Homework 5

1. Using the quicksort algorithm discussed in class, show how quicksort would work on the following array:

   [6 10 3 2 12 14 11 5]

   In particular, explicitly list **each** call to `quicksort` and `partition` that is made, along with the arguments. Also show the contents of the entire array at the end of every call to `partition`. Make it clear which elements are swapped and when.

   Do this using a recursion diagram similar to the one we used for mergesort, showing the order that calls are made. Here's a start:

```
quicksort([6  10  3  2  12  14  11  5], 0, 7)
    pivot is 6
    swapping positions 1 (10) and 7 (5)
    swapping positions 0 (6) and 3 (2)
    array is now [2  5  3  6  12  14  11 10]
    quicksort([2  5  3  6  12  14  11  10], 0, 2)
        etc ….
```

2. We learned that on average, quicksort is an O(n log n) sort, however, it can degrade to $O(n^2)$ in certain cases. Here, we will study what happens in one of those cases, specifically when quicksort is asked to sort an array that is already sorted.

   a. Suppose quicksort is asked to sort an array with a single number in it. For instance, suppose we call quicksort([0], 0, 0). How many total calls are there to `quicksort` (including the initial call and any recursive calls?) How many total calls are there to `partition`?

   b. Re-answer the two questions in part (a) for an array that is already sorted that has a length of 2, such as [0, 1].

   c. Re-answer the two questions in part (a) for an array that is already sorted that has a length of 3, such as [0, 1, 2]. Hopefully you see a pattern now.

   d. Generalize your answers for an already-sorted array of length *n* (total number of calls to quicksort, number of calls to partition). Show your work, and give an exact answer in terms of n, not big-oh.

3. Heaps

   Refresher on heaps:

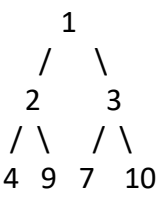
   To insert an item into the heap, put it at the end of the heap (end of the array, or equivalently the "last" spot in the tree, top to bottom, left to right). Then use the heap-up algorithm to move it up the tree until it's in a valid location with respect to its parent and children.

   To delete an item from a heap (you can only delete the largest item in a max-heap, which will be the root), move the last item in the heap into the root's place, then use the heap-down algorithm to move it down the tree until it's in a valid location with respect to its parent and children.

   a. Add the following items in order to an empty min-heap, drawing the heap as a binary tree at the end of each insertion: 5, 2, 6, 4, 1. [So you should have 5 pictures in total].

   b. Draw the heap at the end of part (a) as an array (remember, heaps stored in arrays are indexed starting from 1).

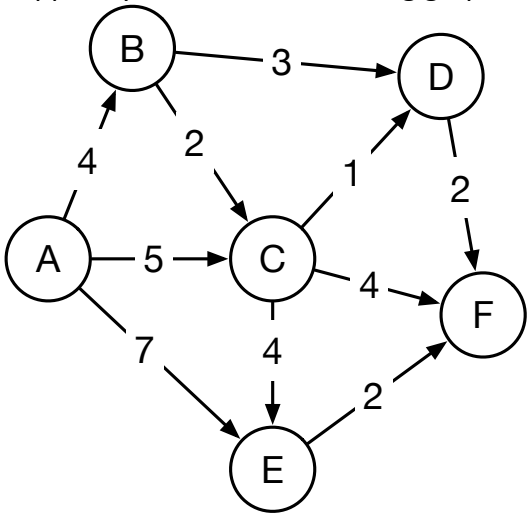
   c. Suppose we start with the following heap:



      Draw the heap after deleting 10.

   d. Continuing from part (c), draw the heap from the end of that problem after deleting the new root [which should be 2, since it's the next-biggest element].

4. Suppose you have the following graph:



Use Dijkstra's algorithm to find the shortest path from A to F.

Fill in the following tables (or rewrite them on your own paper), showing the dist and prev tables, and the order that vertices are visited (this is not the final path, this is the order that the vertices come off the priority queue).

When you update an item in the tables, lightly cross out the old item so I can read it.

Order that we visit vertices: _____

Dist table		Prev table	
dist[A]:	_____	prev[A]:	_____
dist[B]:	_____	prev[B]:	_____
dist[C]:	_____	prev[C]:	_____
dist[D]:	_____	prev[D]:	_____
dist[E]:	_____	prev[E]:	_____
dist[F]:	_____	prev[F]:	_____

Final shortest path distance: _____

Final shortest path: _____