XXX. LAST_NAME, FIRST_NAME

CS136 Final

Fall 2023

- **Do not turn this page until time is called.**
- We will read through the info below together. 🙂👍

- The exam is 8 questions long.
- Each question is 1 page long; Some questions will have helpful code above them.
- The exam is 90 points total.
- The exam is scheduled for 120 minutes.
- No notes are allowed.

- Partial credit will be given. Make sure you try answering all questions!
  Especially Question 8 (the coding question). 🙂👍

- Just like in the homework, we are compiling with Java 8 and importing java.util.*
- Assume code snippets are inside of suitable main methods.

- This exam may be curved, but only up (the curve cannot make your grade worse).
- This exam will be primarily graded on correctness.
- This exam may also be graded (to a lesser extent) on speed, clarity, and robustness.

- `char c = (char) ('A' + 3); // 'D'`
- In PostScript, **def** is "define," **mul** is "multiply," **dup** is "duplicate," **exch** is "exchange," **sqrt** is "square root"
- We can use `ArrayDeque queue;` as a single-ended (regular, typical) queue.
  `queue.add(...)` adds to the back
  `queue.remove()` removes from the front and returns it
- `isEmpty()` returns whether an `ArrayList`, `Stack` or `ArrayDeque` is empty
  `foo.isEmpty()` it is equivalent to `(foo.size() == 0)`

Please sign below.

I am the person whose name is listed at the top of this page.
I have neither given nor received unauthorized help on this exam.

_____

1. **(10 points) Draw the state of the stack and map after these PostScript functions run.**
   NOTE: Functions **ARE** separate.

```
1.0 3.0 add sqrt
```

```
-----
stack + map {                                                    }
```

```
/side 2.0 def /area side side mul def /volume side area mul def
```

```
-----
stack + map {                                                    }
```

2. **(10 points) What could this code print?**
   NOTE: Assume `buildFrequencyTables(...)` does **NOT** "wrap around."

   The last window to consider is:
   aabbbbaa
   --^ ("ba" followed by 'a')

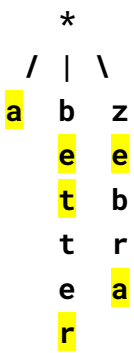
```
String sourceText = "aabbbbaa";
int windowLength = 2;
HashMap<String, HashMap<Character, Integer>> frequencyTables;
frequencyTables = buildFrequencyTables(sourceText, windowLength);
System.out.println(frequencyTables);

// Answer goes below here. vvv
```

```
class LinkedList {
    Node head;
    // list of nodes with values 1, 2, 3 prints as
    // 1 -> 2 -> 3
    String toString();
}

class Node {
    int value;
    Node next;
    Node(int value); // Constructor.
}
```

3. **(10 Points) What does this code print?**
   NOTE: The blocks of code are NOT separate.

```
LinkedList list = new LinkedList();

list.head = new Node(3);

System.out.println(list); //                    <- Answer goes here.

Node node = new Node(5);

System.out.println(list); //                    <- Answer goes here.

list.head.next = node;

System.out.println(list); //                    <- Answer goes here.

node.next = new Node(7);

System.out.println(list); //                    <- Answer goes here.

list.head = list.head.next.next;

System.out.println(list); //                    <- Answer goes here.
```

4. **(10 points) Tries and Traversal.**
   a. **What words are stored in this trie?**
   NOTE: Terminal nodes are highlighted in yellow.



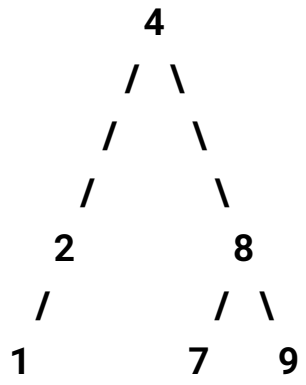

```
//                                                    <- Answer goes here.
```

   b. **Say `root` is the root node of the trie shown above. What does this code print?**
   NOTE: This code uses the same trie `Node` from HW09.

