1. **(6 points) Just sort() it!**
   What method is called by Comparable Interface when using sort()?
   __________________________
   What method is called by Comparator when using sort()?
   __________________________
   List 2 reasons why would you use Comparator instead of Comparable?
   a. __________________________
   b. __________________________

   For Questions 2-8: You don’t need to show all the steps just the final answers.
   If you need to write out the steps, do it on Page 7 or 8 (or an extra sheet).

2. **(5 points) It’s all about the pivot!** Partition the following array based on the first element and median of the first three elements

<table>
<thead>
<tr>
<th>Original Array</th>
<th>6</th>
<th>8</th>
<th>7</th>
<th>1</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partition based on first element as the pivot of the list (6)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Partition based on median of 1st 3 elements as the pivot of the list (6, 8, 7)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3. **(6 points) Prioritize!** Consider the following priority queue. Show the result of deleting 16 (draw this tree) and then adding 15 (also draw this tree) (including resulting sinks/swims). You should draw two trees for this problem. Use priority queue order with highest number.

   ![Priority Queue Tree](tree1.png)

4. **(6 points) Binary Search:** Consider the following binary search tree. Show the result of deleting 17 (draw this tree) and then adding 13 (draw this tree). You do not need to balance but you should follow correct deletion and insertion process. You should draw two trees (do not use tombstones).

   ![Binary Search Tree](tree2.png)
5. **(6 points) 2-3 Trees Search**: Consider the following 2-3 tree. Show the result of adding I, then B, then F. Draw a separate tree for each addition (for a total of 3 trees that you draw). (You need to follow the correct insertion process for each step and show the final result).

![2-3 Tree](image)

**Anatomy of a 2-3 search tree**

6. **(6 points) Red-Black Tree**: Draw the right-leaning red-black tree equivalent to the 2-3 tree below. Indicate a red link by a double line.

![Red-Black Tree](image)

**Anatomy of a 2-3 search tree**

7. **(4 points) Tic-Tac-Toe**: Consider the game of 3x3 Tic-Tac-Toe. Using symmetry (rotation/flips), what is the minimum number of unique first moves you have to consider? _________

   For each first move, what is the unique number of responses (considering symmetry/rotation/flips), you have to consider? ______________

8. **(6 points) Minimum Spanning Tree**

   Using Kruskal’s or Prim’s algorithm (starting with vertex 0), number the edges in the order you will add them to the Minimum Spanning Tree in the figure on the right. Consider the weight to the edges to be:

   - 0-7: 61
   - 2-3: 71
   - 1-7: 91
   - 0-2: 11
   - 5-7: 82
   - 1-3: 92
   - 1-5: 23
   - 2-7: 43
   - 4-5: 53
   - 1-2: 63
   - 4-7: 73
   - 0-4: 83
   - 6-2: 14
   - 3-6: 25
   - 6-0: 85
   - 6-4: 39

   Checkmark the algorithm you used: □ Kruskal’s or □ Prim’s?
9. (10 points) APT Clone 1: LengthSort

Space is tight so you want to eliminate duplicates and sort Strings by their length (shortest first) so you can store as many as possible. You are given an array of Strings with possible duplicates and you need to return an array of Strings that contains exactly one occurrence of each unique string with the shortest string first. If two strings are the same length then return them in alphabetical order. Consider these strings (quotes are for clarity, they're not part of the strings).

{{"de", "f", "abc"}} returns {{"f", "de", "abc"}} because the shortest length comes first.
{{"bcde", "wyz", "bcde"}} returns {{"wyz", "bcde"}} because duplicates are eliminated.
{{"bcd", "abc", "aaaa"}} returns {{"abc", "bcd", "aaaa"}} because alphabetical order breaks ties.

Note you may assume that String has the following built in methods:

str1.length() returns the length of String str1
str1.compareTo(str2) returns +1 if str1 > str2, 0 if they are equal, −1 if str1 < str2.

```java
public class LengthSort {
    public String[] sortByLength(String[] s) {
        // Implementation goes here
    }
}
```
10. (10 points) APT Clone 2: EmergencyNetwork

In EmergencyNetwork, email was not available causing delays in propagating information. Assume that email is now available and the rest of the problem is similar to EmergencyNetwork... You want all of your employees to be notified in case of an emergency as quickly as possible. Your company is organized in a tree-like structure: each employee has exactly one direct boss, no employee is his own direct or indirect boss, and every employee is your direct or indirect subordinate. You will send an email to all your direct subordinates, all at once. After hearing the emergency notification, each subordinate must notify all of his/her direct subordinates, all at once. The process continues this way until everyone has heard the emergency notification. Each person may only email direct subordinates, and each email takes exactly one minute (regardless of the number of employees emailed). Note that there may be multiple emails taking place simultaneously. Return the minimum amount of time, in minutes, required for this process to be completed.

Employees will be numbered starting from 1, while you will be numbered 0. Furthermore, every boss is numbered lower than his or her direct subordinates. You are given a int[] bosses, the ith element of which is the direct boss of employee i. The first element of bosses will be -1, since the manager has no bosses.

{-1, 0, 0} Returns: 1
{-1, 0, 0, 2, 2} returns: 2 (you email employees 1 and 2, employee 2 emails employees 4 & 5)
{-1, 0, 1, 2, 3} returns: 4 (everyone has exactly one subordinate, resulting in a chain of emails)

You may review and use any part of the following code from Professor Astrachan’s solution of the original EmergencyNetwork (or you may start from scratch). Warning: This code does not exactly solve this problem but it solves a similar problem and can help you with the solution.

```java
public class EmergencyNetwork {
    public int lagTime(int[] bosses) {
        return minWithBoss(0, bosses);
    }

    public int minWithBoss(int supervisor, int[] bosses) {
        ArrayList<Integer> subs = new ArrayList<Integer>();
        // recursive call for all subordinates
        for (int k = 0; k < bosses.length; k++) {
            if (bosses[k] == supervisor) {
                subs.add(minWithBoss(k, bosses));
            }
        }
        if (subs.size() == 0) return 0; // leaf node
        Collections.sort(subs, Collections.reverseOrder()); // sort subs by time
        int max = 1 + subs.get(0);
        for (int k = 1; k < subs.size(); k++) {
            int now = k + 1 + subs.get(k); // add 1 min for each (sequential) call
            if (now > max) max = now;
        }
        return max;
    }
}
```
11. (10 points) APT Clone 3: DukeGrid

CompSci 201 TAs invented a game called DukeGrid which involves counting the number of ways you can move a piece called Duke from the bottom left corner of an \( \text{NxN} \) Grid to the top right. At each step, Duke can take a step up, to the right, or diagonally (up and right). Duke may never move left, down, or diagonally that involves left or down. In other words, Duke must always take a step closer to its final destination (straight or diagonal), one step at a time. See visualizations below...

2 returns 3 because you can move up-right, right-up, or diagonally.

3 returns 13 because there are 5 ways after moving up, 5 ways after moving right, and 3 ways after taking a diagonal step. Another way to calculate is that there are 6 \((4!/(2!\ 2!))\) ways without a diagonal move, 6 ways with one diagonal move, and 1 way with two diagonal moves.

```java
public class DukeGrid {
    public int numPaths(int N) {
        // Implementation goes here
    }
}
```
Recitation Room: ________ & Time: ________

(1 point) Community Standard acknowledgment (signature): ________________________________