CSE 373 18wi: Practice Midterm

Name:

UW email address:

Instructions

- **Do not start the exam until told to do so**.
- You have **80** minutes to complete the exam.
- This exam is closed book and closed notes.
- You may not use a cell phone, a calculator, or any other electronic devices.
- Write your answers neatly in the provided space. Be sure to leave some room in the margins: we will be scanning your answers.
- If you need extra space, use the back of the page.
- If you have a question, raise your hand to ask the course staff for clarification.

Question	**Max points**	**Earned**
Question 1	??	
Question 2	??	
Question 3	??	
Question 4	??	
Question 5	??	
Question 6	??	
Question 7	??	
Question 8	??	
Question 9	??	
Total	**??**	

Additional notes about this practice exam

- This practice midterm is structured in roughly the same way the actual midterm is.
- These questions are either roughly about the same difficulty or are slightly harder then the questions you will be asked on your midterm.
- Since this is a new practice exam, our solutions may contain a few mistakes or typos. Please ask on Piazza if you see something that does not make sense.

1. AVL rotations

Insert the following sequence of values into an AVL tree *in the given order*.

a, d, h, m, p, a, g, n, t, m, o, z

2. Hash tables

Consider the following sequence of key-value pairs:

(3.4, a), (7.1, b), (9.6, c), (3.5, d), (8.4, e), (3.4, f), (3.1, g), (2.7, h)

Suppose that these floats are implemented such that they use the hash function "$h(k)$ = roundDown(k)". For example, the key 3.4 would have a hash code of 3; the key 9.9 would have a hash code of 9.

(a) Insert the above sequence of key-value pairs *in the given order* into a hash table with an internal array of size 5 using *separate chaining*. Assume each bucket is implemented using an *binary search tree*. Do not worry about resizing the internal array.

(b) Insert the same pairs into an a hash table with capacity 10 that uses *quadratic probing*. Again, do not worry about resizing the internal array.

3. Asymptotic analysis

(a) Consider the following two functions:

$$f(n) = \frac{1}{10}n^2 \qquad\qquad g(n) = \begin{cases} n^5 & \text{if } n \leq 5 \\ 99n + \log(n) & \text{otherwise} \end{cases}$$

Show that $f(n) \in \Omega\left(g(n)\right)$ is true by finding a c and n_0 that satisfies the definition of "dominates" and big-Ω. Please show your work.

(b) Suppose we discovered a function $h(n)$ and discovered that the tightest possible upper bound is $h(n) \in \mathcal{O}\left(n\right)$ and the tightest possible lower bound is $h(n) \in \Omega\left(\log(n)\right)$. Draw a plot of what this function might look like.

4. Eyeballing Big-Θ bounds

For each of the following snippets of code, please give a big-Θ bound of the worst-case runtime with respect to n. You do not need to justify your answer.

(a)

```
public void partA(int n) {
    for (int i = 0; i < n * n; i++) {
        if (i % 2 == 0) {
            for (int j = 0; j < i; j++) {
                System.out.println("?");
            }
        }
    }
}
```

Your answer here:

(b)

```
public void partB(int n) {
    // An AvlDictionary is a dictionary internally implemented
    // using an AVL tree.
    IDictionary<Integer, Integer> dict = new AvlDictionary<>();
    for (int i = 0; i < n; i++) {
        if (i < 100000) {
            for (int j = 0; j < i; j++) {
                dict.put(j * i, i);
            }
        } else {
            dict.put(i, i);
        }
    }
}
```

Your answer here:

(c)

```
public void partC(int n) {
    IDictionary<Integer, Integer> dict = new AvlDictionary<>();
    for (int i = 0; i < n; i++) {
        dict.put(4, i);
        for (int j = 0; j < dict.size(); j++) {
            System.out.println(dict.containsKey(j));
        }
    }
}
```

Your answer here:

(d)

```
public void partD(int n) {
    if (n == 0) {
        System.out.println("...");
    } else {
        System.out.println("...");
        partD(n - 1)
    }
}
```

Your answer here:

5. Modeling code

Consider the following Java program. Let n represent the value of the input parameter `n` and let m represent the value of the parameter `m`.

```java
public static int mystery(int n, int m) {
    if (n >= 40) {
        for (int i = 0; i < n * m; i++) {
            if (i % m == 0) {
                System.out.println("...");
            }
        }
        return -2 * mystery(n - 3, m / 3) + 3 * mystery(n - 5, m + 3);
    } else {
        return m * 2;
    }
}
```

In the following questions, you will be asked to construct several mathematical functions modeling different aspects of the `mystery` method. Your answers to all three questions should be a recurrence. Your recurrence may include summations, if you want. You do *NOT* need to find a closed form to your models.

(a) Construct a mathematical function $T(n, m)$ that represents the *approximate worst-case runtime* of `mystery`.

(b) Construct a mathematical function $P(n, m)$ that represents the *total number of lines printed out* by `mystery`.

(c) Construct a mathematical function $F(n, m)$ that represents the *exact integer output* of `mystery`. That is, it should be the case that $F(n, m)$ == `mystery(n, m)`.

