\|V\|$th update changes any updates, there exists a negative cycle.	$O(\|E\| \cdot \|V\|)$	Used to find shortest paths to all nodes v from source s. Works for negative weight graphs and can detect negative weight cycles.
Prim's Algorithm	Keep all the edges that cross the cut in a heap and repeatedly pop off the lightest one.	$O(\|E\| \log \|V\|)$	This is similar to Dijkstra's algorithm and has the same running time.
Kruskal's Algorithm	Sort the list of edges and repeatedly add the lightest one that connects two separate trees.	$O(\|E\| \log^* \|V\|)$	If the edges are not initially sorted, the running time is $O(\|E\| \log \|E\|)$ The iterated log in the runtime comes from the disjoint set data structure.
Disjoint Set (Union-Find)	A data structure that maintains a collection of disjoint sets. Useful for Kruskal.	m UNION and n FIND operations take $O((m + n) \log^* n)$ steps	FIND uses path compression, so whenever we FIND an element, we set its parent to point to the root. UNION uses union-by-rank, so we always combine trees in a way that minimizes the resulting depth.

Integer and Matrix Multiplication	(Integer multiplication) Let $x = 10^{n/2}a + b, y = x = 10^{n/2}c + d$. Then compute $ac, bd, (a+b)(c+d)$ and compute $ad + bc = (a+b)(c+d) - ac - bc$. Recursively divide and conquer until base case is reached.	$T(n) = 3T(n/2) + O(n)$, or $T(n) \approx O(n^{1.59})$	Strassen's algorithm for matrix multiplication with a similar trick, except by dividing the matrices into blocks and reducing eight multiplications into seven.
Selection	Use the median of medians approach to select pivots. Follow the quickselect algorithm, which partitions the list based on the pivot and recurses on a progressively smaller list.	$T(n) \leq T(\frac{n}{5}) + T(\frac{7n}{10}) + O(n)$ (selecting median of the $\frac{n}{5}$ medians), or $T(n) \approx O(n)$	This is a divide and conquer algorithm. The median of medians approach reduces our worst-case running time by guaranteeing that the recursive call is on a list of size $\leq \frac{7n}{10}$.
Greedy Horn	For each implication, if it is not satisfied, set the implied variable to true. If all the pure negatives are satisfied, return the assignment, else return "unsatisfiable".	$O(n)$ where n is the length of the formula	Each clause is either an *implication* (two negations and one positive literal) or a pure negative clause. 3SAT is NP-hard – this algorithm operates on restricted formulas with at most one positive literal per clause.
Huffman Coding	Repeatedly replace the two smallest symbols with a meta-character with frequency equal to the sum of their frequencies.	$O(n \log n)$	Any encoding must be prefix free, so for any two distinct strings x and y, $E(x)$ is not a prefix of $E(y)$.
Set Cover	Keep adding the subset that covers the most uncovered elements.	$O(n)$ where n is the number of elements	This is an approximation algorithm. If k is the size of the smallest set, this algorithm finds a cover of size at most $k \ln n$.
Bloom filter	Split m buckets into k "blocks" (other implementations exist). Each hash function H_i, $0 \leq i \leq k-1$ "controls" the kth block of buckets. To hash x, compute $H_1(x), \ldots H_k(x)$ and set buckets to 1. For set membership query, returns IN SET if all hashed buckets are set to 1 already.	$O(m)$ space, $P(collision) = (1 - e^{-nk/m})^k$	No false negatives, but false positives may occur (at lower rate than single hash function).
Document similarity	Construct a sketch of the document D by applying n different hash functions to a set A (e.g. k-shingles of D). Set the sketch $S_D = (min(\pi_1(A)), \ldots (min(\pi_n(A))))$, where each π_i is a random permutation of the hash values. For two documents corresponding to sets A, B, we know $P(min\pi_i(A) = min\pi_i(B)) = r = \frac{\lvert A \cap B \rvert}{\lvert A \cup B \rvert}$.	$P(\geq m \ matches \lvert \ n \ hash \ functions) = \sum_{k=m}^{n} \binom{n}{k} r^k (1-r)^{100-k}$	See resemblance notes (lecture 12) for more on why $P(min\pi_i(A) = min\pi_i(B)) = r$.
Fingerprinting	For all patterns of length $\lvert P \rvert$ in D, we want to compute a quick hash. Instead of hashing every pattern separately, note that if N and N' differ by two digits (so that the leftmost digit a from N is removed and the rightmost digit b is added to create N'. Note $N' = 10 \cdot (N = 10^{\lvert P \rvert - 1} \cdot a) + b$. So, to compute the hash, we choose a prime p and compute the update mod p.	For a random prime, the probability of a false match is at most $\frac{\lvert D \rvert \log_2 10^{\lvert P \rvert}}{\pi(Z)}$.	We can choose a set of k primes to reduce the probability of a false match (intuition similar to Bloom filters).

