Fall 2017 [Bono]

CS 455 Midterm 2 Solution

Note: on your graded exam you will only see the part of the rubric that was actually applied for your solution (to make it easier to read.) So here we are also giving you the broad outlines of the rubric used for each problem.

Problem 1

[6, 7, 12, 15, 5, 4, 9]

Problem 1 Rubric:
-1 number of passes is off by one
-1 missing or incorrect values in unsorted part of array
-3 wrong sorting method (or blank)

Problem 2 (several answers accepted)

Changing a key while it's in a Map would invalidate the Map. For instance, the Map would be unlikely to find the updated entry when you call the `get` or `containsKey` methods with the new key.

Problem 2 Rubric:
-1 part of the answer is correct, or answer needs a little bit more explanation
-2 vague, but may be on the right track
-3 incorrect (or blank)

Problem 3
Solution code in bold. There are several correct variations on how you might implement `compare`.

```
public static void sortByScore(ArrayList<Student> students) {

   Collections.sort(students, new ScoreComparator());

}

class ScoreComparator implements Comparator<Student> {

   public int compare(Student a, Student b) {

      int diff = b.getScore() - a.getScore();

      if (diff == 0) {
         return a.getName().compareTo(b.getName());
      }

      return diff;

   }
}
```

Problem 3 Rubric outline:

- *2 points total: the code in `sortByScore`*
- *1 point total: the headers for `ScoreComparator` and `compare`*
- *5 points total: the body of `compare` method*

Problem 4

Part A. The main alternate solution to this: the top index was the location of the last full element instead of the first empty element.

```
public class Stack {

  private int[] data;
```

```
   private int top;
   private static final int INITIAL_CAPACITY = 10;


   public Stack() {
      data = new int[INITIAL_CAPACITY];  // no points deducted for using a magic number here
      top = 0;
   }

   public void push(int val) {
      if (top == data.length) {   // array is already full -- need to grow it
         data = Arrays.copyOf(data, data.length * 2);
      }
      data[top] = val;
      top++;
   }

   public int pop() {
      top--;
      return data[top];
   }

   public int top() {
      return data[top - 1];
   }

   public boolean isEmpty() {
      return top == 0;
   }
}
```

Part B. representation invariant
The exact correct answer depends on the names of your instance variables and the representation used in your code. (the parts in parens were not required to get points)

- (when the stack is not empty) `data[top-1]` is the top element of the stack
- O <= `top` <= `data.length`
- (when the stack is not empty) the stack elements are in positions [0, `top`-1] of data
- (when the stack is empty `top` is 0)

Some common wrong answers and why they wouldn't be part of a rep. invar:

- *mentioning that the elements have to be type `int` [the compiler checks this -- no need to mention]*
- *mentioning the preconditions on `pop` and `top` [not an attribute of the representation used, it's part of the interface]*
- *mentioning how a stack works [also part of the interface]*

Problem 4 Rubric outline:

Part A (15). Points assigned for each method:

- *3 total: instance variable definitions and constructor*
- *5 push*
  - *3 general case*
  - *2 array is already full*
- *2 pop*
- *3 top*
- *2 isEmpty*

-3 solution takes O(n) for one or more methods.

Part B (4). representation invariant
-2 doesn't state exactly where top element is in array
-1 doesn't state the range of valid values for `top`
-1 doesn't state exactly where the stack values are in array

Problem 5

Part A.

- O(1) constructor
- O(1) numNames
- O(log n) lookup
- O(n) remove
- O(n) insert
- O(n) printNames

Part B. TreeSet

Part C.

- O(1) constructor
- O(1) numNames
- O(log n) lookup
- O(log n) remove
- O(log n) insert O(log n)
- O(n) printNames

Problem 5 Rubric:

Part A (5). one point for each method except constructor

Part B (4).

-1 HashSet [takes O(nlogn) for printNames]
-2 TreeMap [what is the value in an entry?] or unspecified Set [not a Java class]
-3 HashMap or unspecified Map
-4 other

Part C (5). one point for each method except numNames. This was scored in relation to what you wrote in part B. Even if you wrote an incorrect answer for Part B (e.g., ordered LinkedList), you could still receive all the points for Part C.

Problem 6
*There were several approaches. I'll give a few of the correct solutions here. Note: to meet the O(n) requirement we need a solution that constructs the new list using `list.add(elmt)` method (i.e., add an element to the **end** of the list we are creating.)*

Solution 1: 3-parameter solution (tail recursive)

```
public static ArrayList<Integer> oddVals(ArrayList<Integer> vals) {

   return oddValsR(vals, 0, new ArrayList<Integer>());

}


// oddValsR: Returns the list that has oddSoFar with all the odd values from the
// part of vals in positions [ start, vals.size() )  appended to it.
// This method modifies oddSoFar (will be the same list as the return value)

public static ArrayList<Integer> oddValsR(ArrayList<Integer> vals, int start, ArrayList<Integer> oddSoFar) {

   if (start == vals.size()) { // we've reached the end of the original list
      return oddSoFar;            // return the constructed odd list
   }

   if (vals.get(start) % 2 == 1) {  // it's an odd number
      oddSoFar.add(vals.get(start));   // O(1) to add to the end of an AL
   }
   // else (even number) do not add anything to oddSoFar

   return oddValsR(vals, start + 1, soFar);

}
```

Solution 2: *2-parameter solution that traverses right-to-left (this non tail-recursive solution needs to traverse R to L, to not reverse the order of the elements, and to meet the O(n) requirement). It builds the list on the way back from the recursive calls.*

```
public static ArrayList<Integer> oddVals(ArrayList<Integer> vals) {

   return oddValsR(vals, vals.size() - 1);

}


// oddValsR: Returns a list that has all the odd values from the part of vals in positions [ 0, end )

public static ArrayList<Integer> oddValsR(ArrayList<Integer> vals, int end) {

   if (end < 0) {   // finished traversing the list
      return new ArrayList<Integer>();
   }

   ArrayList<Integer> firstPart = oddValsR(vals, end - 1);

   if (vals.get(end) % 2 == 1) {  // it's an odd number
      firstPart.add(vals.get(start));   // O(1) to add to the end of an AL
   }
   // else (even number) do not add anything to the new list

   return firstPart;

}
```

Problem 6 Rubric Outline:

- *1 point for the* `oddvals` *method (i.e., correct call to helper and correct return statement)*
- *14 for the helper method:*
  - *2: has a test for odd/even value and attempts to add to or remove from AL based on that. (can't get any other points unless you have this) [these are the only points available if your solution does not use recursion]*
  - *4: base case for recursion (2 for correct test, 2 for correct return value)*
  - *8: recursive cases. (Because of all the variations on approaches, there were many ways partial credit was awarded within these 8 points.)*
- *-2 O(n^2) solution (for example, repeatedly inserts into, or removes from, the **front** of the AL)*
