CS2040S: Data Structures and Algorithms

Recitation 1

Problem 1. Order of Growth (Review)

Problem 1.a. Consider the following function `isPrime` for checking if an integer n is prime:

```
public static boolean isPrime(int n) {
    if (n<2) {
        return false;
    }
    for (int i=2;i<n;i++){
        if (isDivisible(n,i)) {
            return false;
        }
    }
    return true;
}
```

What is the order of growth for `isPrime` given the following order of growths for `isDivisible(n,i)`?

1. Time: $O(1)$
2. Time: $O(n)$
3. Time: $O(i)$

Problem 1.b. Next, consider the following function `isPrime2` for checking if an integer n is prime:

```
public static boolean isPrime2(int n) {
    if (n<2) {
        return false;
    }
    for (int i=2;i<sqrt(n);i++){
        if (isDivisible(n,i)) {
            return false;
```

```
            }
        }
        return true;
    }
```

What is the order of growth for `isPrime2` given the following order of growths for `isDivisible(n,i)`?

1. Time: $O(n)$
2. Time: $O(i)$

Problem 1.c. Consider yet another variant of an algorithm to check if an integer n is prime that we call `isPrime3` below:

```
public static boolean isPrime3(int n) {
        if (n<2) {
            return false;
        }
        for (int i=2;i<=sqrt(n);i++){
            if (isPrime3(i) && isDivisible(n,i)) {
                return false;
            }
        }
        return true;
    }
```

We are simulating an (inefficient) version of the Sieve of Eratosthenes (see https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes). For this algorithm, express the time complexity expression as a concise recurrence relation (solving it is left as an optional challenge). Assuming the order of growth for `isDivisible(n,i)` is O(1), how does `isPrime3` compare against `isPrime2`?

Problem 2. How Much Do You Make?

Learning objectives:

1. How do we make an algorithmic question precise?

2. Metrics: what makes a solution a good solution? How do we measure goodness?

3. Problem solving: how to iterate a solution?

A group of graduating CS students got offered well-paying jobs. They all want to know the market rate (average salary), but they do not want to tell anyone their own salary. (No one wants to brag or be embarrassed!) What should they do?

Problem 2.a. Come up with an algorithm that will allow the students to determine the average pay of the group without revealing their own salaries to any of the other members in the group.

If we want to evaluate how good our algorithm is, what metrics should we use? Realize that when evaluating an algorithm, it isn't always obvious what metrics we care about, and you have to think hard to make sure you are optimizing the right thing!

As you come up with an algorithm, think about what invariants are being satisfied. Since the goal is related to average salary, maybe you want an invariant that relates the current state to the average salary being computed?

ProTip: The choice of metrics should provide an good measure of how well the chosen objective is met. Metrics therefore drives the solution. You may be looking at multiple metrics at once, which is in the case where you are optimizing for multiple objectives. Often times, a single total metric can be obtained by linearly combining multiple metrics via weighing the relative importance of the objectives they correspond to.

Problem 2.b. Now think about what happens if k members of the group decide to collude and share information. Is your algorithm still able to preserve the privacy of all the members, or could the salary for some of the members be leaked?

If there is a possibility for privacy to be compromised, modify your proposed algorithm to "fix" this leak. (Think about the invariant that we used last time.)

How well does your new algorithm perform? Is that the best we can do? Can you do even better? What is the best that we can do?

Problem 2.c. *[Optional Part]* We had earlier assumed that all the participants in the group are all honest. Suppose that one of the group members is a saboteur who wants to mislead the rest by submitting a salary that is many times higher or lower than the actual salary. You can assume that the salaries have a normal distribution with a "reasonable" variance. This saboteur would introduce a salary point that will appear as an outlier in the full data set. What if instead of just 1 saboteur, we had a small number s of them?

Given your proposed algorithm, how else could a saboteur decide to do to cause inaccuracies in the results?