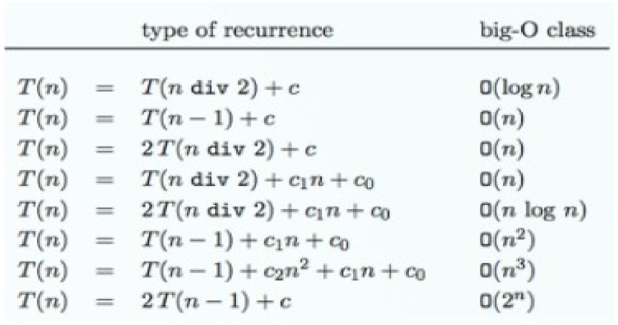
Complexity
Big-O – upper bound, Big-Omega – lower bound, Big-Theta – exact bound

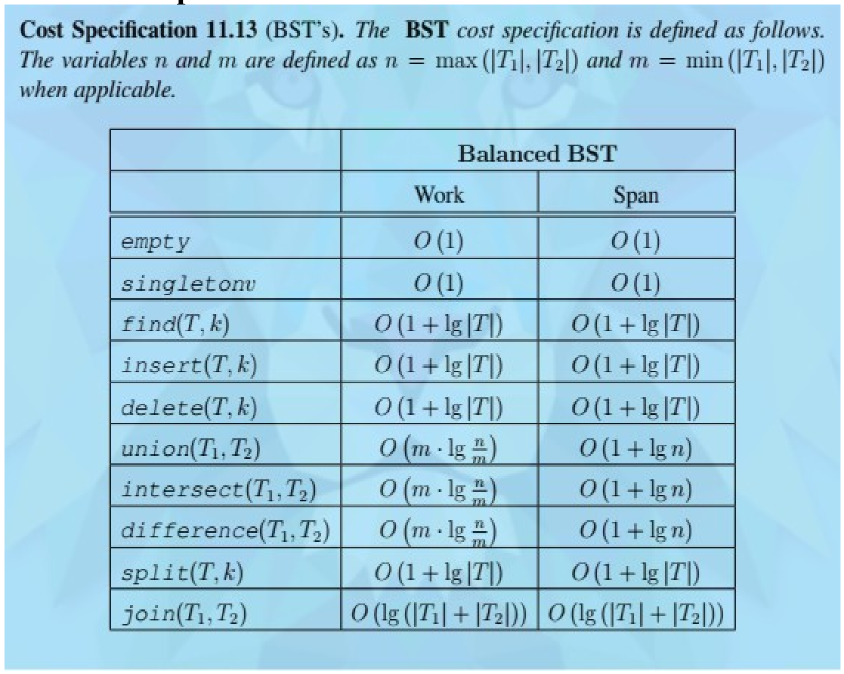
Sum [0,n]	Result
i	n(n-1)/2
a^i	(a^(n+1)-1)/(a-1)
i^2	(1/6)n(n+1)(2n+1)
ia^i	a(na^(n+1)-(n+1)(a^n)+1)/(a-1)^2
1/i	O(logn)



type of recurrence			big-O class
$T(n)$	$=$	$T(n \text{ div } 2) + c$	$O(\log n)$
$T(n)$	$=$	$T(n-1) + c$	$O(n)$
$T(n)$	$=$	$2T(n \text{ div } 2) + c$	$O(n)$
$T(n)$	$=$	$T(n \text{ div } 2) + c_1 n + c_0$	$O(n)$
$T(n)$	$=$	$2T(n \text{ div } 2) + c_1 n + c_0$	$O(n \log n)$
$T(n)$	$=$	$T(n-1) + c_1 n + c_0$	$O(n^2)$
$T(n)$	$=$	$T(n-1) + c_2 n^2 + c_1 n + c_0$	$O(n^3)$
$T(n)$	$=$	$2T(n-1) + c$	$O(2^n)$

Probability/Randomization
Markov's Inequality: Pr[X >= aE[X]] <= 1/a OR Pr[X >= a] <= E[X]/a
Union Bound: Pr[UA_i] <= sum of A_i
Kth Smallest: Quicksort, but only going down one branch. W is O(n), S is O((log^2)(n))
Quicksort: Elements compared iff one of them is a pivot and they aren't split. The probability that element i and element j are compared is 2/(j-i+1). Summed, this predicts an overall work of O(nlogn)

BSTs & Treaps



Cost Specification 11.13 (BST's). *The* **BST** *cost specification is defined as follows. The variables* n *and* m *are defined as* $n = \max(|T_1|, |T_2|)$ *and* $m = \min(|T_1|, |T_2|)$ *when applicable.*

