(a) Consider each grouping of code below. Identify any errors (there may not be any) in the code and explain what is wrong.

```
ArrayList<int> numbers = new ArrayList<int>();
```

```
List<String> words = new ArrayList<String>();
```

(b) Circle the bullet in front of each statement that accurately describes your instructor's course policies:

- Having a copy of all or part of another student's source code, just to look at for a little help, is considered cheating.
- You must submit every lab assignment in order to pass this course.
- The final exam for this course will be comprehensive.
- If you miss more than three lectures, your grade will be dropped by half a letter grade.
- If you don't do your homework you will not be allowed to attend some classes.
- All students must submit a copy of their lecture notes before the end of the day.
- Lab attendance is required.
- Your instructor is such a nice guy that if you turn in an assignment 45 minutes late, he wouldn't consider it late.
- Your instructor will give you a free lunch if you invite him to lunch.

Do not circle the bullet in front of any statement that incorrectly describes your instructor's course policies.
Feel free to provide an explanation for your answer if you believe the statement to be ambiguous.

Implement the ArrayList.size() and ArrayList.indexOf(Object) methods using the same assumptions that we made in lecture.

Suppose that the LinkedList class that we have been developing in lecture was modified so that it only had one attribute: tail that pointed to the last node in the list. Further, suppose that the Node class was modified so that the next attribute was replaced with a previous attribute that points to the previous node in the list instead of the next node in the list. Implement the contains method for the list

List the methods that are part of a pure stack interface.

Explain how a pure stack interface could be implemented using a LinkedList.

Complete the method below using recursion (no for/while/do-while loops allowed). Hint: Break the problem into three cases: 1) the list is empty, 2) the list is non-empty and the first value matches the target, or 3) the list is non-empty and the first value does not match the target. Use get(0) to return the first element of the list and remove(0) to remove the first element from the list before the recursive call.

```
/**
 * Counts the number of times target is found in data
 * @param data The list of values to be searched over
 * @param target The value to be searched for
 * @return The number of times target is found in data
 */
public static int countMatches(List<Integer> data, int target) {
```

(a) Use big-O notation to describe the overall worst case time complexity for a recursive method that calculates the height of a balanced binary tree (e.g., the height(Node) method we implemented in lecture). Be sure to explain your reasoning.

(b) Use big-O notation to describe the overall worst case time complexity for a non-recursive method returns the largest element in a balanced binary search tree. Be sure to explain your reasoning.

Implement rightRotate().

(a) Suppose the following Integers are added to a hash table with a capacity of 6. Illustrate the resulting data structure. Note that the hashCode() method for the Integer class just returns the value of the integer.

0, 8, 1, 5, 6, 7

(b) What is the load factor, L, for this hash table?