Data Structures in Java - Homework 9

Problem 1

7.20 Using the quicksort implementation in this chapter, determine the running time of quicksort for
a. sorted input
b. reverse-ordered input
c. random input

a) Answer: $O(Nlog(N))$

b) Answer: $O(Nlog(N))$

c) Answer: Worst case $O(N^2)$ with average $O(Nlog(N))$

Problem 2

7.21 Repeat Exercise 7.20 when the pivot is chosen as
a. the first element
b. the larger of the first two distinct elements
c. a random element
*d. the average of all elements in the set

a) Answer:

Sorted input: $O(N^2)$

Reverse-ordered input: $O(N^2)$

Random input: Worst case $O(N^2)$ with average $O(Nlog(N))$

b) Answer:

Sorted input: $O(N^2)$

Reverse-ordered input: $O(N^2)$

Random input: Worst case $O(N^2)$ with average $O(Nlog(N))$

c) Answer:

Sorted input: Worst case $O(N^2)$ with average $O(Nlog(N))$

Reverse-ordered input: Worst case $O(N^2)$ with average $O(Nlog(N))$

Random input: Worst case $O(N^2)$ with average $O(Nlog(N))$

d) Answer: (Averages only if uniformly distributed)

Sorted input: Worst case $O(N^2)$ with average $O(Nlog(N))$

Reverse-ordered input: Worst case $O(N^2)$ with average $O(Nlog(N))$

Random input: Worst case $O(N^2)$ with average $O(Nlog(N))$

Problem 3

7.28 When implementing quicksort, if the array contains lots of duplicates, it may be better to perform a three-way partition (into elements less than, equal to, and greater than the pivot), to make smaller recursive calls. Assume three-way comparisons, as provided by the `compareTo` method.

a. Give an algorithm that performs a three-way in-place partition of an N-element subarray using only $N - 1$ three-way comparisons. If there are d items equal to the pivot, you may use d additional `Comparable` swaps, above and beyond

the two-way partitioning algorithm. (*Hint:* As `i` and `j` move toward each other, maintain five groups of elements as shown below):

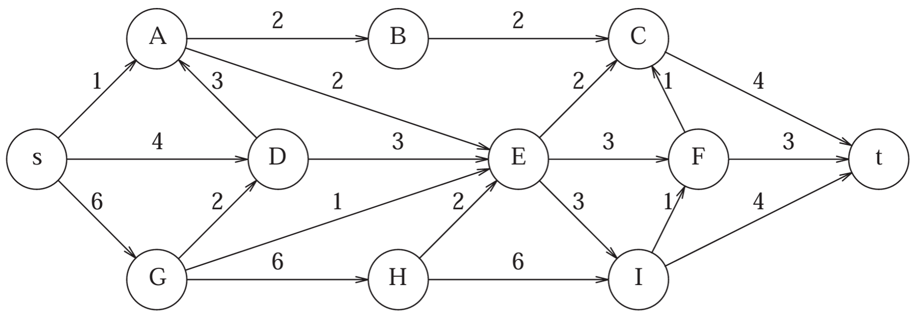
```
EQUAL  SMALL  UNKNOWN  LARGE  EQUAL
             i         j
```

```
public class P3 {
    public static <T extends Comparable<? super T>>
        int[] threewayPartition(T[] theList, int start, int end) {

        if (start-end+1 > theList.length || start >= end) {
            return new int[]{start, start};
        }

        T pivot = theList[start];

        int i = start+1;
        int j = end;
        int leftEq = start;
        int rightEq = end+1;
        boolean iWaiting = false;
        boolean jWaiting = false;

        while (i <= j) {
            if (!iWaiting)
            {
                int iCmp = theList[i].compareTo(pivot);

                if (iCmp > 0) {
                    iWaiting = true;
                }
                else if (iCmp < 0) {
                    i++;
                }
                else { // equal to pivot
                    swap(theList, i++, ++leftEq);
                }
            }
```

```java
            if (iWaiting && i >= j) { break; }

            if (!jWaiting && j > i)
            {
                int jCmp = theList[j].compareTo(pivot);

                if (jCmp < 0) {
                    jWaiting = true;
                }
                else if (jCmp > 0) {
                    j--;
                }
                else { // equal to pivot
                    swap(theList, j--, --rightEq);
                }
            }

            if (iWaiting && jWaiting) {
                swap(theList, i++, j--);
                iWaiting = false;
                jWaiting = false;
            }
        }

        // Swapping Equals back into the middle
        j = i--;

        for (int a=start; a<=leftEq; a++) {
            swap(theList, a, i--);
        }

        for (int b=end; b>=rightEq; b--) {
            swap(theList, b, j++);
        }

        return new int[]{i+1, j-1};
    }

    public static <T extends Comparable<? super T>>
        void swap(T[] theList, int a, int b) {
        T tmp = theList[a];
        theList[a] = theList[b];
        theList[b] = tmp;
    }
}
```

Problem 4

9.1 Find a topological ordering for the graph in Figure 9.81.

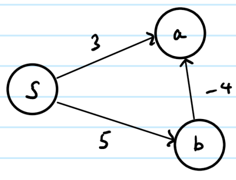


Answer: s, G, D, A, B, H, E, I, F, C, t

Problem 5

9.7 a. Give an example where Dijkstra's algorithm gives the wrong answer in the presence of a negative edge but no negative-cost cycle.

Answer:



Starting at S:

min unknown adjacent vertex = a

Locks in a.known = true

Locks in a.distance = 3

Next min unknown adjacent vertex = b

Locks in b.known = true

Locks in b.distance = 5

Path to a is present but a is known already.

∴ Dijkstra concludes a.distance = 3 and b.distance = 5

However $cost(S \to b \to a) = 5 - 4$

$= 1 < 3$

So Dijkstra is wrong.

Problem 6

9.10 a. Explain how to modify Dijkstra's algorithm to produce a count of the number of different minimum paths from *v* to *w*.

Implement Dijkstra with an extra int counter as an attribute of each Vertex, representing the number of paths that can get to that vertex with that (temporary) cost/distance.

Initialize counter = 0 for all vertices.

Set starting vertex v.counter = 1.

When going through each adjacent vertex B of a vertex A:

```
if ( A.cost + cost(from A to B) < B.cost ) {
    B.counter = A.counter;
    B.cost = A.cost + cost(from A to B);
}
else if ( A.cost + cost(from A to B) == B.cost ) {
    B.counter += A.counter;
}
```

Problem 7

9.38 You are given a set of *N* sticks, which are lying on top of each other in some configuration. Each stick is specified by its two endpoints; each endpoint is an ordered triple giving its *x*, *y*, and *z* coordinates; no stick is vertical. A stick may be picked up only if there is no stick on top of it.

a. Explain how to write a routine that takes two sticks *a* and *b* and reports whether *a* is above, below, or unrelated to *b*. (This has nothing to do with graph theory.)

b. Give an algorithm that determines whether it is possible to pick up all the sticks, and if so, provides a sequence of stick pickups that accomplishes this.

a) Answer:

Find the intersection of sticks a and b in the x-y plane. If there is no x-y plane intersection, a and b are unrelated.

Find the z-coordinates of both a and b at that intersection (x, y). Whichever stick's z-coordinate is larger here indicates that it is on top of the other.

b) Answer:

Model the stack of sticks as a directed graph, where each vertex is a stick and each edge is when the sticks (represented by the edge's vertices) overlap directed from the upper stick to lower stick.

If the resulting graph is cyclic, then it is not possible to pick up all the sticks. Otherwise it is possible, then topologically sort the graph to get a topological ordering for the sequence that accomplishes it.