[**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.

1. (10 points) Explain concisely and precisely why the asymptotic time complexity for finding where to insert an element into a complete binary search tree is $O(\log(n))$.

2. (5 points) Why is important for a binary search tree to remain bushy/balanced? Justify your answer.

3. (5 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.

4. A palindrome is a phrase that reads the same forward and backward. For example:

- radar
- Straw? No, too stupid a fad. I put soot on warts.

(a) (20 points) Notice that punctuation, capitalization, and spacing are ignored when determining whether a phrase is a palindrome or not. Use **both** the Stack<E> and Queue<E> classes developed in lecture to implement the following method:

```
// Return true if and only if the phrase is a palindrome
public static boolean isPalindrome(String phrase) {
```

Part **(a)** cont. . .

(b) (10 points) Use big-oh notation to describe the overall worst case time complexity for your algorithm. Be sure to explain your reasoning.

5. (a) (15 points) Suppose a BinaryTree class (not a binary search tree) has an inner class (Node<E>) with the following attributes:

```
private E value;
private Node<E> parent;
private Node<E> left;
private Node<E> right;
```

Implement a recursive method that can be called by the following method to determine the largest element in the tree. If the tree is empty, the method must return `null`.

```
public E maximum() {
  return maximum(root);
}
```

(b) (10 points) Use big-oh notation to describe the overall worst case time complexity for your algorithm. Be sure to explain your reasoning.

6. (a) (15 points) Suppose a BinarySearchTree class has one attribute:

```
private Node<E> root;
```

Where (Node<E>) is an inner class with the following attributes:

```
private E value;
private Node<E> parent;
private Node<E> left;
private Node<E> right;
```

Implement a non-recursive method to determine the largest element in the tree. If the tree is empty, the method must return `null`.

(b) (10 points) Use big-oh notation to describe the overall worst case time complexity for your algorithm. Be sure to explain your reasoning.

Additional work area for any problem. Clearly identify which problem is associated with the work on this page.