String Reconstruction	DP with recurrence $D(i,j) = \vee_{i+1,\ldots,j-1} D(i,k) \wedge D(k,j)$. If $D(i,j)$ is true, $A[i,...j]$ is the concatenation of words from the dictionary.	$O(n^3)$ where n is the length of the initial string	This is an example of dynamic programming. Remember the steps for DP: problem description, recurrence relation, time/space complexity. This algorithm uses $O(n^2)$ space.
Edit Distance	DP with recurrence $D(i,j)=$ $\min[D(i-1,j)+1,$ $D(i,j-1)+1,$ $D(i-1,j-1)+I(i \neq j)]$	$O(nm)$ time and space	DP, the base cases are $D(i,0)=i$ and $D(0,j)=j$.
Primality Testing	To test if n is prime, first compute $n-1=2^t u$. Pick a random base a. Repeatedly square a^u. Composite if $\exists i$ such that $a^{2^{i-1}u} \neq \pm 1 \bmod n$ and $a^{2^i u}=1 \bmod n$, or if $a^{n-1} \neq 1 \bmod n$. Repeat with k random choices of a.	$O(k \log n)$ multiplication operations.	This is a randomized algirithm, and we get a good error bound using *amplification*: The algorithm is wrong once with probability $1/4$ and wrong k times with probability $(1/4)^k$.
RSA	Bob finds two large primes p and q and a random integer e s.t. $\gcd((p-1)(q-1),e)=1$. (n,e) is his public key. His private key is (p,q,d) where $d=e^{-1}\bmod(p-1)(q-1)$. If Alice wants to send Bob the message m, she sends $e(m)=m^e \bmod n$. Bob decrypts the message by computing $d(e(m))=(e(x))^d \bmod n$	$O(\log e)$ and $O(\log d)$ multiplication operations for encryption and decryption respectively	The proof of correctness relies on Fermat's little theorem: If p is prime, then for a such that $a \neq 0 \bmod p$, $a^{p-1}=1 \bmod p$.
(Extended) Euclid's Algorithm	Given integers a and b, compute their greatest common divisor d and two integers x and y such that $ax+by=d$.	$O(\log a)$ modular arithmetic operations	Full algorithm in the lecture notes. Note that modular arithmetic takes $O(\log^2(a))$ bit operations.
Randomized 2SAT	Choose a random assignment. While there is an unsatisfied clause, random flip one of its variables. Repeat until the formula is satisfied or you're tired.	$O(n^2)$ expected iterations to find a solution	The complexity analysis for this algorithm uses the notion of a random walk, where we move either 1 step closer or farther from the solution in each step. To find the expected value of a random walk, use a recurrence.
Local Search	Represent the solution space as nodes connected by edges (which represent possible "moves" in the search). Define a cost function that assigns a cost to each possible solution and use it to move from some starting point towards a better solution.	Varies	Some things to consider: what is your starting point? what is your solution space? what is your cost function? what are the "neighborhoods" or edges connecting solutions? how do we move between neighbors?
Greedy Vertex Cover	Repeatedly choose the vertex with the highest degree and put it in the cover.	$O(\|V\|)$	The algorithm could be off by a factor of $\Omega(\log n)$
Simple vertex cover	Repeatedly choose an edge, and throw both of its endpoints into the cover. Throw the vertices and their adjacent edges out of the graph, and continue.	$O(\|V\|)$	The algorithm uses twice as many vertices as the optimal vertex cover.
Random Max Cut Approx	Flip a coin to decide which set to put each vertex in.	$O(\|V\|)$	This algorithm yields a solution that is within a factor of 2 of the optimal solution.

