THIRD QUIZ

You have 20 minutes from the start of class to complete this quiz. Give partial answers if you can’t give complete ones. Read the questions with care; work with deliberate speed. Don’t give us more than we ask for. The usual instructions apply. Good luck!

Problem 1 (12 points)

Attached is a version of the restaurant collection program that uses an ArrayList. It has one new feature: The command ‘h’ prints out the name of the restaurant in the collection with the highest-priced dish.

(a) (8 points) In the RList class, the definition for `getHighestPricedRrant` is incomplete. Fill it in, using the space on the program listing.

Answer to part (a); of course there are many equivalent possibilities:

```java
double highestPriceSoFar = 0;
Rrant highestPricedRrantSoFar = null;
for (Rrant r : theRrants) {
    if (r.getPrice() > highestPriceSoFar) {
        highestPriceSoFar = r.getPrice();
        highestPricedRrantSoFar = r;
    }
}
return highestPricedRrantSoFar;
```

2 points for loop over collection (1 for attempt, 1 for correct (any form))
1 point for checking current Rrant against highest price so far (1/2 for attempt/any var; 1/2 for correct)
2 points for keeping track of the highest price and corresponding Rrant so far (1 for attempt/at least one assignment; 1 for correct.)
1 point for correctly initializing highest price and corresponding Rrant (to whatever works in their code)
1 point for correctly returning the correct restaurant
1 point for handling the empty restaurant case and everything else correctly.

(b) (4 points) In the main Q3RPArrayList class, we have modified `printMenu` to list the new command. We have left two blank lines in `handleCommands` to handle printing the name of the highest-priced restaurant. Fill in those lines.

```java
else if (response.equals("h"))
    System.out.println(C.getHighestPricedRrant().getName());
```

1 point for correct check of "h" command
1 point for correct printing and a call to `getHighestPricedRrant`
1 point for correctly calling `getHPR` on the Collection `C`.
1 point for recognizing that you need to print just the NAME of the Rrant (and doing it right)
Problem 2 (13 points)

Your local radio station wants to computerize its collection of CDs. They have asked you consider these three alternative data structures:

I. A conventional linked list, in no particular order
II. An array ordered by CD title, with an additional field that stores the number of CDs in the collection
III. A binary search tree, ordered by CD title.

When we ask for O-notations below, give the closest-fit O-notation in terms of \( n \), the number of CDs \( n \) in the collection, assuming each operation is coded as efficiently as possible in Java.

(a) (4 points) Suppose the first task is to add all the CD data into the new system. Which of the three data structures would be most efficient for this task alone? As part of your answer, give each alternative’s O-notation for adding a CD to the collection.

Some will read this as asking about one addition instead of all; that’s okay if their answers are consistent. List is O(n); array is O(n log n)—create array, add items, sort—but give credit if they say \( n^2 \) for adding them in linearly; tree is O(n log n). List wins.

1 point for each, plus 1 point for conclusion.

(b) (4 points) A DJ spends most of his or her time looking up CDs by title. What is each data structure’s O-notation for locating a CD by title? Which data structure do you most want to avoid for this task?

Searching list is O(n); searching array is O(log n) using binary search; searching the tree is O(log n); avoid List.

1 point for each, plus 1 point for conclusion (that follows from O-notation)

(c) (4 points) The station manager wants to know at any given moment exactly how many CDs are in the collection. Give each alternative’s O-notation for determining the number of CDs and say which alternative is most efficient for this operation. Don’t assume the existence of any data fields not specified above.

List is O(n) because there’s no size field; array is O(1) ‘cause it has size; BST is O(n). Array wins.

1 point for each, plus 1 point for conclusion

(d) (1 point) What would be the best data structure to use for implementing this collection, and (in one brief sentence including O-notations where appropriate) why? Assume that task (a) will be done just once and that most of the collection’s usage will be split evenly between tasks (b) and (c). You may propose small modifications to the data structures described above if they would help produce a clear winner.

A variety of answers is possible, depending on how they modify the collection. What I was thinking of was adding a size field to each, which would make the array and the tree equivalent by O-notation on the two critical operations. It’s okay if they just say they’re tied; it’s also okay if they break the tie either (a) by saying the tree is better for the one-time task of filling the collection or (b) by saying the array is better because it requires less storage per CD or (c) by saying the tree is better because it dynamically allocates just enough storage for the current size of the collection. Basically, they get the points if what they say is consistent and correct, and they don’t if it’s not.