Question 1. (7 points) Write the Java specification and implementation for a public method in the `StringList` class called `remove` that takes a `String str` as a parameter and removes `str` from `strings` if `str` is in the array. The method should return `true` if an element was successfully removed from `strings` and `false` if an element is not removed. The method should shift elements toward the front of the array and update `numStrings` appropriately.

Fine points: If there is more than one copy of the string to be removed in the array, only the first one should be removed. Unused positions in the Strings array should contain null.

```
/** Remove first copy of str found in this StringList.
 *  Return true if str is found and removed, false otherwise.
 */
public boolean remove (String str) {
    boolean removed = false;
    int i = 0;

    while (i < numStrings && !removed) {
        if (strings[i].equals(str)) {
            removed = true;
        }
        i++;
    }

    if (removed) {
        while (i < numStrings) {
            strings[i - 1] = strings[i];
            i++;
        }
        numStrings--;
        strings[numstrings] = null;
    }
    return removed;
}
```

Question 2. (7 points) Assume we have the following sorting method in the class StringList. // method omitted

(a) (4 points) Trace the method and indicate the contents of the array `strings` at the end of each iteration of the `for` loop. Assume that the array `strings` contains the following elements before `mysterySort` is called:

```
strings = ["joe", "tammy", "hal", "carter"]
numStrings = 4
```

Use the following chart for the trace.

Variables	**Contents of `strings`**
`k = 3 2 1 0`	Iteration 1: `["joe", "tammy", "hal", "carter"]`
	Iteration 2: `["joe", "tammy", "carter", "hal"]`
	Iteration 3: `["joe", "carter", "hal", "tammy"]`
`str = "carter" "hal" "tammy" "joe"`	Iteration 4: `["carter", "hal", "joe", "tammy"]`
	(keep going if needed until mysterySort completes execution)
`position = 3 2 3 1 2 3 0 1 2`	

(b) (2 points) What is the invariant for the mysterySort sorting algorithm? You can either draw a picture or explain the invariant in English, or a combination of the two.

The Strings in positions k to numStrings – 1 are in sorted order.

(c) (1 point) Which sorting algorithm is mysterySort **most** similar to with respect to its operation? (Circle the correct answer.):

A. Bubble Sort

B. Insertion Sort

C. Selection Sort

Question 3. (4 points) Suppose the `strings` array in `StringList` is unsorted. There are N strings in the list (`i.e., numStrings`=N). What is the **average** number of required `String` comparisons to find an element in the StringList using **linear search**?

N/2 comparisons

Now suppose the `strings` array in `StringList` is sorted. Again, there are N strings in the list. What is the **maximum** number of required String comparisons to find an element in the StringList using **binary search**?

log2(N) or log(N)

Question 4. (12 points) Complete the following method for class `StringList`. Method `mergeList` takes as a parameter a `StringList sl` and merges the strings in `sl` with the current object's array of Strings. See the method header for pre- and post-conditions.

```
/** Merge strings in sl with current object's strings
 *  Precondition: sl.getStrings() are in sorted order and strings are in
 *  sorted order.
 *  Postcondition: strings contains all strings it originally had plus
 *  all strings in s1.getStrings(), and all strings are in sorted order.
 */
public void mergeList(StringList sl) {
    String [] newList = new String[numStrings + sl.getNumStrings()];
    String [] s1strings = s1.getStrings();
    int original = 0;
    int additional = 0;
    int combined = 0;
    // copy strings while there are uncopied strings in both lists
    while (original < numStrings && additional < sl.getNumStrings()) {
        if (strings[original].compareTo(slstrings[additional]) < 0) {
            newList[combined] = strings[original];
            original++;
        } else {
            newList[combined] = slstrings[additional];
            additional++;
        }
        combined++;
    }
    // one list has been completely copied here; copy what's left
    for (int i = additional; i < sl.getNumStrings(); i++) {
        newList[combined] = slstrings[i];
        combined++;
    }
    for (int i = original; i < numStrings; i++) {
        newList[combined] = strings[i];
        combined++;
    }
    strings = newList;
    numStrings = numStrings + s1.numStrings
}
```

(It is also correct to access the instance variables s1.numStrings and s1.strings directly.)

Question 5. (5 points) You are working for an online retail store that sells clothing. Your manager has asked you to modify the Customer class so that it assigns a new unique customer ID number when a customer object is created. The current implementation constructs a new Customer object using the ID number supplied as a parameter.

```
/** A class representing a Customer */
public class Customer {
   private static int nextIDNumber = 1;  //added

   // instance variables
   private int idNumber;               // customer ID number
   private String name;                // customer name
   private ShoppingCart items;         // customer's shopping cart of
                                       // to-be-purchased items
   private Address mailingAddress;     // customer's mailing address


   /** Construct a new customer object with idNumber, name, and
    * mailing address */
   public Customer(int idNumber, String name, Address mailingAddress) {
       this.idNumber = idNumber;  nextIDNumber;
       nextIDNumber++;           // added
       this.name = name;
       this.mailingAddress = mailingAddress;
       this.items = new ShoppingCart();
   }

   // other methods omitted to save space

}
```

Question 6. (11 points) The online retail store sells shirts, pants, and jackets. Here are the classes representing RetailItems and, on the next page, Shirts.

```
    /** return a String representation of the shirt */
    public String toString() {
        return "Shirt[price=" + getPrice() + ", manufacturer=" +
               getManufacturer() + ", size=" + size + "]";
    }
}
```

(a) (2 points) Complete the `toString` method, which should return a String that contains complete information about the state of a Shirt object, above.