Deterministic Max Cut Approx	Start with all the vertices in one set. While moving a vertex across the cut will improve the cut, do so.	$O(\|E\|)$	This algorithm yields a solution that is within a factor of 2 of the optimal solution.
TSP Approximation	Find an MST, create a *pseudo-tour* by walking around the MST, *short cut* repeated vertices in the tour	Depends on which MST algorithm you use	This tour is within a factor of 2 of the optimal. This algorithm works for any TSP that satisfies the triangle inequality for distances
Random Max SAT	Flip a coin to decide if each variable is true or false.	$O(v)$ where v is the number of variables	On average, $1 - 2^{-k}$ clauses will be satisfied, where each clause has k literals.
Randomized Rounding Max Sat	Use linear programming to find a decimal solution. If a variable has the value t, round it up to 1 with probability t.	Depends on which linear programming algorithm you use	This solution will be within a factor of $(1 - 1/e)$ of the optimal. If we combine the random Max SAT and randomized rounding Max SAT algorithms, we can satisfy 3/4 of the clauses on average.
Ford-Fulkerson	Push n units of flow over an augmenting path from source to sink using DFS. Update residual network capacities. Repeat until no units of flow can be pushed over the graph.	$O(Ef^*)$ with DFS, $O(VE^2)$ with BFS.	Max flow = min cut. Try to think about what the source and sink should be when constructing a flow problem.
Linear Programming: Definition	An optimization problem for an objective function, subject to constraints on the variables. The objective function and constraints are all linear in the variables.	The simplex algorithm runs in polynomial time	It is possible that the optimal solution is infinity or that no feasible solution exists.
Linear Programming: Reductions	To turn an inequality $\sum a_i x_i \leq b$ into an equality constraint, rewrite as $\sum a_i x_i + s = b, s \geq 0$.	Depends on the reduction, but usually polynomial time	s is called a slack variable.
Two Player Games	Given a payoff matrix, can convert each player's strategy into an LP optimization, where the strategy probabilities must add to 1.	Optimal play is mixed strategy, which can be solved by treating each player's strategy as a linear program.	

NP Hard Problems

- **circuit SAT**. Given a Boolean circuit and the values of some of its inputs, determine if there is an assignment to the rest of its inputs such that the circuit will output T.
  - Cook's theorem proves that circuit SAT is **NP** complete.
- **3SAT** Given a 3-CNF formula, determine if there is some satisfying assignment. A 3-CNF formula, for our purposes, is the conjunction of disjunctions, where each disjunction contains at most 3 terms. In other words, "and's of or's." For example, $(x_1 \vee x_2) \wedge (\overline{x_3} \vee \overline{x_2} \vee \overline{x_1})$ is a possible input formula.
- **Integer Linear Programming** Find an integer assignment to the variables that maximizes a given objective function while respecting given constraints. For example, maximize the function $x + y$ under the constraints $x < 2$ and $y < 2$, where x and y must be integers.
- **Independent Set** An independent set is a set of vertices such that no two vertices in the set are connected by an edge. Given a graph and a number K, is there an independent set of size at least K?
- **Vertex Cover** If $G = (V, E)$ is a graph, a vertex cover of G is a set of edges $C \subseteq V$ such that every edge in E has at least one endpoint in C. Given a graph G and a number K, determine if G has a vertex cover of size at most K. Note that if C is a vertex cover, $V - C$ is an independent set.
- **Clique** A clique is a set of vertices that is fully connected, so every possible edge between vertices in the set is in the graph. Given a graph G and a number K, is there a clique of size at least K? C is a clique in G if and only if it is an independent set in G^c.
- **Max cut** Given a graph G, divide the vertices into two sets such thatthe number of edges crossing the cut is maximized.
- **Euclidean Travelling Salesperson** Given n cities in the $x - y$ plane, find a tour (cycle) of minimum length that travels through all the cities.
- **Max Sat** Given a formula in conjunctive normal form (ands of ors), determine the maximum number of clauses that can be satisfied. An example of a formula in CNF is: $(x \vee y \vee z \vee w) \wedge (\overline{x} \vee y)$