CSCI 104L Lecture 10: Graph Search

Dijkstra's Algorithm

```
int d[n]; //distances from the start node u
int p[n]; //predecessors
int c[n][n]; //edge costs
void Dijkstra (int u) {
  PriorityQueue<int> pq();  //How should we implement this?
  d[u] = 0;
  pq.add(u, d[u]);
  while(!pq.isEmpty()) {
    int v = pq.peek();
    pq.remove();
    for all nodes outgoing edges (v,w) from v {
      if (w hasn't been visited || d[v] + c[v][w] < d[w]) {
        d[w] = d[v] + c[v][w];
        p[w] = v;
        if (this is w's first visit) {
          pq.add(w, d[w]);
        }
        else pq.update(w, d[w]);
      }
    }
  }
}
```

Question 1. How many add/remove/update calls are needed?

Question 2. What is the runtime of Dijkstra's using an unsorted array as an implementation?

Question 3. What is the runtime of Dijkstra's using a sorted array?

Question 4. What is the runtime of Dijkstra's using a MinHeap?

Note that in Dijkstra's Algorithm, we only ever bubbleUp.

Question 5. Is there any type of graph where you'd want to use an unsorted array instead of a heap?

A* Search

A* search is a heuristic search. That means that it uses "rules of thumb" to often improve the runtime. It never runs worse than Dijkstra, and always finds the correct solution, but it requires more information (which you may not have readily available).

- A* modifies Dijkstra's so that we always next explore the node with smallest d[v]+h[v], where h[v] is our estimate of how far v is from the destination.
- This only works if our heuristic never over-estimates.
- Our heuristic, when it is wrong, must underestimate. We want it to be as accurate as possible however.

The two extremes:

- h(v) = 0.
- h(v) = actual distance.

A simple heuristic for A* is Manhattan Distance. This works well for the 15-tile puzzle, and Pac-man. There are better heuristics, but Manhattan Distance is good and simple enough for you to code up yourself.

Use A* to solve the following 5-tile puzzle, using Manhattan distance as your heuristic.

Starting state:

5	2	
1	4	3

Goal State:

1	2	3
4	5	

Search space size = 6! = 720