

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 (10 points)

(a.1) (5 points) What is the polynomial representing the execution time of the following code in terms of n ? Count assignments (except in for-loops) and function calls. Showing your work will help you get partial credit.

```
System.out.println("Did I ever tell you that Mrs. McCave");
System.out.println("Had twenty-three sons, and she named them all Dave?");
for (i = 0; i < n; i++)
{
    doSomethingFun(i);
    System.out.println("Well, she did, and it wasn't a smart thing to do.");
    for ( j = 1; j <= n; j = j * 2 )
    {
        doSomethingHard(j);
        doSomethingEasy(i*j);
    }
    System.out.println("You see, when she wants one and calls out, 'Yoo-Hoo!");
    System.out.println("Come into the house, Dave!' she doesn't get one.");
}
System.out.println("All twenty-three Daves of hers come on the run!");
```

(a.2) (2 points) What is the O -notation of the execution time of the code in part (a.1)?

(b) (3 points) Give the O -notation of each of the following polynomials. Read them carefully.

(b.1) $24n^3 + 15n^2 + 1355n$

(b.2) $7 \log n + 2n \log n + 18n + 180$

(b.3) $10 \log n + 20n + 30$

Problem 2 (15 points)

Your local bakery wants to computerize its customer list. You have decided to consider these three alternative data structures; each structure described below includes an additional field that stores the number of customers.

- I. A conventional linked list, in no particular order
- II. A (well-balanced) binary search tree, ordered by customer name
- III. A hash table, using the customer name to compute the hash code (and assuming that the table size, T , is adequate and the hash function effectively distributes the customers throughout the table)

When we ask for O -notations below, give the closest-fit O -notation in terms of n , the number of customers, and T where appropriate. Also assume each operation is coded as efficiently as possible in Java. If we ask for the most efficient alternative and there's a tie for the lowest O -notation, just say so.

(a) (4 points) Suppose the first task is to add *all* the customer data into the new system. In terms of execution time, which one 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 *one* customer to the collection.)

(b) (4 points) The bakery clerk spends a busy hour in the morning taking phone orders; each order requires looking up a customer by name. What is each data structure's O -notation for finding a customer by name? Which data structure is most efficient (in terms of execution time) for this task?

(c) (4 points) Each customer's record also includes any current orders (but the collection includes all regular customers, even if they don't have an order pending). The baker wants to know at any given moment how many orders need to be filled. Give each alternative's O -notation for counting current orders and say which alternative is most efficient for this operation.

(d) (3 points) What would you recommend as 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.

If we had more time, we might have given problems like these two. We provide them here just for practice; they're not required.

Extra Problem 1

Suppose you decide to store a large collection of data items, each with a “key” value you can use to order the items or look them up. (In class, we called this data structure a table.) You consider four different data structures for your implementation:

Structure I – An array, ordered by the key, with an additional single integer to store the number of items in the table. No gaps are left in the array when items are removed.

Structure II – A dynamically allocated doubly linked list (where each element includes one pointer to the next item and one pointer to the previous item), ordered by the key, with an additional single integer to store the number of items in the table.

Structure III – A reasonably balanced binary search tree, ordered by the key.

Structure IV – A hash table with an ideally random hash function, in which collisions are resolved by linear chaining (i.e., by a singly linked list of items with the same hash value).

(a.1) (10 points) Complete the following table, giving the best O-notation for each operation on each data structure, assuming that each operation is implemented in Java using the most efficient algorithm available (without changing the structure described). Assume that there are i items in the collection, that the array's capacity is a elements, and that the hash table's size is t . (You might look at part (a.2) before you start; it asks you to think about data movements vs. comparisons.)

Operations:	Structure I	Structure II	Structure III	Structure IV
Add a new item				
Search for an item, given its key				
Search for an item by some characteristic <i>other than the key</i>				
Print all the items (in any order)				
Print all the items (in order by owner's name)				

(a.2) (1 point) In the table above, some of the entries measure primarily comparisons and others mainly measure data movements. Place an asterisk (“*”) next to each entry whose O-notation measures primarily data movements.

(b.1) (2 points) In the table below, give the best O-notation for the storage required by each structure, using the variables listed in part (a.1) above.

	Structure I	Structure II	Structure III	Structure IV
Storage required				

(b.2) (1 point) Suppose that you want to print the items in the structure in order according to the key. If any of the structures would require more than a constant amount of extra storage, mark its entry in this table with an asterisk (“*”).

(c) Now suppose that you work for the Home Shopping Network and you intend to use this table in a system for entering information provided by people who call your 800 number after seeing one of your shows. You may have hundreds of callers at the same time, and tens of thousands in the table. Your key will be the customer’s full name; for simplicity, we’ll ignore the possibility that two customers could have the same name (though there are ways to deal with this).

(c.1) (2 points) Which one of the four data structures (I through IV) would most efficiently meet the needs of this application (i.e., rapid entry of new customers and location of existing customers)?

(c.2) (2 points) Suppose that the manager wants to see an up-to-date ordered list of customers in real time (that is, immediately after the request is made). Which one of the four data structures (I through IV) most efficiently meets this constraint while still providing reasonably rapid data entry?

(c.3) (2 points) Finally, suppose that a caller gives her name as Grace Murray Hopper. When the operator enters that name, the system doesn’t find it. But in the table there is an entry for Grace M. Hopper that refers to the same person. Suppose you would like to add this additional capability to your system: When a name is not found, the operator can “browse” the table, looking at a few names before and after the place where the caller’s name would go (in order), just in case one of those names is a variation on the caller’s name. Two of the four data structures would make coding this feature relatively easy; the other two would make it significantly more complex. Which two data structures (I through IV) make coding this feature easier?

Extra Problem 2

In the ArrayList version of the Java Restaurants Program (a copy of which we would have provided if this were a real quiz question), add the following new method to the RList class:

```
public double averagePriceByCuisine (String c) {  
    // Return the average price of all restaurants in the collection  
    // that serve the specified cuisine.
```

```
}
```

Harder problem to think about: Create a list containing the lowest-priced restaurant for each different cuisine.