	Balanced BST	
	Work	Span
empty	$O(1)$	$O(1)$
singletonv	$O(1)$	$O(1)$
find(T, k)	$O(1 + \lg \lvert T \rvert)$	$O(1 + \lg \lvert T \rvert)$
insert(T, k)	$O(1 + \lg \lvert T \rvert)$	$O(1 + \lg \lvert T \rvert)$
delete(T, k)	$O(1 + \lg \lvert T \rvert)$	$O(1 + \lg \lvert T \rvert)$
union(T_1, T_2)	$O\left(m \cdot \lg \frac{n}{m}\right)$	$O(1 + \lg n)$
intersect(T_1, T_2)	$O\left(m \cdot \lg \frac{n}{m}\right)$	$O(1 + \lg n)$
difference(T_1, T_2)	$O\left(m \cdot \lg \frac{n}{m}\right)$	$O(1 + \lg n)$
split(T, k)	$O(1 + \lg \lvert T \rvert)$	$O(1 + \lg \lvert T \rvert)$
join(T_1, T_2)	$O(\lg(\lvert T_1 \rvert + \lvert T_2 \rvert))$	$O(\lg(\lvert T_1 \rvert + \lvert T_2 \rvert))$

Treap: Satisfies BST property on keys, max-heap property on priorities
Treap Height: For a given S, the treap for S can be generated by Quicksort, indicating that the height of S is the same as the recursion depth for Quicksort which is O(logn) with high probability

Sets & Tables
Sets represented as BST

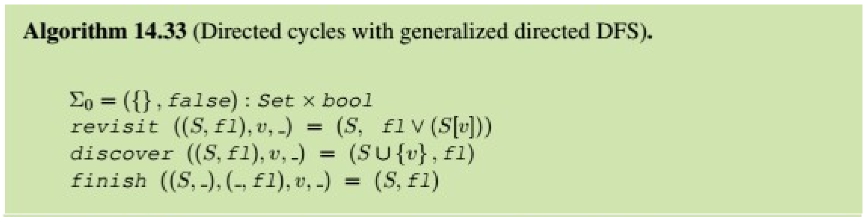
Graphs

Costs	Edge Set	Adj Table	Adj Seq
(u,v) in G	O(log|E|)	O(log|V|)	O(dG(u)))
Map over edges	O(|E|)	O(|E|)	O(|E|)
Find neighbors	O(|E|)	O(log|V|)	O(1)
Map over Ng(v)	O(dG(v))	O(dG(v))	O(dG(v))
dG(v)	O(|E|)	O(log|V|)	O(1)

DFS Numberings and DFS trees:

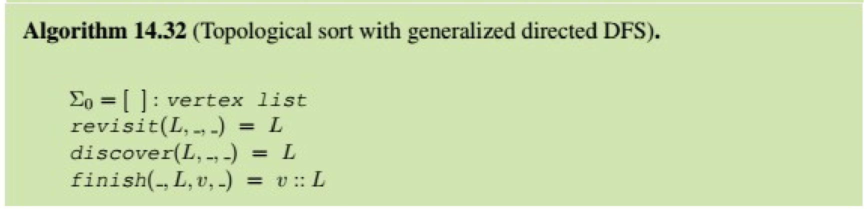
- *Tree edge*: (u,v) if v is discovered by u during DFS
    - d(u) < d(v) < f(v) < f(u)
- *Back edge*: (u,v) if u is a descendant of v
    - d(v) < d(u) < f(u) < f(v)
- *Forward edge*: (u,v) if u is an ancestor of v
    - d(u) < d(v) < f(v) < f(u)
- *Cross edge*: (u,v) if an edge is not one of the other edges
    - d(u) < f(u) < d(v) < f(v)

Theorem: On a directed acyclic graph (DAG) when finishing a vertex v in DFS, all vertices reachable from v have already exited



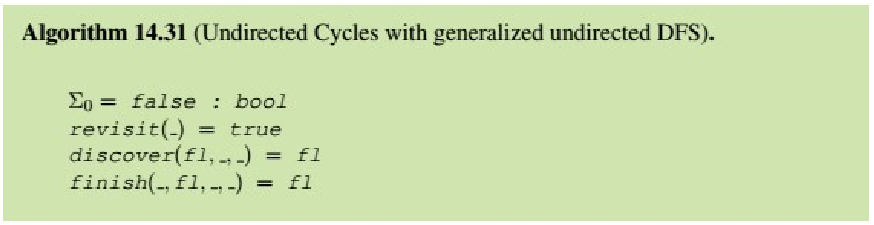
Algorithm 14.33 (Directed cycles with generalized directed DFS).

```
Σ₀ = ({}, false) : Set × bool
revisit  ((S, fl), v, _)  =  (S,  fl ∨ (S[v]))
discover ((S, fl), v, _)  =  (S ∪ {v}, fl)
finish   ((S, _), (_, fl), v, _)  =  (S, fl)
```



Algorithm 14.32 (Topological sort with generalized directed DFS).

```
Σ₀ = [ ] : vertex list
revisit(L, _, _)  =  L
discover(L, _, _)  =  L
finish(_, L, v, _)  =  v :: L
```



Algorithm 14.31 (Undirected Cycles with generalized undirected DFS).

