	Implementation	Runtime	Properties	Applications
Breadth-first Search	Frontier(Queue), Visited[], ParentEdges[]	AL: ***O(V+E)*** AM: ***O(V²)***	ParentEdges[] does not form a cycle	Unweighted SSSP #connected components Detect cycles Bipartite check
Depth-first Search	Frontier(Stack), Visited[], ParentEdges[]	AL: ***O(V+E)*** AM: ***O(V²)***		Unweighted SSSP #connected components #strongly connected components (u, v) same component Union Find labelling Topo-sort Detect cycles
Topo-sort	<u>Post-Order DFS:</u> Visited[], Topo[] <u>Kahn's Algorithm:</u> Topo[], Count[], Queue<>()	***O(V+E)***	**Strongly Connected Components**: ∀u, v ∈ SCC, u → v and v → u Graph of SCC is acyclic DAG with V nodes has V SCCs	DAG_SSSP Task Scheduling LIS
Union Find (Weighted + Path Compression)	Node::size, parent, <u>Union</u>: findRoot(u).parent = findRoot(v), for u.root.size < v.root.size <u>Find</u>: findRoot(u) == findRoot(v) <u>findRoot</u>: set node.parent = root from node to root path	***O(α(m, n))***	Depth of Tree: O(logn)	Kruskal's Algorithm

A graph is planar iff it contains $K_{3,3}$ or K_5
Euler's Formula: V – E + F = 2
Properties:
- A cycle with even number of nodes is bipartite -#edges == #vertices in a cycle
- A connected graph with n vertices and n – 1 edges is a tree

	Implementation	Data Structures/methods	MaxST	Runtime	Properties
Prim's (undirected)	Choose an arbitrary start node. Repeat until MST: pick the minimum weight edge across inMST[] & G	inMST[], PriorityQueue decreaseKey(), extractMin()	Pick the maximum weight edge from PQ	***O(Elog(V))***	
Kruskal (undirected)	Sort the edges in ascending order. For every edge e, add e into the MST if it does not form a cycle.	UnionFind[], Edges[], inMST[] union(), find()	Sort the edges in descending order	***O(Elog(V))***	
DAG_MST	Topo-sort, then relax in topo-order			***O(V+E)***	
BFS/DFS	**If unweighted/same weights**, just use BFS/DFS		MaxST == MinST	***O(V+E)***	Cost of MST = (V – 1)k, k = weight
▪ MST does not contain a cycle	▪ Removing an edge from an MST produced 2 smaller MSTs; converse not necessarily true	▪ Max-weight edge in any cycle will not be in the MST; inverse might not be true	▪ If an edge e is not in the MST, then it is the heaviest edge in some cycle in G; inverse may not be true	▪ Min-weight edge across a cut will be in the MST. Min-weight in G will always be in the MST; inverse of both may not be true	▪ Min-weight edge adjacent to any node will be in the MST; inverse may not be true

	Implementation	Runtime	Conditions	Longest Path	Properties
Bellman-Ford	Initial estimate: INF; Relax all edges, until no changes (max V)	AL: ***O(VE)*** AM: ***O(V³)***	Negative Weight Cycle → works but cannot find shortest path	Negate edges/ Initial: -INF + ΔRelax	Triangle Inequality After j iterations of BMF, the j hop estimate on the shortest path tree is correct.
Dijkstra's	Initial estimate: INF; Dequeue from PQ, relax all outgoing edges, enqueue neighbours if relaxed	AL: ***O(Elog(V))*** AM: ***O(V²log(V))*** (AVL Tree PQ) ***O(V²)*** (Array)	Only non-negative weights	If all non-positive: negate edges	Each time a node is dequeued from PQ, the node has the correct estimate Each E visited exactly once If *v* is extracted after *u*, v.priority ≥ u.priority
DAG_SSSP	Topo-sort, then relax in topo-order	***O(V+E)***	DAG	Negate edges/ ΔRelax	LIS Prize Collection
BFS	Relax while traversing in BFS order	***O(V+E)***	Unweighted Graph	Not possible, unless tree (then longest == shortest path)	
Floyd-Warshall	Initial estimate: INF; k * V table DP[k][v][w] = MIN { DP[k-1][v][w], DP[k-1][v][s] + DP[k-1][s][w] }	***O(V³)***	-	Not possible	

master theorem

$$T(n) = aT(\tfrac{n}{b}) + f(n) \quad a \geq 0, b > 1$$

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

orders of growth

$$1 < \log n < \sqrt{n} < n < n \log n < n^2 < 2^n < 2^{2n}$$

$$\log_a n < n^a < a^n < n! < n^n$$

$$a^{\log_a b} = b \quad \left(1 - \frac{1}{n}\right)^n \to \frac{1}{e} \quad S_{AP,n} = \frac{n}{2}(2a + (n-1)d)$$

$$S_{GP,n} = \frac{a(1-r^n)}{1-r} \quad \sum_{i=1}^{\infty} \frac{i}{2^i} = \sum_{i=1}^{\infty} \left(\sum_{j=i}^{\infty} \frac{j}{2^j}\right) = 2$$

Graph Augmentation

SSSP property: The set of shortest paths from s to any node u in G forms the shortest path tree

