**15-121**
**Fall 2021 Final Exam**
**December 5, 2021**

**Name:**

**Andrew ID:**

- You may not use any books, notes, or electronic devices during this exam.
- Show your work on the exam to receive credit.
- You may complete the problems in any order you'd like; you may wish to start with the free response problems, which are worth most of the credit.
- All code samples run without crashing unless we state otherwise.
- Assume any imports are already included as required.

Don't write anything in the table below.

| Question | Points | Score |
|---|---|---|
| 1 | 13 | |
| 2 | 9 | |
| 3 | 8 | |
| 4 | 10 | |
| 5 | 6 | |
| 6 | 6 | |
| 7 | 6 | |
| 8 | 10 | |
| 9 | 5 | |
| 10 | 10 | |
| 11 | 20 | |
| Total: | 103 | |

There are 103 points on this exam, but your final score will be out of 100. (So, 3 points are effectively bonus.) However, the highest score than can be received on this exam is still 100.

1. **Multiple Choice / Short Answer**

(a) (2 points) Imagine that you have a class, `Dog` that inherits from another class, `Pet`. In `Dog` you override `toString`. Which of the following lines of code, when executed inside the `toString` of `Dog`, will call the `toString` from `Pet`?

- ◯ `super().toString()`
- ◯ `super(Pet).toString()`
- ◯ `super(Dog).toString()`
- ◯ `Dog.toString()`
- ◯ `super.toString()`
- ◯ `Pet.toString()`

(b) (2 points) How do you efficiently find the smallest item in a max heap?

- ◯ It is located at the end (i.e., the furthest right node in the last level)
- ◯ It is located at the root
- ◯ It is not possible to find the smallest item
- ◯ You keep going "left" until you can't go left anymore
- ◯ You need to check every node in the data structure
- ◯ You find the in-order predecessor of the root

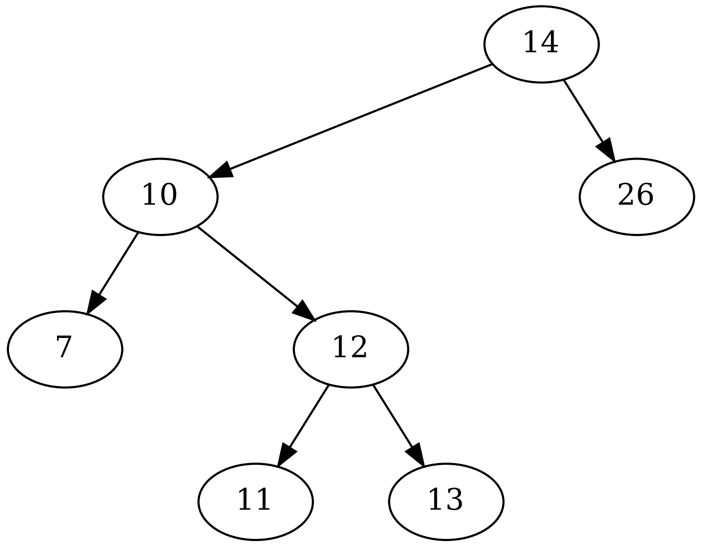
(c) (2 points) How do you efficiently find the smallest item in a binary search tree?

- ◯ It is located at the end (i.e., the furthest right node in the last level)
- ◯ It is located at the root
- ◯ It is not possible to find the smallest item
- ◯ You keep going "left" until you can't go left anymore
- ◯ You need to check every node in the data structure
- ◯ You find the in-order predecessor of the root

(d) (2 points) Assuming that the array has enough space to hold the items needed, if you implement a queue with an array, can all the operations be O(1)? Why or why not?

(e) (2 points) The worst-case efficiency of inserting into a hash table is technically $O(N)$, but in practice it is usually $O(1)$. Why?

(f) (3 points) Consider the following binary tree:



Is it a full binary tree? ◯ Yes ◯ No

Is it a complete binary tree? ◯ Yes ◯ No

Is it a perfect binary tree? ◯ Yes ◯ No

2. **Code Tracing**. Indicate what the code will print. Place your answer (and nothing else) in the box below the code.

