Name: ______________________ Andrew Id: ______________

15-121 Fall 2022 Quiz 10

Up to 20 minutes. Show your work. No calculators, no notes, no books, no computers, no other people.

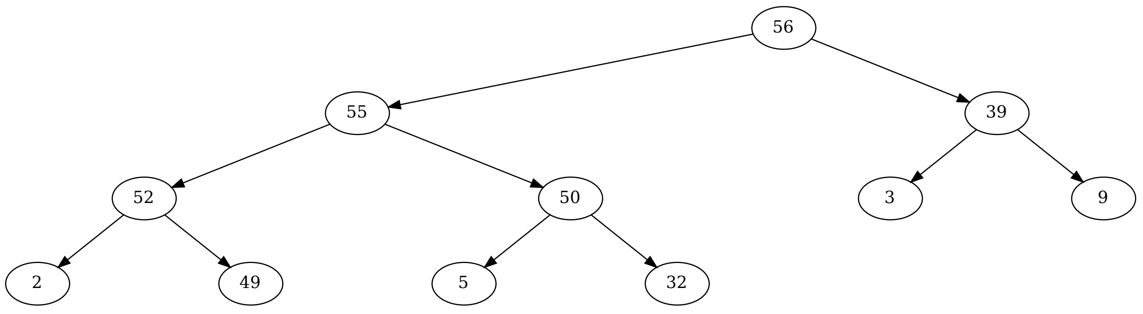
1. **Binary Heap**

   Consider the following array representation of a binary heap:

56	55	39	52	50	3	9	2	49	5	32			

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

Solution:



Grading:

- +0.5 if they have the correct number of nodes and "shape" of the tree.
- +0.5 if 94 is the root of their tree.
- +0.5 if they got the last two bullets, and their tree is a valid heap. (Parents are always greater than all children and grandchildren.)
- +1.5 if they got the last three bullets, and all nodes are in the correct locations.

(b) (3 points) Give the array after adding 45 to the heap. Write your final answer in the boxes below.

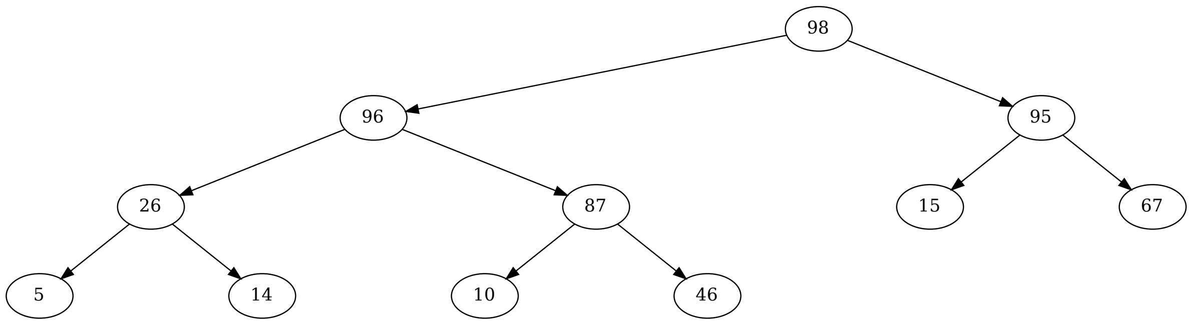
Solution:

56	55	45	52	50	39	9	2	49	5	32	3		

Grading:

- -0.5 for each incorrect node, stop at 0.
- Minimum score of +0.5 is given if they have an answer with the correct number of entries.

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.

Solution:

98	96	95	26	87	15	67	5	14	10	46			

Grading:

- -0.5 for each incorrect node, stop at 0.
- Minimum score of +0.5 is given if they have an answer with the correct number of entries.

(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.

Solution:

96	87	95	26	46	15	67	5	14	10				

Grading:

- -0.5 for each incorrect node, stop at 0.
- Minimum score of +1 is given if they have an answer with the correct number of entries and the first entry from their answer in part (a) has been removed from the array.

3. (3 points) **RPN**

Assume I am building a simple calculator that has the following behavior. When I see an integer, I push its value onto a stack. When I see an operator (`+, -, *, /`), I pop the stack once, and store the result in a variable called `op2`. I then pop the stack a second time, and store the result in a variable called `op1`; I then perform the arithmetic operation specified by that operator (`op1` operator `op2`) and push the result of that operation back onto the stack.

For example, if I entered `7 2 -` , `peek()` would return `5` since `7` would be pushed onto the stack, then `2`, and then the `-` would cause `2` to be popped and stored in `op2`, `7` to be popped and stored in `op1`, and the result of `7-2`, which is `5`, would be pushed onto the stack.

Tell me what value is returned by peek() after the following sequence of values is processed. Write your answer, and nothing else, in the box below, but show your work in the empty space next to the box.

`4 7 9 - 2 3 + * +`

Solution:
-6
Partial credit for 14 (swapped order of op1/op2 on the -)

4. **Buggy Sets**

Consider the following code for a Book class:

```
public class Book implements Comparable<Book> {
    private String title;
    private String author;
    private int isbn;

    public Book(String title, String author, int isbn) {
        this.title = title;
        this.author = author;
        this.isbn = isbn;
    }

    // Natural order for books is defined by the title
    @Override
    public int compareTo(Book arg0) {
        return this.title.compareTo(arg0.title);
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        if (this.title != null) {
            result = prime * result + this.title.hashCode();
        }
        if (this.author != null) {
            result = prime * result + this.author.hashCode();
        }
        result = prime * result + this.isbn;
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null || !(obj instanceof Book)) {
            return false;
        }
        return this.compareTo((Book) obj) == 0; // ensure equals and compareTo are consistent
    }
}
```

After this code is deployed in a real product for a large library, the following bug report comes in:

> After we made Book objects for all two million books in the library, we added them all to a HashSet. Strangely, some of the books "disappeared" inside the HashSet. (Meaning that we called .add to add them to the HashSet, but they never actually got added.) In total, we've found that when we add all two million unique books to the HashSet, about 3 books disappear in this way.
>
> One of the librarians noticed that it seems like this only happens when two books have the same title, but different author. Strangely, though, it doesn't happen every time this is the case: Plenty of books with the same title and different author work just fine. Only three of them seem to disappear.

(a) (2 points) What is the bug in the code that causes this behavior?

Solution: The equals() method and the hashCode method aren't consistent. Equals uses only title, and hashCode uses title, author, and ISBN.

(b) (3 points) Fix the bug in the code above. You can either modify the code directly on the previous page, or rewrite any methods that you need to here. (Note: There are multiple ways to fix the bug, but you should choose the way you think is best.)

Solution: There are many ways to fix this. The preferred way is to modify compareTo. Also acceptable would be a new version of equals that covers all three items instead of call compareTo. Least acceptable is modifying hashCode to only handle title.

```java
public int compareTo(Book arg0) {
    int ret;
    ret = this.title.compareTo(arg0.title);
    if (ret != 0) {
        return ret;
    }
    ret = this.author.compareTo(arg0.author);
    if (ret != 0) {
        return ret;
    }
    return this.isbn - arg0.isbn;
}
```

(c) (2 points (bonus)) Explain how/why the bug causes this behavior. Make sure your explanation covers why the bug happens in some cases, but not others, as mentioned in the bug report.

Solution: The bug is manifested if two books that have the same title end up in the same bucket in the hash table. In that case, during add, the second book would not get added to the table because the HashSet won't add what equals tells it is a duplicate item. It doesn't always happen, though, because usually books with the same title and different authors would end up in different buckets, and in that case the equals method bug won't be triggered.