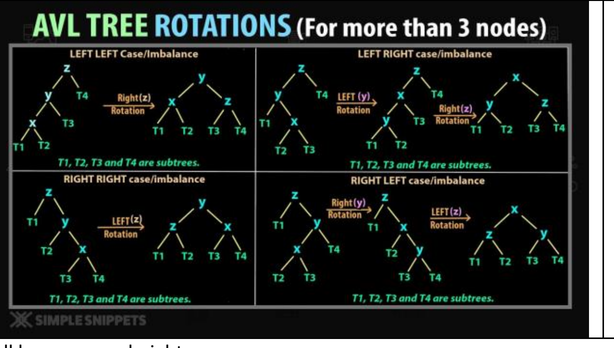
1. Need to alternate between edges: G(u, v'): +ve, G'(u', v): -ve. Bipartite graph
2. Need to run SSSP *n* times on *n* nodes: Connect to dummy node and run a single SSSP from dummy node
3. Mandatory Nodes:
   (i) Connect dummy to mandatory nodes. Run SSSP from Source to dummy. Then, run SSSP from Dest to Dummy. Take minimum sum.
4. k weighted special edges where you can choose to/not to take: Create k copies of the graph. Take min. of shortest paths to all k end nodes. If unweighted, set special edges to 1 and the rest to 0. Just run SSSP on original graph
5. Weight of edge depends on current node: duplicate graph, then connect Node u to Node u' with edge weight == 0; "Bridge"
6. Graph cloning:
   - k mandatory nodes in any order: 2^k graphs (eg. pizza sushi qn); visit each graph in partial order
   - k mandatory nodes in a fixed order: k graphs
   - at least 1 of k mandatory nodes/edges: 2 graphs, connected via k mandatory nodes
7. Shortest distance (w neg weights) at most k hops away: k * BMF, using estimates from only the previous iteration.

	Upper Random	Upper Sorted	Upper Reversed	Upper Almost	Lower	Space	Stable	Invariants
Bubble	***O(n²)***	***O(n)***	***O(n²)***	***O(n²)*** ***O(n),*** *depends*	***O(n)***	***O(1)***	***yes***	After *k* iterations, the last *k* items are in final sorted order. Every iteration only puts items closer to their final positions
Selection	***O(n²)***	***O(n²)***	***O(n²)***	***O(n²)***	***O(n²)***	***O(1)***	***no***	After *k* iterations, the first *k* items are in final sorted order. The next smallest item will swap with first item in current unsorted subarray.
Insertion	***O(n²)***	***O(n)***	***O(n²)***	***O(n)***	***O(n)***	***O(1)***	***yes***	After *k* iterations, the first *k* items are in sorted order. After the *k-th* iteration, the items in [k : end] will be in their original, unsorted positions (in order words, only the first *k* elements are being shifted around).
Merge	***O(nlogn)***	***O(nlogn)***	***O(nlogn)***	***O(nlogn)***	***O(nlogn)***	***O(nlogn)***	***yes***	During merge operation. the left and right halves are already sorted. Items at one end (or half) won't "jump" to the other end (or half) until the final few merge steps
Fixed Quicksort	***O(nlogn)***	***O(n²)***	***O(n²)***	***O(n²)***	***O(n)***	***O(1)***	***no***	After *k* iterations, at least *k* items are in final sorted order (the pivot).
Random Quicksort	***O(nlogn)***	***O(nlogn)***	***O(nlogn)*** ***O(nlogk)***	***O(nlogn)***	***O(n)***			
Heap	***O(nlogn)***	***O(nlogn)***	***O(nlogn)***	***O(nlogn)***	***O(n)*** (all same)	***O(1)***	***no***	Heapify in O(n). Call extractMin() (replace root with last node, then bubble down the new root) n times

Chaining	Open Addressing
Search: **O(α)**; Insert: **O(1)**	Search & Insert: **O(1/1-α)**
Space: **O(m + n)**	Space: **O(m)**
SUHA: P(X(i, j)) = 1/m ---Linearity of Ex. ---> n/m	UHA: 1 + n/m(1 + n-1/m-1(1 + ...)) $\leq$ 1 + a + a² +... = 1/1-a
Double Hashing: h(k, i) = f(k) + **ig(k)** mod **m**, g(k) & m are co-prime to cover all buckets Amortized cost: Sum(k operations) $\leq$ kT(n)	

	Upper	Invariants	Applications
Binary Search	***O(logn)***	start $\leq$ target $\leq$ end	Monotonically increasing/decreasing functions
Peek Finding	***O(logn)***	start $\leq$ target $\leq$ end & target is a peak in [0: length)	
Quick Select	***O(n²)*** (W) ***O(n)*** (Ex)	After the *i*-th iteration, the *i* items are in correct position	Find *k*-th largest/smallest elements in O(n) (expected) time

AVL Tree



Invariants

Max #nodes = $2^{h+1} - 1$
Min #nodes = $2^{h/2}$
Max height = 1.44logn
Max lvl diff = logn or h/2

B Tree	All leaves same height #children = #keys + 1, #keys = [B, 2B-1] Min height = $log_{2B}\, n$; Max height = $log_B\, n$ Siblings(u, v) = | deg(u) – deg(v) | $\leq$ B, height(u) – height(v) == 0 Min #nodes = $\frac{a^{h+1}-1}{a-1} \geq a^h$
Heap	Parent: (i – 1)/2; Left: 2i + 1; Right: 2i + 2 Min #interval nodes: 2^{logn} - 1 #comparisons: C[n] = #children + MAX{ C[c1], C[c2] } 2nd largest node will be of the children of thr root
Tries	Search: O(L); time does not depend on number of strings

	Adjacency List	Adjacency Matrix
Space	O(V + E) Cycle: O(V) Clique: O(V²)	O(V²) Cycle: O(V²) Clique: O(V²)
Find any neighbour	Fast (O(1))	Slow (O(V))
Enumerate all neighbours (searching)	Fast (O(1))	Slow (O(V))
Are u and v neighbours?	Slow (O(min(V, E))	Fast (O(1))
	Better if Sparse	Better if Dense
		Matrix multiplication takes O(V³) time