(b) (5 points) For each of the following statements, answer the questions that appear indented under the statements. Assume that the statement(s) in each part is(are) written in a main method in a class called `Test` and are executed independently of other parts of the question.
(Grading note: if "Object" was not included in any of the dynamic types, this was treated as only one error.)

```
RetailItem r = new RetailItem(25.99, "Eddie Bauer");
```
What is/are the static type(s) of r? **RetailItem**
What is/are the dynamic type(s) of r? **RetailItem, Object**

```
Shirt s = new Shirt(45.50, "Land's End", "L");
```
What is/are the static type(s) of s? **Shirt**
What is/are the dynamic type(s) of s? **Shirt, RetailItem, Object**

```
RetailItem t = new Shirt(80.50, "Ralph Lauren", "M");
```
What is/are the static type(s) of t? **RetailItem**
What is/are the dynamic type(s) of t? **Shirt, RetailItem, Object**

```
RetailItem t = new Shirt(80.50, "Ralph Lauren", "M");
t.getSize();
```
Will the second statement produce a compiler error (Yes or No)? **Yes**
Why or why not? **Static type of t (RetailItem) does not have a getSize method defined.**

```
RetailItem t = new Shirt(80.50, "Ralph Lauren", "M");
t.toString();
```
Will the second statement produce a compiler error (Yes or No)? **No**
If yes, what is the error? If no, what String will be returned?
`Shirt[price=80.5, manufacturer=Ralph Lauren, size=M]`
(or whatever the toString method in the Shirt class returns)

(c) (4 points) Complete the sentences below with the best term/phrase from the following:

Subclass
Inherit
Override
Overload
Type
Parameter
Constructor

Question 7. (10 points) Use the `isPalindrome` method below to answer the following questions. Assume `isPalindrome` is defined in a class called `Word`.

```
public class Word {
  ...
  /** isPalindrome returns true if word is a palindrome ... */
  public static boolean isPalindrome(char[] word, int start, int end) {
    if (start == end) {       // looking at same character
                              // char[start] must be equal to char[end]
        return true;  // base case
    } else if ((start + 1 == end) && (word[start] == word[end])) {
        return true;  // base case
    } else if (word[start] != word[end]) {
        return false; // base case
    } else return isPalindrome(word, start+1, end-1); // recursive case
  }
  ...
}
```

(a) (2 points) Label the base case(s) and the recursive case(s) in the method definition above. Indicate the cases in the left margin next to the code above.

(b) (3 points) Suggest a set of test cases (typical cases, edge cases, "incorrect" cases) to test the `isPalindrome` method. For each test case, describe why you included it in your set of test cases.

Typical cases
- **Words with >2 characters that are/are not palindromes**
- **Words with >2 characters that have even/odd length**
- **(both of the above in all combinations)**

Edge cases
- **Words with 1 or 2 characters;**
- **2 character words where characters are same/different**

Incorrect cases
- **Start > end, start=1 instead of 0, end=word.length instead of word.length-1**

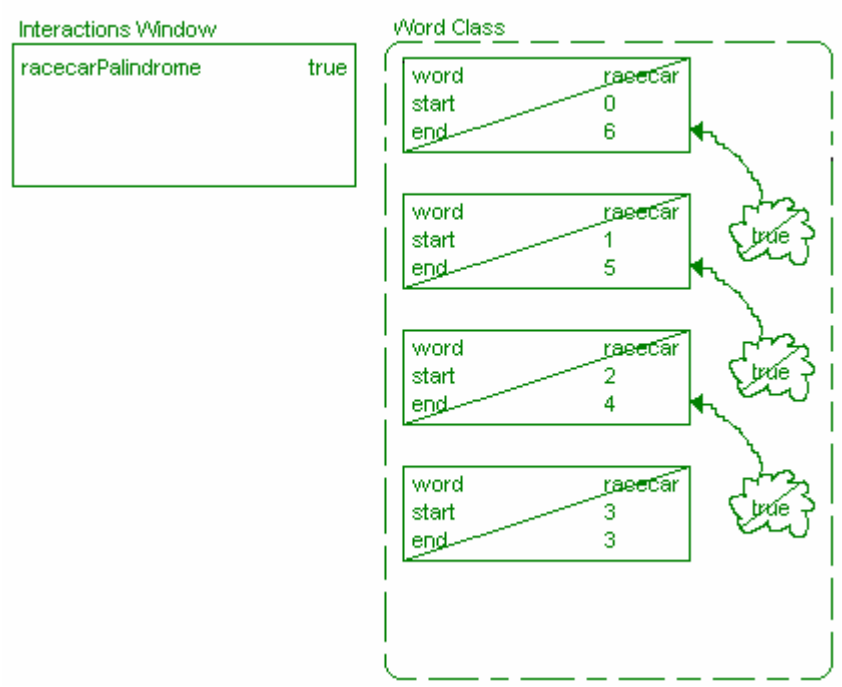
Actual solutions varied widely

(c) (4 points) Assume that the character array `racecar` has been initialized to the following in the Dr. Java interactions window:

```
char[] racecar = {'r','a','c','e','c','a','r'};
```

Draw the scoping diagram showing what happens during execution of the following method call in the Dr. Java interactions window:

```
boolean racecarPalindrome =
      Word.isPalindrome(racecar, 0, racecar.length-1);
```



(d) (1 point) What happens if the following is executed in the Dr. Java interactions window?

```
boolean racecarPal = Word.isPalindrome(racecar, 4, 2);
```

An index out of bound error occurs.