[**You may use a single side of an 8.5×11 in sheet of paper for reference.**] In this exam, wk8.Stack, wk8.Queue, wk8.CircularQueue, and wk10.BST refers respectively to stack, queue, circular queue, and binary search tree implementations created in lecture.

1. (20 points) True/False (**T** or **F**)

____ The Iterator.hasNext() method will throw a NoSuchElementException called on a completely empty collection.

____ The List.remove() method is more efficient than the Iterator.remove() method since the list knows more about the underlying data structure than the iterator.

____ The List.listIterator(index) returns an iterator that begins just before the position index.

____ A ListIterator can be used to navigate both forward and backward over a Collection.

____ The ListIterator is a subinterface of the Iterator interface.

____ The enhanced for loop makes use of an iterator to navigate the collection.

____ Iterator objects throw an IllegalStateException if they are asked to retrieve the next element after all elements have been processed.

____ If a call to java.util.Iterator.remove() is not preceded by a call to next(), an IllegalStateException will be thrown.

____ The Iterator interface declares the iterator() method.

____ The Collection.forEach() method relies on an iterator to navigate the collection.

____ When creating JUnit tests, a method annotated with @BeforeAll is run once before each method annotated with @Test.

____ System tests should be performed after integration tests.

____ java.util.Queue is an interface.

____ java.util.Stack is an interface.

____ It is appropriate to adapt either the java.util.LinkedList or java.util.ArrayList to implement the PureStack interface.

____ A recursive method must have at least one base case.

____ The recursive case is when we call the same method at least once.

____ You cannot always write an iterative solution to a problem that is solvable by recursion.

____ The Set interface extends the Iterable interface.

____ The Map interface extends the Iterable interface.

2. (8 points) Explain concisely and precisely why the asymptotic time complexity for finding where to insert an element into a complete binary search tree is $O(\log(n))$.

3. (8 points) What is the worst possible $O()$ time for `contains()` if the binary search tree is not balanced? Justify your answer.

4. (7 points) For a binary tree (not a binary search tree), what is the $O()$ time for contains()? Justify your answer.

5. (7 points) For arbitrarily long data structures, it makes more sense to implement the Queue interface using a LinkedList instead of an ArrayList as the fundamental data structure on which the implementation is built. Explain why this is the case.

6. (15 points) Implement a non-recursive version of the contains method for a binary search tree. Assume that an empty tree has a `root==null`.

7. (15 points) Suppose we have triangle made of blocks. The topmost row has 1 block, the next row down has 2 blocks, the next row has 3 blocks, and so on. Compute recursively (no loops or multiplication) the total number of blocks in such a triangle with the given number of rows.

```
*    *    *    *
     **   **   **
          ***  ***
               ****
1    3    6    10
```

```
public static int triangle(int rows) {
```

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.