Big O/Code Analysis Review Session

19su Week 3

Part 1: Big O Quick Maths

For each of the runtimes below, write the **simplified** tight big O bound. You are not required to show work.

1. 2π
2. $n\ log\ n\ + log\ n + \ log\ log\ n$
3. $3^n + 7^n + 3^n$
4. $log_3 n\ +\ log_7 n\ + log_3 n$
5. $n\ log\ (n^2)\ +\ n\ log\ (n)$

Part 2: Big O & Its Bounded Friends

For each of the runtimes below, circle **all** items that are true for that runtime. You are not required to show work.

1. $5n^2 + 6n + log(n)$ is:

   $\Omega(n^3)$ $\Theta(n^3)$ $O(n^3)$

2. $log_2(n)\ + log_4(n)$ is:

   $\Omega(log_4(n))$ $\Theta(log_4(n))$ $O(log_4(n))$

3. $3^n + 2^n$ is:

   $\Omega(3^n)$ $\Theta(3^n)$ $O(3^n)$

Part 3: Scenarios for Consideration

For each of the given scenarios select an ADT and data structure you would use for each scenario. Justify your answers *briefly* for full credit. Try to use each ADT only once.

ADT: List, Dictionary (Map), Queue, Stack
Data Structure: Array, Linked List

1. You are responsible for developing software that plays arrival announcements for Link light rail trains. Before the train arrives at each station, it needs to obtain the filename containing the audio of the station that's coming up next. The system must only play the station announcements in order of station sequence, from UW to Angle Lake and back. You already know that there are 16 stations. You are wondering how the filenames should be stored.

2. You are developing a new group picker for CSE 373 projects, in which a student picks one project partner. As the TAs iterate through the list of students, it should be easy for them to retrieve each student's project partner. Since there will be frequent adds and drops to the course, you should make sure that the tool is still reasonably fast even when there are a lot of changes to the student database. You are wondering how the students will be stored internally.

3. Fed up with jGRASP, you decided to implement a brand new code editor that is more friendly for CSE 14x students. An important feature for any code editor is the ability to revert the latest edit(s) using Ctrl+Z, and you are trying to create a history of all past changes to code for reverting later. Remember that when reverting, the latest edit should be undid first. At this time, you're unsure how many edits a user will likely make per working session.

4. You are developing a mobile order app for Lee Paul Sieg's Gyro & Falafel. Since your menu items have long names ("#15, Grilled beef salad with lettuce, tomatoes, onions, lemons and tahini"), you want to provide the ability for the users to key in their dishes using a short index number (15). You want to optimize for fast lookup of menu items, even when different indexes are used in no particular order.

Part 4: Code Snippets (Significantly Borrowed from CSE 332 18su)

Describe the worst case running time of the following pseudocode functions in Big-O notation in terms of the variable n. Your answer should be as “tight” and “simple” as possible.

```
public void question1 (int n) {
    for (int i = n; i > 0; i = i / 2) {
       System.out.println(“it all keeps adding up”);
    }
}
```

Runtime: ____________

```
public void question2 (int n) {
    for (int i = 0; i < n * n; i++) {
        for (int j = 100; j < n * n * n; j++) {
           System.out.println(“i think i’m cracking up”);
           break;
        }
    }
}
```

Runtime: ____________

```
public int question3 (int n) {
    int tuition = 0;
    if (n < 20) {
        for (int i = 0; i < n * n * n; i++) {
            tuition++;
        }
    } else {
        for (int i = 0; i < n * n; i++) {
            tuition--;
        }
    }
    return n / 2;
}
```

Runtime: ____________

```
public int question4 (int n) {
    if (n < 100) {
        return 1;
    } else {
        return question4(n - 1) * question4(n - 1);
    }
}
```

Runtime: ____________

Part 5: Proofs

By finding a c and an n_o, use the definition of Big O to write a proof to show that the following is $O(n^2)$:

$$3n^2 + \log n - 100$$

Part 6: Case Analysis

```
public static IDictionary<String, Integer> countStrings(IList<String> input) {
    IDictionary<String, Integer> dict = new ArrayDictionary<String, Integer>();
    for (String curString : input) {
        if (!dict.containsKey(curString)) {
            dict.put(curString, 1);
        } else {
            dict.put(curString, dict.get(curString) + 1));
        }
    }

    // for (KVPair<String, Integer> pair : dict) {
    //     System.out.println(pair.getValue()); // pair.getValue() always takes O(1)
    // }

    // the following code works exactly the same as the for-each loop above
    Iterator<KVPair<String, Integer>> itr = dict.iterator();
    while (itr.hasNext()) {
        KVPair<String, Integer> next = itr.next();
        System.out.println(next.getValue());
    }
    return dict;
}
```

Assume that all the of the following methods always run in O(1) time:

- dict.iterator();
- itr.hasNext();
- itr.next();
- next.getValue();

Describe a valid example input that will trigger the best case (fastest) asymptotic runtime for the 2nd loop (over `dict`). What is the runtime for this loop in the best case?

Describe a valid example input that will trigger the worst case (slowest) asymptotic runtime for the 2nd loop (over `dict`). What is the runtime for this loop in the worst case?