Final Exam Review:

Single Source Shortest Path:

I: - Weighted Graph $G = (V, E)$ where $len(e_i) > 0$ for all $e \in E$
  ↳ G = directed or undirected
  - Source nodes $s \in V$

O: - array dist where $dist[u]$ is the shortest distance between s and u, for some reachable node $u \in V$.
  - array prev where $prev[u]$ is the previous node along the shortest path between s and u.

Dijkstra's = pick the next unvisited node that's closest to s based on dist and process its neighbors

I: - Graph $G = (V, E)$, directed or undirected;
  - positive edge lengths $\{l_e : e \in E\}$; vertex $s \in V$

O: - for all vertices u reachable from s, $dist(u)$ is set to the distance from s to u.

pseudocode

```
for all u ∈ V:    dist(u) = ∞ , prev(u) = nil , dist(s) = 0

H = makequeue (using dist-values as keys)
while H is not empty:    u = deletemin(H)
  for all edges (u,v) ∈ E; if dist(v) > dist(u) + l(u,v):
      dist(v) = dist(u) + l(u,v)  ; prev(v) = u ; decreasekey(H, v)
```

Minimum Spanning Tree

I: weighted, undirected, connected Graph $G = (V, E)$

O: A Tree $T = (V, E')$, with $E' \subseteq E$ that minimizes edge weight sum

Prim's Algorithm

- each nodes w/ neighbors w/ some cost; need to visit each node/at least one edge to neighbor to create valid MST
- every time new node has been visited → pick one w/ least cost edge
- Time Complexity: $O(|E| \log |V|)$; (binary heap & adjacency matrix)

Kruskal's Algorithm

- picks cheapest edge, then pick another edge, and another...
- for each pick, do Not create a cycle (avoid cycles!)
- Time Complexity: $O(E \log E)$ or $O(E \log V)$

(0) = Matrix = | 0 | 0 | 0 |

List = [ ]

Name	Best	Average	Worst	Memory	Stable
Quick Sort	$n \log(n)$	$n \log(n)$	n^2	$\log(n)$	No
Merge Sort	$n \log(n)$	$n \log(n)$	$n \log(n)$	n	Yes
Selection Sort	n^2	n^2	n^2	1	No
Insertion Sort	n	n^2	n^2	1	Yes
Heap Sort	$n \log(n)$	$n \log(n)$	$n \log(n)$	1	No

Min Heap
tree element $\geq$ parent

Max Heap
tree element $\leq$ parent

[source = no arrows @ node
sink = no arrows out of node]

Recursive Relation $[T(n) = a\,T(n/b) + f(n)]$

$a > 0$ — # of recursive calls made @ each step?

$b > 1$ — reducing problem by what multi. factor?

$f(n)$ — # of work needed to be done @ recurse calls?

Pre-Order → print node; print left; print right →

In-Order → print left; print node; print right →

Post-Order → print left; print right; print node →

Master Theorem

given RR
$T(n) = aT(n/b) + O(n^k)$;
one of the following is true

1) $a < b^k \rightarrow T(n) \in \Theta(n^k)$
2) $a = b^k \rightarrow T(n) \in \Theta(n^k \log(n))$
3) $a > b^k \rightarrow T(n) \in \Theta(n^{\log_b a})$

top → bottom / left → right

left → node → right

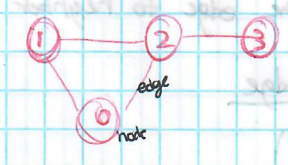
bottom → top / left → right

BST Add, Remove, Searching: Best = $O(1)$; Worst = $O(n)$; Average = Random Tree, usually $O(n)$

height: # of edges along longest path; root to leaf

root = top node

parent → child



matrix:

	0	1	2	3
0	0	1	1	0
1	1	0	1	0
2	1	1	0	1
3	0	0	1	0

List:
0: [1,2]
1: [0,2]
2: [0,1,3]
3: [2]

→ [[1,2], [0,2], [0,1,3], [2]]