   (a) (3 points) CT1

```
public class A {
    protected int a;
    protected int b;

    public A(int a, int b) {
        this.a = b;
        this.b = a;
        System.out.println("A1: " + a + " " + b);
    }

    public int getOne() {
        return this.a;
    }

    public int getTwo() {
        return this.b;
    }
}

public class B extends A {
    public B(int c, int d) {
        super(d, c);
        System.out.println("B1: " + a + " " + b);
        a = d + 4;
        b = c - 4;
        System.out.println("B2: " + this.a + " " + this.b);
    }
    public static void main(String[] args) {
        B bob = new B(12, 21);
    }
}
```

(b) (3 points) CT2

Assume the existence of a doubly linked list. (It is a simplified version of what was in the homework. If you don't remember what that looks like, please see the handout.) Assuming that the following two methods are added to the DoublyLinkedList class, what will the output be when `main` is executed?

```
public void mystery(DoubleListNode<ListType> node) {
    node.next.prev = node.prev;
    node.prev.next = node.next;
    node.next = head.next;
    node.prev = head.next.prev;
    head.next = node;
    node.next.prev = node;
}

public static void main(String[] args) {
    DoublyLinkedList<Integer> d = new DoublyLinkedList<Integer>();
    d.addBefore(null, 5);
    DoubleListNode<Integer> tmp = d.getHead();
    d.addBefore(tmp, 10);
    d.addBefore(tmp, 15);
    d.addAfter(tmp, 20);
    d.addAfter(tmp, 25);
    d.addBefore(tmp, 30);
    System.out.println(d);
    d.mystery(tmp);
    System.out.println(d);
}
```

(c) (3 points) CT3

```
public class CT3 {
    public static void main(String[] args) {
        int b = 8;
        int a = 2;

        System.out.println(--b - a++ + ++b - --a + a++ - b++);
        System.out.println(a);
        System.out.println(b);
    }
}
```

Note that there are three prints.

3. (8 points) **Big-Oh**

Determine the big-oh runtime of each of the following, in terms of N. Write your answer, and nothing else, in the box next to each method.

```java
// N is the length of arr
public int f1(String[] arr) {
    HashMap<String, Integer> h = new HashMap<String, Integer>();
    int a = 0;
    for (String item : arr) {
        int v = 1;
        if (h.containsKey(item)) {
            v = h.get(item) + 1;
        }
        h.put(item, v);
        if (v > a) {
            a = v;
        }
    }
    return a;
}
```

```java
// We want the Big-O of this function.  N is the length of arr.
public static int f2(String[] arr) {
    int a = 0;
    for (String item : arr) {
        int v = helper(arr, item);
        if (v > a) {
            a = v;
        }
    }
    return a;
}

public static int helper(String[] arr, String item) {
    int i = 0;
    for (String s : arr) {
        if (s.equals(item)) {
            i++;
        }
    }
    return i;
}
```

```java
// N is the length of arr
public static int f3(String[] arr) {
    TreeMap<String, Integer> h = new TreeMap<String, Integer>();
    int a = 0;
    for (String item : arr) {
        int v = 1;
        if (h.containsKey(item)) {
            v = h.get(item) + 1;
        }
        h.put(item, v);
        if (v > a) {
            a = v;
        }
    }
    return a;
}
```

```java
// N is the length of arr
public static int f4(String[] arr) {
    HashSet<String> h = new HashSet<String>(Arrays.asList(arr));
    return arr.length - h.size();
}
```

4. **Linked Lists**

Consider the following program that creates a linked list. You may assume that the `ListNode` class exists and was defined as in class and the handout of this exam.

```
1  public class LinkedListCT {
2      public static void main(String[] args) {
3          ListNode<String> head = new ListNode<String>("P");
4          ListNode<String> t = new ListNode<String>("V");
5          ListNode<String> a = null;
6          t.next = head;
7          head = new ListNode<String>("M");
8          head.next = t;
9          t.next = new ListNode<String>("Z");
10         a = new ListNode<String>("T");
11         a.next = head.next;
12         head.next = a;
13         t.next.next = head;
14         a = head.next;
15         head = head.next.next;
16         a.next = null;
17     }
18 }
```

Draw the state of the linked list starting at `head` after the execution of each specified line of code.

