1 Sort

1.1 Insertion Sort

Insertion sort works by scanning from right to left and when an item is out of place, "bubbling" that item to the left until it is in its rightful place.

- stable? Yes.
- time complexity: Worst case is θn^2.
- space complexity: $\theta 1$ as everything is done in place.
- best-case input: A nearly-sorted array, as almost all needed work is done.
- worst-case input: An array in reverse order. $\sum_{k=1}^{n} = 1 + 2 + 3 + ... + n$

1.2 Selection Sort

Selection sort scans through the entire *unsorted* part of the list looking for the next smallest item. Therefore the running-time of this algorithm is given by the triangular series $\sum_{k=1}^{n} k$ where k is the number of compares.

1.3 HeapSort

When done in the classic "in place" method, HeapSort breaks items into single item queues and then merges those already *sorted* queues into larger sorted queues

1.4 MergeSort

1.5 QuickSort

Picks

2 Search

Adjacency list: list of nodes that can be *immediately* reached from the current node. A.K.A. connections or outgoing edges.

Adjacency matrix: table or matrix of $nodes \cdot nodes$ where each intersection is either *true* or *false*

	a	b	c
a	0	1	1
b	1	0	0
c	1	0	0

2.1 Djikstra's

time complexity: $O(|E|log|V|)$ according to Hug, and $O(|E|+|V|log|V|)$ according to Wikipedia. Hug's justification is as follow::

- we do V insertions, which each cost $O(log|V|)$ time.
- we do V min-deletions, costing $O(log|V|)$ time each.
- we do E "priority decreases", costing $O(log|V|)$ time each.
- so our total runtime is $O(V \cdot log|V| + V \cdot log|V| + E \cdot log|V|) \Rightarrow O(2V + E \cdot log|V|)$
- which becomes $O(E \cdot log|V|)$

Algorithm for finding shortest path in a graph. *Greedy* algorithm in that it enqueues the lowest cost path th

2.2 A*

Djisktra's but with heuristics added to edge-weights.

2.3 Kruskal's

time complexity: V is at most E^2, so it follows that $O(logV^2) \Rightarrow O(2logV) \Rightarrow O(logV)$.

Algorithm for finding the MST of a graph.

Pseudocode:

```
insert all edges into MinPQ
while (not all nodes are connected):
    dequeue edge
    if (edge doesn't create a cycle):
         add edge
```

One issue with this algorithm is that it runs the risk of adding edges that are disconnected to the larger whole of the graph if not careful about enqueuing edges. So for Kruskal's to be the ideal algorithm we must be sure that the graph is connected.

2.4 Primm's

time complexity:

- with binary heap and adjacency list: $O(|E|\ log\ |V|)$
- with adjacency matrix $O(|V|^2)$

Another algorithm for finding the MST of a graph. Algorithm of choice if not sure that graph is connected. As Primm's is run from an arbitrary vertex, it will only find vertices that are connected to starting vertex.

Pseudocode:

```
start at a vertex
while (MST isn't found):
    visit closest unvisited vertex to any visited vertex
    add edge to MST
```

So what happens is essentially that we pick a place to start, then visit the node closest to the start if we haven't already visited it. We will then continue to do this until we've found the MST for the graph.

3 Addenda

3.1 Asymptotics

$$Taking : \frac{r(\alpha)}{r(\beta)} = \left(\frac{\alpha}{\beta}\right)^b as\ \gamma = \delta \tag{1}$$

$$we\ can\ siplify\ to :\ log_\gamma \delta = b \text{ and solve for } b \tag{2}$$