1. **Call stack**

   Consider the method `isPrime` that uses a private recursive method to determine if an integer is prime:

```
/**
 * Determines if an integer is prime.
 *
 * @param x
 *          an integer value greater than 2
 * @return true if x is prime, and false otherwise
 */
public static boolean isPrime(int x) {
    return isPrime(x, 2);
}


/**
 * Returns true if x is divisible by y.
 *
 * @param x
 *          an integer value greater than 2
 * @param y
 *          an integer to divide x by
 * @return true if x is divisible by y, and false otherwise
 */
private static boolean isPrime(int x, int y) {
    if (y > Math.sqrt(x)) {
        return true;
    }
    if (x % y == 0) {
        return false;
    }
    return isPrime(x, y + 1);
}
```

   Draw the memory diagram for the method invocation `isPrime(29)` up to the point where a base case of the recursive method is reached.

2. **contains base case**

   See Slide 26 of the lecture.

   What is the base case (or cases) for `contains`?

3. **contains recursive call**

   See Slide 26 of the lecture.

   What is the recursive call for `contains`?

4. **minToFront base case**

   See Slide 33 of the lecture.

   What is the base case (or cases) for `minToFront`?

5. **minToFront recursive call**

   See Slide 33 of the lecture.

   What is the recursive call for `minToFront`?

6. **minToFront updating the list**

   See Slide 33 of the lecture.

   How do you update the list after the recursive call returns in `minToFront`?

7. **selectionSort base case**

   See Slide 42 of the lecture.

   `selectionSort` has an implicit base case (a base case that is not explicitly stated). How would you explicitly state the base case for `selectionSort`?

8. **Jump It**

   What is the lowest cost for the following four Jump It boards?

7	12

7	12	5

3	7	12	5

3	7	6	5

9. See Slide 71 of the lecture.

How does the board get smaller at each recursive call?

10. See Slide 80 of the lecture.

    How can you improve the solution to eliminate unnecessary computation?

11. **What's wrong with the implementation?**

    Consider a slightly modified implementation of `contains`:

```
public class Week10 {

    public static <T> boolean contains(T element, List<T> t) {
        boolean result;
        if (t.size() == 0) {              // base case
            result = false;
        }
        if (t.get(0).equals(element)) {   // base case
            result = true;
        }
        else {                            // recursive call
            result = Week10.contains(element, t.subList(1, t.size()));
        }
        return result;
    }
}
```

    The implementation contains an error; what is it?