(a) (2 points) After Line 5

(b) (2 points) After Line 6

(c) (2 points) After Line 8

(d) (2 points) After Line 12

(e) (2 points) After Line 16

5. **Binary Search Trees**

   Imagine you are constructing a binary search tree of integers, and the following integers are added in the following order:

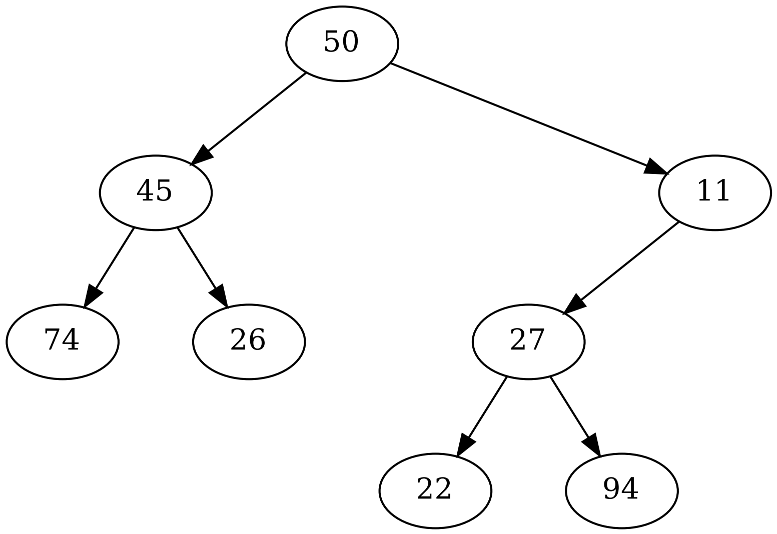
   14, 40, 37, 50, 26, 29, 62, 42, 53, 44

   (a) (3 points) Draw the resulting binary search tree.

   (b) (3 points) Assuming you are using the in-order successor removal technique discussed in class, draw the state of your tree from (a) after 40 has been removed from it.

6. **Not-Binary Search Trees**

The traversal orders (in-order, pre-order, and post-order) can be used on any kind of binary tree, not just a binary search tree. Consider the following tree (which is *not* a binary search tree, but that doesn't matter much for this problem):



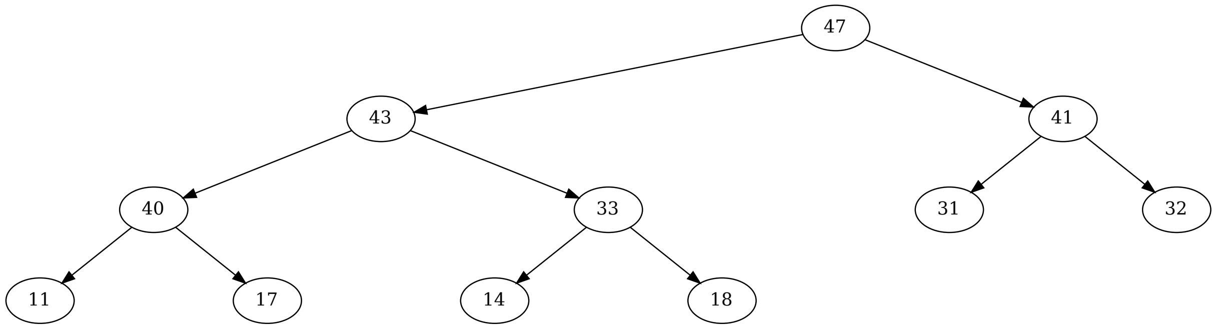
(a) (2 points) Assuming the tree is traversed *in-order* and the nodes printed, what is the resulting sequence? (Assume that left is followed before right.)

(b) (2 points) Assuming the tree is traversed *pre-order* and the nodes printed, what is the resulting sequence? (Assume that left is followed before right.)

(c) (2 points) Assuming the tree is traversed *post-order* and the nodes printed, what is the resulting sequence? (Assume that left is followed before right.)

7. **Binary Heap**

Consider the following heap:



(a) (3 points) Draw the heap after adding 46 to it.

(b) (3 points) Building on your answer from part a, draw the heap after calling removeMax() on it.