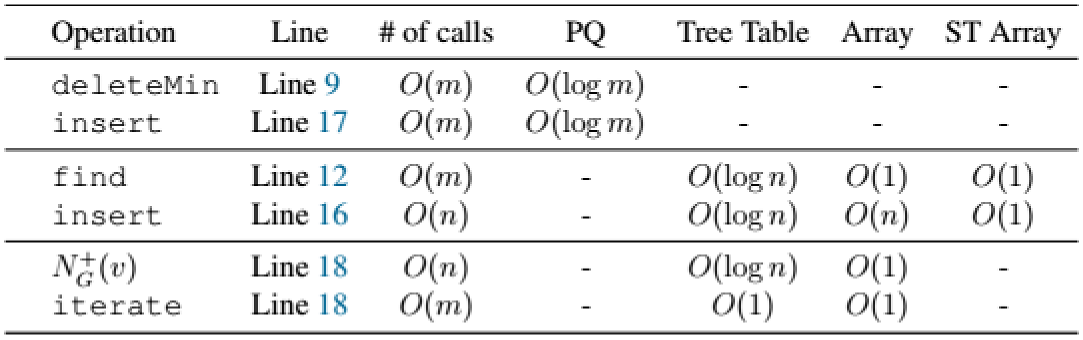
```
Σ₀ =  false  :  bool
revisit(_)  =  true
discover(fl, _, _)  =  fl
finish(_, fl, _, _)  =  fl
```

Shortest Paths

Djikstra's Algorithm: For a weighted graph G=(V,E,w) and a source s, Dijkstra's algorithm is priority-first search on G starting at s with d(s)=0, using priority p(v) = min(d(x) + w(v)) and setting d(v)=p(v) when v is visited.

- Sequential, O(mlogn)
- Djikstra's Property: The overall shortest-path weight from s via a vertex in X directly to a neighbor in Y (the frontier) is as short as any path from s to any vertex in Y

- Can fail on graphs with negative weights (cyclic w/ negative weights → infinite length path)



Operation	Line	# of calls	PQ	Tree Table	Array	ST Array
`deleteMin`	Line 9	$O(m)$	$O(\log m)$	-	-	-
`insert`	Line 17	$O(m)$	$O(\log m)$	-	-	-
`find`	Line 12	$O(m)$	-	$O(\log n)$	$O(1)$	$O(1)$
`insert`	Line 16	$O(n)$	-	$O(\log n)$	$O(n)$	$O(1)$
$N_G^+(v)$	Line 18	$O(n)$	-	$O(\log n)$	$O(1)$	-
`iterate`	Line 18	$O(m)$	-	$O(1)$	$O(1)$	-

Figure 15.1: The costs for the important steps in the algorithm `dijkstraPQ`.

Bellman-Ford
- Constructs path by adding edges when beneficial. Stops when more than |V| edges are used.
- At each step, distances updated by D' = {v → min(D[v],min(D[v] + w(u,v)) for neighbors_ : v in V}
- Cost
    -Table: W = O(nmlogn), S = O(nlogn)
- Array sequence: W = O(nm), S = O(nlogn)

Graph Contraction
Star contraction: For example, to determine the centers, we can flip a coin for each vertex. If a vertex flips heads, then it becomes the center of a star. If a vertex flips tails, then it tries to become a satellite by finding a neighbor that is a center. If no such neighbor exists (all neighbors have flipped tails or the vertex is isolated), then the vertex becomes a center. If a vertex has multiple centers as neighbors, can pick one arbitrarily.
- Removes n/4 vertices in expectation

