[**Closed book and notes.**] Show all of your work clearly in the space provided. Be sure to **read each problem carefully**. Note that the exam is double sided. **Points given for short answer questions will be scored much like answers to interview questions would be evaluated. An answer that makes you more likely to be hired will receive a higher score.**

1. (8 points) The java.util.Queue interface defines two methods to remove an element from a Queue: remove() and poll().

(a) Explain which element is removed when either of these methods are called.

(b) Explain how these two methods differ.

2. (6 points) The pure stack interface can be implemented so that all methods are $O(1)$. Explain when, if ever, it would make sense to use an implementation that caused one or more of the methods to be $O(n)$.

3. (6 points) When implementing a recursive method, what is a base case? Why is it needed?

4. (a) (20 points) Implement the following method to determine if the parentheses match up. E.g.,

- $((a + b) + c)$ – Good
- $([a + b] + c)$ – Good
- $(a + \{b + c\})$ – Good
- $((a + b) + c$ – Bad
- $([a + b] + c]$ – Bad
- $(a + \{b + c)\}$ – Bad

```
/**
 * Returns true if the parentheses, square brackets, and curly braces match up
 * @param stack An empty object that implements the PureStack interface from class
 * @param expression The expression to be evaluated
 * @return true if the braces are all balanced
 */
public static boolean braceChecker(PureStack<Character> stack, String expression) {
```

4. (b) (5 points) Use big-oh notation to describe the overall worst case time complexity for your algorithm where n is the length of expression. Be sure to explain your reasoning.

5. (7 points) Implement the following method without loops or multiplication:

```
/**
 * Given the number of mountain goats, returns the number of horns.
 * This method assumes each mountain goat has two horns.
 * @param goatCount the number of mountain goats
 * @return the total number of horns on the goats
 */
public static int goatHorns(int goatCount) {
```

6. (a) (7 points) Draw the **perfect** binary tree that would produce the following if the elements were visited using in-order traversal: $8, 3, 7, 17, 1, -1, 12$

(b) (7 points) List the elements in the tree if visited using post-order traversal.

7. (a) (7 points) Draw the **full** binary search tree with the following elements in it: $A, B, C, D, E, F, G, H, I, J, K$

(b) (7 points) Show what the tree would look like if the value at the root of the binary search tree you drew in part (a) is removed. The tree does not need to remain **full**.

8. **(a)** (15 points) Suppose we have a TrinaryTree<E> where each Node<E> contains four attributes: E value, Node<E> lKid, Node<E> cKid, and Node<E> rKid. Implement the recursive version of TrinaryTree.size() method that is called by the method below such that the method returns the number of elements in the tree.

```
public int size() {
    return size(root);
}
```

(b) (5 points) Use big-oh notation to describe the overall worst case time complexity for your algorithm. Be sure to explain your reasoning.