6. Systems and B-Trees

(a) Consider the following code:

```
public static int sum(IList<Integer> list) {
    int output = 0;
    for (int i = 0; i < 128; i++) {
        // Reminder: foreach loops in Java use the iterator behind-the-scenes
        for (int item : list) {
            output += item;
        }
    }
    return output;
}
```

You try running this method twice: the first time, you pass in an array list, and the second time you pass in a linked list. Both lists are of the same length and contain the exact same values.

You discover that calling sum on the array list is consistently 4 to 5 times faster then calling it on the linked list. Why do you suppose that is?

(b) Suppose you are trying to use a B-Tree somebody else wrote for your system. You know the following facts:

- $M = 10$ and $L = 12$
- The size of each pointer is 16 bytes
- The size of each key is 14 bytes
- The size of each value is 11 bytes

Assuming M and L were chosen wisely, what is most likely the page size on this system?

7. Short answer

This section has questions that require very short answers. For full credit, write at most two to three sentences per each question.

(a) Suppose you were trying to implement edit/undo functionality in an image editing program. We want to implement this by keeping track of each operation the user makes. Which ADT would be the most appropriate way of storing these operations: a stack, a queue, or a list? Pick one, and briefly justify.

(b) What is the worst-case runtime to append a value to the end of a singly-linked list?

(c) True or false: If $f(n) \in \mathcal{O}\left(g(n)\right)$ is true, then $g(n) \in \mathcal{O}\left(f(n)\right)$ is also always true. Briefly justify.

(d) True or false: "$f(n) \in \mathcal{O}\left(n^3\right)$" means the exact same thing as "$f(n)$ has a worst-case runtime of n^3". Briefly justify.

(e) Suppose you need a dictionary where you can traverse over the keys in sorted order. Which data structure should you use?

(f) True or false: an AVL tree is always more asymptotically efficient then a BST. Briefly justify.

(g) Suppose you want to implement an efficient dictionary where the iterator always returns key-value pairs in the order the client added them to the dictionary. You know the client will never remove any key-value pairs or update any previously-added pairs. Briefly describe how you would implement this dictionary by combining two data structures we studied in class. The `put(...)` method should still have an average runtime of $\Theta(1)$.

(h) True or false: A hash table's `get(...)` method will always have a runtime of $\Theta(1)$. Briefly justify.

(i) Which is faster: printing a list of numbers stored in an in-memory array or stored in a text file? Briefly justify.

8. Debugging

In this problem, we will consider an algorithm named `isBalanced(String str)` that returns "true" if the input string has a "balanced" number of parenthesis and false otherwise. We say a string has "balanced" parenthesis if each opening paren is paired with a matching closing one.

For example, this string is balanced: "((a)b)". This string is also balanced: "(x)(y)(z)".

However, the following two strings are **not** balanced: "((((" and ")))z(".

(a) List at least four distinct kinds of inputs you would try passing into the `isBalanced` algorithm to test it. For each input, also list the expected outcome (assuming the algorithm was implemented correctly). Be sure to think about different edge cases.

(b) Here is one (buggy) implementation of this algorithm in Java. List every bug you can find.

```java
boolean isBalanced(String str) {
    if (str == null || str.size() == 0) {
        return false;
    }
    int numUnmatchedOpenParens = 0;
    for (char c : str) {
        if (c == '(') {
            // Handle opening parens
            numUnmatchedOpenParens += 1;
        } else {
            // Handle closing parens
            numUnmatchedOpenParens -= 1;
        }
    }
    return numUnmatchedOpenParens == 0;
}
```

9. Design

In this problem, you will implement an algorithm named `containsString(String searchTerm, String document)` that returns 'true' if the document contains the search term, and false otherwise.

A naive way of implementing this is to use two nested loops that check if `searchTerm` is equal to a substring of `document`. However, this is inefficient if `searchTerm` is large: comparing two strings of length n takes $\mathcal{O}(n)$ time since we need to check each char one by one.

Your goal is design a faster algorithm by using a kind of hashing algorithm known as a "rolling hash". A rolling hash implements the following methods:

```java
public class RollingHash {
    // Instructs the object to keep track of the last 'k' characters eaten.
    public RollingHash(int k) { ... }

    // The number of characters remembered. Returns a number between 0 to k.
    public int size() { ... }

    // Adds the given char to the internal state. If size() > k, forgets the oldest char eaten.
    public void eat(char c) { ... }

    // Returns the hash of the last k characters eaten
    public int getHash() { ... }
}
```

Amazingly, every method in this class has a worst-case runtime of $\mathcal{O}(1)$!

(a) List at least four distinct kinds of inputs you would try passing into your `containsString` algorithm to test it. For each input, also list the expected outcome (assuming the algorithm was implemented correctly). Be sure to think about different edge cases.

(b) Write an **English description** or **high-level pseudocode** describing an algorithm that implements `containsString`. Note: do **NOT** write Java code.

You may assume `RollingHash` is already implemented for you.

(c) Provide a tight big-Θ bound of the worst-case runtime of your algorithm. Write your answer in terms of s and d, where s is the length of the `searchTerm` string and d is the length of the `document` string.

Briefly justify your answer

This page is intentionally left empty

Feel free to use this page for scratch paper.