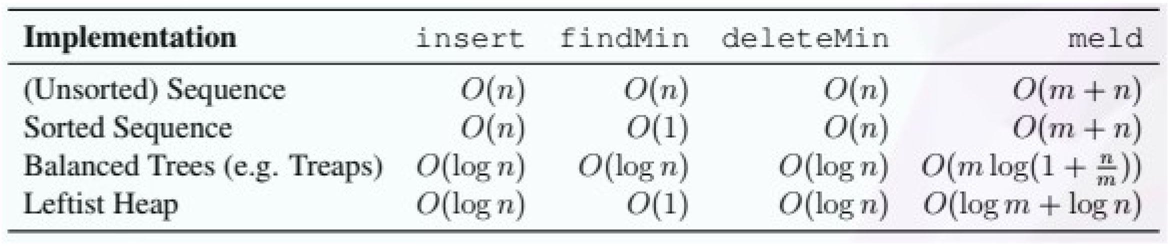
Minimum Spanning Trees
Light-edge property: Let G=(V,E,w) be a connected, undirected,weighted graph with distinct edge weights. For any cut of G, the minimum weight edge that crosses the cut is in the minimum spanning tree of G
Kruskal's Algorithm: At each step, choose the minimum weight edge which does not form a cycle
- Cost: W = S = O(mlogn)
Prim's Algorithm: Priority-first search on G starting at an arbitrary vertex s using priority p(v) = min w(x,v) for x in X and setting T = T U P(u,v)} when visiting v where w(u,v) = p(v).
- Cost: W = S = O(mlogn)
Boruvka's Algorithm: At each step, select the minimum edge incident on each vertex and contract these edges. Remove self edges and keep the minimum-weight redundant edges. Add selected edges to MST. Rinse and repeat.
- Cost:
    - Tree contraction: W = O(mlogn), S = O((log^3)(n))
    - Star contraction: W = O(mlogn), S = O((log^2)(n))

Dynamic Programming
Top-down approach (memoization): The top-down approach is based on generating implicitly the recursion structure from the root of the DAG down to the leaves. Each time a solution to a smaller

instance is found for the first time it generates a mapping from the input argument to its solution.
Bottom-up approach: One way to implement bottom-up dynamic programming is to do some form of systematic traversal of a DAG. We can start from the bottom of the graph and work our way back up towards subproblems that depend on the current subproblem.

Hashing
Load Factor: n/m where n is total number of keys, m is number of distinct hash values
Collision Resolution: Separate chaining, open addressing (linear probing, quadratic probing), perfect hashing, multiple choice hashing/cuckoo hashing
Parallel hashing (with open addressing): To insert keys into a hash table in parallel, we perform multiple rounds of writings into the table in parallel. Any key that cannot be written because of a collection continues into next round until all keys have been written.
- Work: O(|K|) where K is the set of keys to be inserted
- Span: O(log|K|)

Priority Queues
Leftist property: For all node x in a leftist heap, rank(L(x)) >= rank(R(x))



Implementation	insert	findMin	deleteMin	meld
(Unsorted) Sequence	$O(n)$	$O(n)$	$O(n)$	$O(m+n)$
Sorted Sequence	$O(n)$	$O(1)$	$O(n)$	$O(m+n)$
Balanced Trees (e.g. Treaps)	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(m\log(1+\frac{n}{m}))$
Leftist Heap	$O(\log n)$	$O(1)$	$O(\log n)$	$O(\log m + \log n)$

PASL
- Compare and swap: When executed with a 'target' atomic object and an 'expected' cell and a new value 'new', the following is done atomically:
    1. Read contents of 'target'
    2. If the contents equals the contents of 'expected', then write 'new' into the 'target' and return T
    3. Otherwise, return F
- ABA Problem: When multiple threads update a value such that it goes from a value A to B and then back to A, the compare-and-swap won't detect this change and will be oblivious to any side effects.
- Thread: A maximal computation consisting of a sequence of instructions that do not contain calls to fork(2) except perhaps as its last action
- Scheduling: Assigning each thread a processor such that:
    1. Each thread is assigned to a unique processor for as many consecutive steps as its weight
    2. No thread is executed before its descendants in the DAG
    3. No processor is assigned more than at most one thread at a time
- Greedy Scheduling Principle: If a computation is run on P processors using a perfect greedy scheduler that incurs no costs in creating, locating, and moving threads, then the total time (clock cycles) for running the computation Tp is bounded by Tp < (W/P) + S. Where W is the work of the computation and S is the span of the computation (in clock cycles)
- P-processor speedup: The speedup on P processors is the ratio Tb/Tp where the term Tb represents the run time of the sequential baseline program and the term Tp is the time measured for the P-processor run
- Asymptotically work efficient: If the work of the algorithm is the same as the work of the best known serial algorithm

- Observed work efficiency: A parallel algorithm that runs in time T1 on a single processor has *observed work efficient factor of r* if r = T1/Tseq where Tseq is the time taken by the fastest known sequential algorithm
- Good parallel algorithm:
    - Asymptotically work efficient
    - Observably work efficient (r < 1.5)
    - It has low span
- Granularity control/coarsening: Switching to a sequential algorithm when the problem size falls below a certain threshold to avoid excessive parallel overhead
- Parallel: An algorithm or application that performs multiple computations at the same time for the purposes of improving the completion or run time
- Concurrent: A computation that involves independent agents which can be implemented with processes or threads, that communicate and coordinate to accomplish the intended result