CSE 373: 24 Wi Midterm

Name:	UW Email:	@uw.edu

Instructions

- The allotted time is **50** minutes. Please do not turn the page until the staff says to do so.
- This is an open-book and open-notes exam. You are NOT permitted to access electronic devices including calculators.
- Read the directions carefully, especially for problems that require you to show work or provide an explanation.
- We can only give partial credit for work that you've written down.
- Unless otherwise noted, when we ask for algorithm runtime, it must be **simplified and tight**.
- **You may assume that all hash functions and operations (find, add, remove) are O(1).**
- If you run out of room on a page, indicate where the answer continues. Try to avoid writing on the very edges of the pages: we scan your exams and edges often get cropped off.

Advice

- If you feel like you're stuck on a problem, you may want to skip it and come back at the end if you have time.
- Relax and take a few deep breaths. You got this :-)

Questions
1. ADT Design
2. Code Analysis

Resubmission Details

- This exam will be graded out of 100 points. If you are not satisfied with your grade, you will be given the opportunity to resubmit it online and earn up to 50% of the missed points back.
- For example, a student scoring 80/100 points may receive up to 90/100 points on the resubmission.

Problem 1 (ADT Design):

Kasey and her TAs are starting up a new grocery store business called Trader O's. They need to keep track of the prices of products in their store. **Each product has a unique UPC code and a unique price**. They want to write a program to get the price of a product by scanning its UPC code.

Their program will need the following functionality:

- **void addProduct(String UPCCode, double price)**
  - adds a product to the database with the given `UPCCode` and `price`
  - if the `UPCCode` already exists in the database, only update the `price`
- **double find(String UPCCode)**
  - returns the price of the product with the given `UPCCode`
  - returns `-1` if the `UPCCode` is not found

Azita, Eesha, and Simon each came up with solutions. They all use **the fastest possible algorithms** with their choices of data structures.

Simon's Solution: We can store (`UPCCode`, `price`) pairs in a sorted ArrayList (the array will be sorted in ascending order by UPC code).

1) What is the <u>worst-case</u>, <u>simplified</u>, <u>tight big-oh</u> runtime of **addProduct**?

2) What is the <u>worst-case</u>, <u>simplified</u>, <u>tight big-oh</u> runtime of **find**?

Eesha's Solution: We can store (UPC, price) nodes in an AVL Tree.

3) What is the <u>worst-case</u>, <u>simplified</u>, <u>tight big-oh</u> runtime of **addProduct**?

4) What is the <u>worst-case</u>, <u>simplified</u>, <u>tight big-oh</u> runtime of **find**?

Azita's Solution: We can store (UPC, price) pairs in a HashMap.

5) What is the <u>in-practice</u>, <u>simplified</u>, <u>tight big-oh</u> runtime of **addProduct**?

6) What is the <u>in-practice</u>, <u>simplified</u>, <u>tight big-oh</u> runtime of **find**?

Some of Kasey's students heard about the success of Trader O's and are planning on getting their groceries there. However, since they are broke college students, they want to get the most bang for their buck when buying Trader O's groceries. To help out her students, Kasey decides to add the following method to her ADT:

- **List<String> getPurchasableProducts(double money)**
  - returns the UPC codes of all products in the store with price ≤ **money**

Kasey wants her program to optimize for the `getPurchasableProduct` operations. However, she still wants to maintain a relatively quick runtime for `addProduct` and `find`. You are asked to design your own implementation of this ADT:

7) Describe what data structures you'll use and what they represent (≤ 3 sentences).

8) Describe your implementation of `getPurchasableProducts` (≤ 3 sentences).

9) Give the <u>worst-case</u>, <u>simplified</u>, <u>tight big-oh</u> runtime of your solution to `getPurchasableProducts`. Use in-practice runtime for any HashMaps, no explanation necessary.

Problem 2 (Code Analysis):

Yafqa wants his students to participate more in section, so he thinks to himself, "it would be nice if I had a random name generator to randomly call on students". He wants to design an ADT called "RandomizedSet" with the following functionalities:

- **boolean insert(String name)**
  - inserts a name to the set
  - returns `true` if the size of the set has increased, `false` otherwise
- **boolean remove(String name)**
  - removes a name from the set
  - returns `true` if the set contained the name, `false` otherwise
- **String pick()**
  - randomly chooses a name from the set and returns it
  - returns `null` if the set is empty
  - each name should have an <u>equal chance of being picked</u>
- **int size()**
  - returns the number of elements in the set

Here's an example of how Yafqa might use such a program:

```
List<String> students = List.of("Simon", "Josh", "Sravani", "Emily", "Azita");
RandomizedSet picker = new RandomizedSet();
for (String student : students) {
   picker.insert(student);
}

List<String> shuffledStudents = new ArrayList<>();
while (picker.size() != 0) {
   String nextStudent = picker.pick();
   shuffledStudents.add(nextStudent);
   picker.remove(nextStudent);
}

System.out.println(shuffledStudents);

// output:
// [Sravani, Azita, Simon, Emily, Josh]
```

Here is Yafqa's first implementation of this ADT.

```
public class RandomizedSet {
    private List<String> names;
    private Random r; // assume r.nextInt(...) is O(1).

    public RandomizedSet() {
        names = new LinkedList<>(); // assume this is singly-linked
        r = new Random();
    }

    public boolean insert(String name) {
        if (names.contains(name)) {
            return false;
        }

        names.add(0, name);
        return true;
    }

    public boolean remove(String name) {
        Iterator<String> itr = names.iterator();
        while (itr.hasNext()) {
            if (itr.next().equals(name)) {
                itr.remove();
                return true;
            }
        }
        return false;
    }

    public String pick() {
        if (size() == 0) {
            return null;
        }

        int index = r.nextInt(size());
        return names.get(index);
    }

    public int size() {
        return names.size();
    }
}
```

Give the **worst-case, simplified, tight big-oh runtime** of the above implementation for the following methods

1) insert:

2) remove:

3) pick:

Here is Yafqa's second implementation of this ADT.

```
public class RandomizedSet {
    private Set<String> values;
    private Random r; // assume r.nextInt(...) is O(1).

    public RandomizedSet() {
        values = new HashSet<>(); // assume this is singly-linked
        r = new Random();
    }

    public boolean insert(String value) {
        if (values.contains(value)) { return false; }
        values.add(value);
        return true;
    }

    public boolean remove(String value) {
        if (!values.contains(value)) { return false; }
        values.remove(value);
        return true;
    }

    public String pick() {
        if (size() == 0) { return null; }
        int index = r.nextInt(size());
        Iterator<String> itr = values.iterator();
        for (int i = 0; i < index; i++) {
            itr.next();
        }
        return itr.next();
    }

    public int size() {
        return values.size();
    }
}
```

Give the **in-practice, simplified, tight big-oh runtime** of the above implementation for the following methods

1) insert:

2) remove:

3) pick:

Yafqa isn't satisfied with either implementation. He thinks he should be able to achieve **O(1) time on *all* operations in-practice**, using some combination of data structures. Describe how you might do this below:

1) Describe what data structures you'll use and what they represent (≤ 5 sentences).

2) Describe your implementation of insert (either plain english (≤ 3 sentences) or pseudocode).

3) Describe your implementation of remove (either plain english (≤ 3 sentences) or pseudocode).

4) Describe your implementation of pick (either plain english (≤ 3 sentences) or pseudocode).