

CSE 143 Sample Midterm Exam #4

(based on Summer 2009's midterm; thanks to Alyssa Harding)

1. **ArrayList Mystery.** Consider the following method:

```
public static void mystery4(ArrayList<Integer> list) {
    for (int i = list.size() - 2; i > 0; i--) {
        int a = list.get(i);
        int b = list.get(i + 1);
        list.set(i, a + b);
    }
    System.out.println(list);
}
```

Write the output produced by the method when passed each of the following ArrayLists:

List

Output

(a)

[72, 20]

(b)

[1, 2, 3, 4, 5, 6]

(c)

[10, 20, 30, 40]

2. **ArrayList Programming.** Write a method `removeShorterStrings` that takes an `ArrayList` of strings as a parameter and that removes from each successive pair of values the shorter string in the pair. For example, suppose that an `ArrayList` called "list" contains the following values:

```
["four", "score", "and", "seven", "years", "ago"]
```

In the first pair of strings ("four" and "score") the shorter string is "four". In the second pair of strings ("and" and "seven") the shorter String is "and". In the third pair of strings ("years" and "ago") the shorter string is "ago". Therefore, the call:

```
removeShorterStrings(list);
```

should remove these shorter strings, leaving the list with the following sequence of values after the method finishes executing:

```
["score", "seven", "years"]
```

If there is a tie (both strings have the same length), your method should remove the first string in the pair. If there is an odd number of strings in the list, the final value should be kept in the list. For example, if the list contains the following values:

```
["to", "be", "or", "not", "to", "be", "hamlet"]
```

After calling `removeShorterStrings`, it should contain the following:

```
["be", "not", "be", "hamlet"]
```

You may not use any other arrays, lists, or other data structures to help you solve this problem, though you can create as many simple variables as you like. You may assume that the list passed is not `null`.

4. **Collections Programming.** Write a method `commonCustomers` that accepts two parameters, a `Map` from customer names (strings) to their television account numbers (integers) and a `Map` from customer names (strings) to their internet account numbers (integers), and returns a `Map` from customer names (strings) to their account number (integers) for each customer that has both a television account and a internet account. Assume that each customer with both types of accounts has the same account number for both accounts. For example, if a map `tvAccounts` contains these pairs:

```
{Marty=84321, David=23435, Stef=13266, Kim=62692, Lisa=25262}
```

and a map `internetAccounts` contains these pairs:

```
{Rob=93754, David=23435, Angela=42622, Kim=62692, Jason=36212}
```

The call of `commonCustomers(tvAccounts, internetAccounts)` should return a map containing these pairs:

```
{David=23435, Kim=62692}
```

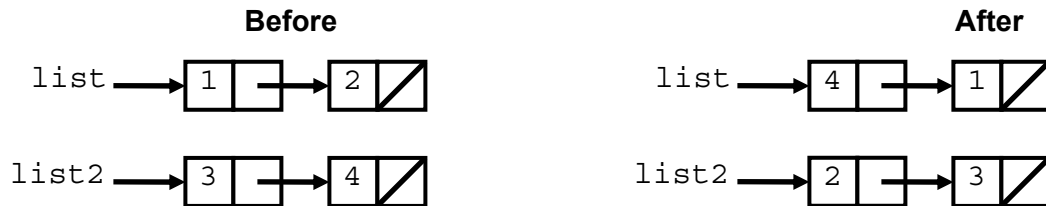
because David and Kim have both television accounts and internet accounts.

You may assume that the maps passed are not `null` and that no key or value in either is `null`. If either of the maps are empty or contain no common customers, your method should return an empty map.

You may create one collection of your choice as auxiliary storage to solve this problem. You can have as many simple variables as you like. You should not modify the contents of the maps passed to your method. For full credit your code must run in less than $O(n^2)$ time where n is the number of pairs in the maps.

5. **Linked Nodes.** Write the code that will turn the "before" picture into the "after" picture by modifying links between the nodes shown and/or creating new nodes as needed. There may be more than one way to write the code, but you are NOT allowed to change any existing node's `data` field value. You also should not create new `ListNode` objects unless necessary to add new values to the chain, but you may create a single `ListNode` variable to refer to any existing node if you like. If a variable does not appear in the "after" picture, it doesn't matter what value it has after the changes are made.

To help maximize partial credit in case you make mistakes, we suggest that you include optional comments with your code that describe the links you are trying to change, as shown in Section 7's solution code.



Assume that you are using the `ListNode` class as defined in lecture and section:

```
public class ListNode {
    public int data;           // data stored in this node
    public ListNode next;     // a link to the next node in the list

    public ListNode() { ... }
    public ListNode(int data) { ... }
    public ListNode(int data, ListNode next) { ... }
}
```

6. **Linked List Programming.** Write a method called `printPairsSwitched` that prints the elements of a list of integers so that the order of each pair of elements is switched. The method should separate each printed element with a single space. You are allowed to have a single extra space after the last element. Once all elements have been printed, the method should move to a new line. If the list contains an odd number of elements, the last element's order is unaffected. For example, if variables called `list1` and `list2` store the following values:

```
[1, 2, 3, 4]           // stored in list1
[5, 6, 7, 8, 9]       // stored in list2
```

then the calls:

```
list1.printPairsSwitched();
list2.printPairsSwitched();
```

should produce the following output:

```
2 1 4 3
6 5 8 7 9
```

Note that a call on the method should not change the structure of the list. That is, when the method is finished executing, the elements of the list should be in the exact same order as when it began.