8. **Free Response:** Binary Search Trees

Consider the following code for a binary search tree of Strings. (There is nothing special here, the code is provided just in case you forgot how a binary tree is built.)

```
public class BinarySearchTree {
    private TreeNode root;

    private class TreeNode {
        private String data;
        private TreeNode left;
        private TreeNode right;

        private TreeNode(String data) {
            this.data = data;
        }
    }

    public BinarySearchTree() {
        root = null;
    }

    public void add(String item) {
        root = add(root, item);
    }

    private TreeNode add(TreeNode node, String item) {
        if (node == null) {
            return new TreeNode(item);
        }

        int res = item.compareTo(node.data);
        if (res < 0) {
            node.left = add(node.left, item);
        } else {
            node.right = add(node.right, item);
        }

        return node;
    }
}
```

The question continues on the next page

(a) (5 points) Write the code for a new method in this class, `contains(String item)`, which returns `true` if `item` is in the tree and `false` otherwise. Your solution must be $O(\log N)$ or better.

(b) (5 points) Write the code for a new method in this class, `reverseSortedList()`, which returns an ArrayList containing all of the items from the tree in reverse sorted order (So, Z before A). You may not use anything from either the Collections class or the Arrays class. (So you can't call Collections.sort() or Arrays.sort(). Instead, leverage the sorted nature of the tree.)

9. (5 points) **Free Response**: Linked Lists

Consider the following partial code for a Linked List:

```
public class MyLinkedList<ListType> {
    public ListNode<ListType> head = null;

    public MyLinkedList() {
        this.head = null;
    }

    /* Add a node to the head of the list in O(1) time. */
    public void addHead(ListType item) {
        // Code omitted, but assume this method works properly.
    }

    /* Add a node to the end of the list in O(N) time. */
    public void add(ListType item) {
        // Code omitted, but assume this method works properly.
    }

    /**
     * Add all the items from an ArrayList to the end of this linked list, in order.
     * This function should run in O(N) time.
     *
     * @param arr The ArrayList containing the items to add to the list.
     */
    public void addList(ArrayList<ListType> arr) {
        // You will write this method.
    }

}

public class ListNode<NodeType> {
    private NodeType data;
    public ListNode<NodeType> next;

    public ListNode(NodeType data) {
        this.data = data;
    }

    public NodeType getData() {
        return this.data;
    }
}
```

Write the code for the method `addList`.

```
public void addList(ArrayList<ListType> arr) {
```

10. (10 points) **Free Response:** RPN Calculator

Recall once again the description of the calculator from an earlier exam in this course:

> 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.

Write the method `calculate`, below, which implements this type of calculator. A testcase is provided to help you understand how it should work.

```java
public class RPN {

    // This might be a useful helper function you can use...
    public static boolean isInt(String s) {
        try {
            int num = Integer.parseInt(s);
        } catch (NumberFormatException e) {
            return false;
        }
        return true;
    }

    public static int calculate(String line) {
        Stack<Integer> st = new Stack<Integer>();

        // You need to write this function
    }

    public static void main(String[] args) {
        System.out.print("Testing the calculator...");
        int result = calculate("15 5 - 4 * 10 / 2 1 + +");
        if (result != 7) {
            System.out.println("failed!");
            return;
        }
        System.out.println("passed");
    }
}
```

You will notice that the code above makes use of a `Stack`. `Stack` is included in the Java standard library. Details for how to use one can be found in the handout.

Write your answer on the next page

```
public static int calculate(String line) {
    Stack<Integer> st = new Stack<Integer>();
```

11. (20 points) **Free Response**

In this problem you are going to write a `Movie` class. Here are some important requirements for your class:

- A `Movie` has a title and year of release. When a new `Movie` is created, these items are passed to the constructor. (You must provide the constructor.)
- The only way for code outside of `Movie` to read and modify the attributes of a `Movie` must be using appropriately named getters and setters. (You must provide the getters and setters.)
- A title must not be `null`. If an attempt is made to set the title to something invalid, an `IllegalArgumentException` must be thrown.
- The release year must be 1888 or later. If an attempt is made to set the release year to something invalid, an `IllegalArgumentException` must be thrown.
- When printing a `Movie`, its type and the value of its instance variables should appear. (Such as when `System.out.println(spiderManNwh)` is used.)
- A `Movie` needs to be able to be properly used in `HashSet`s and `HashMap`s.
- An array of `Movie`s must be sortable using `Arrays.sort(someArrayOfMovies)`. The natural order is based on the title, with ties being broken by the release year in descending order. (So, if two movies have the same title, the newer movie is listed first.)

Write the `Movie` class to the above specifications. Pay careful attention to which classes you may need to extend or implement as well as which methods you need to write or override.

Extra space for Question 11.

More extra space for Question 11.