Name: _____ Andrew Id: _____

15-121 Fall 2021 Quiz 11

Up to 25 minutes. Show your work. No calculators, no notes, no books, no computers, no other people.

1. **Binary Heap**

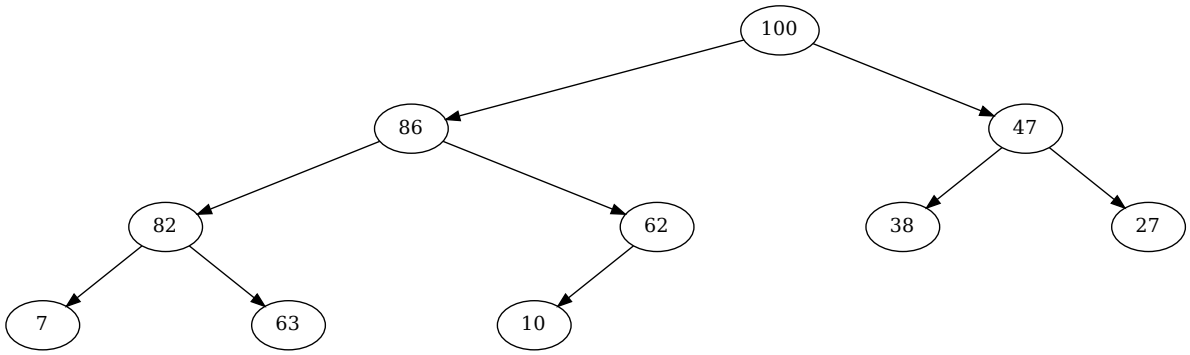
   Consider the following array representation of a binary heap:

94	84	78	70	76	52	72	57	54	71				

   (a) (3 points) Draw the heap represented by this array.

   (b) (3 points) Give the array after adding 85 to the heap. Write your final answer in the boxes below.

2. (a) (3 points) Consider the following heap:



Write this heap in array form, assuming that the root is stored in location 0. Write your final answer in the boxes below.

(b) (3 points) Building on your answer from part a, give the array after calling `removeMax()` on the heap. Write your final answer in the boxes below.

3. (8 points) **Sets and Maps and Efficiency... Oh My!**

Be sure to note the efficiency requirement described at the bottom of this page.

In this question you will write a `Names` class which, given an array of names, is able to determine the most common name found in the array. In the event that there is a tie, both items are returned. (In order to handle this, you will always return a `Set` containing the answers. If there is only one name that is most common, then the `Set` will contain only one item.)

For example...

- Given the array `["Jane", "Aaron", "Cindy", "Aaron"]`, your code will return a set containing the string `"Aaron"`.
- Given the array `["Jane", "Aaron", "Cindy", "Aaron", "Jane"]`, your code will return a set containing the strings `"Aaron"` and `"Jane"`.

Consider the following testcase to help you understand how the class is organized and what each of the methods should do:

```
public static void main(String[] args) {
    System.out.print("Testing Names class...");

    // Test 1
    String[] someNames = { "Jane", "Aaron", "Jane", "Cindy" };
    Names n = new Names();
    n.addNames(someNames);

    Set<String> res = n.mostCommonName();
    if (res.size() != 1) {
        System.out.println("fail");
        return;
    }
    if (!res.contains("Jane")) {
        System.out.println("fail");
        return;
    }

    // Test 2
    String[] someNames2 = { "Jane", "Aaron", "Jane", "Cindy", "Aaron" };
    Names n2 = new Names();
    n2.addNames(someNames2);

    Set<String> res2 = n2.mostCommonName();
    if (res2.size() != 2) {
        System.out.println("fail");
        return;
    }
    if (!res2.contains("Jane")) {
        System.out.println("fail");
        return;
    }
    if (!res2.contains("Aaron")) {
        System.out.println("fail");
        return;
    }
    System.out.println("pass");
}
```

On the following page, write the `Names` class.

Important Note: Your solutions to `addNames` and `mostCommonName` must each be $O(N)$ or better. (So think carefully about which Java data structure you should use to solve this.)

Solution space for Question 3.