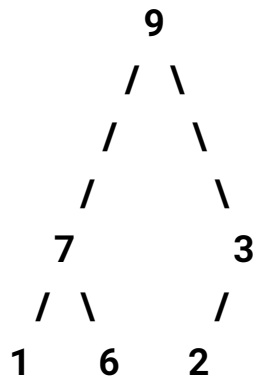
```
// HINT: This is a breadth-first traversal (using a queue).
ArrayDeque<Node> queue = new ArrayDeque<>();
queue.add(root);
while (!queue.isEmpty()) {
    Node curr = queue.remove();
    for (int i = 0; i < 26; ++i) {
        if (curr.children[i] != null) {
            if (curr.children[i].isTerminal) {
                char c = (char)('A' + i);
                System.out.println(c);
            }
            queue.add(curr.children[i]);
        }
    }
}
```

```
// Answer goes below here. vvv
```

5. **(10 points) Binary Search Trees and Max Binary Heaps.**
   a. **Draw the state of this binary search tree after calling `tree.add(5)`.**
      NOTE: Use the naive `add(int i)` from HW10 (**NOT** self-balancing).



   b. **Draw the state of this max binary heap after calling `heap.remove()`.**



```
class ToyMap {
    ArrayList<KeyValuePair>[] buckets;
    ToyMap() {
        buckets = (ArrayList<KeyValuePair>[]) new ArrayList[5];
        for (int i = 0; i < buckets.length; ++i) {
            buckets[i] = new ArrayList<>();
        }
    }
    void put(String key, bool value) {
        int bucketIndex = (2 * key.length()) % buckets.length;
        ArrayList<KeyValuePair> bucket = buckets[bucketIndex];
        for (KeyValuePair pair : bucket) {
            if (key.equals(pair.key)) {
                pair.value = value;
                return;
            }
        }
        bucket.add(new KeyValuePair(key, value));
    }
}

class KeyValuePair {
    String key;
    bool value;
    KeyValuePair(String key, bool value) {
        this.key = key;
        this.value = value;
    }
    public String toString() { return key + "=" + value; }
}
```

6. (10 Points) What does this code print?

```
ToyMap map = new ToyMap();
map.put("Mew", true);
map.put("Arbok", true);
map.put("Muk", true);
map.put("Arbok", false);
System.out.println(Arrays.toString(map.buckets));
//                                          <- Answer goes here.
```

7. **(10 Points) Write the worst case big O runtime in terms of n.**
   NOTE: The blocks of code ARE separate.

```
// Worst-case big O runtime:                                    ?
int result = 0;
while (n > 0) {
    result += n;
    n /= 2;
}
```

```
// Worst-case big O runtime:                                    ?
int result = n;
for (int i = 0; i < 742; ++i) {
    result += i;
}
```

```
// Worst-case big O runtime:                                    ?
String result = "";
for (int i = 0; i < n; ++i) {
    result = result + "asdf";
}
```

```
// Worst-case big O runtime:                                    ?
Stack<Integer> stack = new Stack<>();
stack.push(n);
while (!stack.isEmpty()) {
    int tmp = stack.pop();
    for (int i = 1; i < tmp; ++i) {
        stack.push(i);
    }
}
```

8. **(20 Points) Implement this function.**

```
HINT: HashMap<Integer, Integer> map = new HashMap<>();
HINT: You can use for-each loops to iterate through...
      - the array's elements: for (int i : array) { ... }
      - the map's keys: for (int i : map.keySet()) { ... }
HINT: HashMap functions:
      - map.containsKey(i)
      - map.get(i)
      - map.put(i, ...)
// Returns the number that shows up most often in array (the mode).
// If there are multiple modes, returns (any) one of them.
// If array is empty, crashes.
// { 7, 5, -4, -4, 9 } -> -4
// { 1, 1, 2, 2, 3 } -> 1 (or 2)
// { 8 } -> 8
// Worst-case Runtime: O(n)
int getMode(int[] array) {









}
```