Assume that we are adding this method to the `LinkedList` class as seen in lecture and as shown below. You may not call any other methods of the class to solve this problem and your method cannot change the contents of the list.

```
public class LinkedList {
    private ListNode front;

    methods
}
```

7. **Comparable Programming.** Suppose you have a pre-existing class `MovieRating` that keeps track of the ratings for a movie. The class has the following data and behavior:

Field/Constructor/Method	Description
<code>private String title</code>	title of the movie
<code>private int numRatings</code>	number of ratings for the movie
<code>private double cumulativeRating</code>	sum of all ratings for the movie
<code>public MovieRating(String title)</code>	constructs a <code>MovieRating</code> object with the given movie title. A newly constructed <code>MovieRating</code> contains no ratings.
<code>public String movieTitle()</code>	returns a <code>String</code> containing the title of the movie
<code>public String averageRating()</code>	returns a <code>String</code> containing the average rating for the movie. Returns "N/A" if the movie has no ratings
<code>public void addRating(double rating)</code>	adds the given rating
<code>public String toString()</code>	returns a <code>String</code> with the title of the movie and the average rating for the movie

Make `MovieRating` objects comparable to each other using the `Comparable` interface. Add any necessary code below, and/or make any changes to the existing code headings shown.

In general, movies with lower average ratings are considered "less" than other movies. Thus, movies with lower average ratings should appear at the beginning of a sorted list. However, movies that have not yet been rated are considered "less" than all movies that have been rated, regardless of the value of the rating. If two movies both have not been rated, they are considered "equal". Your method should not modify any account's state. You may assume the parameter passed is not null.

```
public class MovieRating {  
    ...  
  
    // write any added code here
```

```
}
```

8. Searching and Sorting.

(a) Suppose we are performing a **binary search** on a sorted array called `numbers` initialized as follows:

```
// index      0  1  2  3  4  5  6  7  8  9 10 11 12 13 14
int[] numbers = {0, 0, 5, 10, 15, 40, 55, 60, 65, 70, 80, 105, 110, 115, 200};

// search for the value 70
int index = binarySearch(numbers, 70);
```

Write the indexes of the elements that would be examined by the binary search (the `mid` values in our algorithm's code) and write the value that would be returned from the search. Assume that we are using the binary search algorithm shown in lecture and section.

- Indexes examined: _____
- Value Returned: _____

(b) Write the state of the elements of the array below after each of the first 3 passes of the outermost loop of the **selection sort** algorithm.

```
int[] numbers = {4, 444, 44, 33, 333, 3, 5, 555};
selectionSort(numbers);
```

(c) Trace the complete execution of the **merge sort** algorithm when called on the array below, similarly to the example trace of merge sort shown in the lecture slides. Show the sub-arrays that are created by the algorithm and show the merging of sub-arrays into larger sorted arrays.

```
int[] numbers = {4, 444, 44, 33, 333, 3, 5, 555};
mergeSort(numbers);
```


9. **Recursive Tracing.** For each call to the following method, indicate what value is returned:

```
public static int mystery(int n) {  
    if (n < 0) {  
        return -mystery(-n);  
    } else if (n == 0) {  
        return 0;  
    } else {  
        return mystery(n / 10) * 10 + 9 - (n % 10);  
    }  
}
```

Call	Value Returned
mystery(0)	
mystery(5)	
mystery(13)	
mystery(297)	
mystery(-3456)	

10. **Recursive Programming.** Write a recursive method called `digitsSorted` that takes an integer as a parameter and returns `true` if the digits of the integer are sorted and `false` otherwise. The digits must be sorted in non-decreasing order (i.e. increasing order with duplicate digits allowed) when read from left to right. An integer that consists of a single digit is sorted by definition. The method should be also able to handle negative numbers. Negative numbers are also considered sorted if their digits are in non-decreasing order. The following table shows several calls to your method and their expected return values

Call	Returns
<code>digitsSorted(0)</code>	<code>true</code>
<code>digitsSorted(2345)</code>	<code>true</code>
<code>digitsSorted(-2345)</code>	<code>true</code>
<code>digitsSorted(22334455)</code>	<code>true</code>
<code>digitsSorted(-5)</code>	<code>true</code>
<code>digitsSorted(4321)</code>	<code>false</code>
<code>digitsSorted(24378)</code>	<code>false</code>
<code>digitsSorted(21)</code>	<code>false</code>
<code>digitsSorted(-33331)</code>	<code>false</code>

You are not allowed to construct any structured objects other than strings (no array, `List`, `Scanner`, etc.) and you may not use any loops to solve this problem; you must use recursion.