CS2040S

Recitation 1
AY22/23 S2

Question 1: Orders of Growth

Quick recap

$$\sum_{i=1}^{n} i = 1 + 2 + \ldots + (n-1) + n$$

Each pair sums to $n+1$

$$2\sum_{i=1}^{n} i = \underbrace{\left(\boxed{\begin{matrix} 1 \\ + \\ n \end{matrix}} + \boxed{\begin{matrix} 2 \\ + \\ n-1 \end{matrix}} + \boxed{\begin{matrix} \ldots \\ + \\ \ldots \end{matrix}} + \boxed{\begin{matrix} (n-1) \\ + \\ 2 \end{matrix}} + \boxed{\begin{matrix} n \\ + \\ 1 \end{matrix}} \right)}_{n \text{ terms}}$$

$$= n(n+1)$$

Trick: We sum the entire sequence twice but visually reverse the second sequence and match terms termwise with original sequence

So $$\sum_{i=1}^{n} i = (n^2+n)/2 \qquad Q.E.D$$

Test yourself!

Which type of algorithmic procedure *usually* entails a arithmetic summation series for its time complexity?

Test yourself!

Which type of algorithmic procedure *usually* entails a arithmetic summation series for its time complexity?

Answer: For-loops, because computation time is cumulative and each iterative step incurs a cost that is usually some constant time more than the previous step due to the growing sub-problem in the loop body.

Prime numbers

A prime number is a natural number greater than 1 that is only divisible by 1 and itself.

Test yourself!

What is the most naive way to test if a number n is prime?

Test yourself!

What is the most naive way to test if a number n is prime?

Answer: Test its divisibility across all numbers from 2 to $n-1$!

Version 1

```java
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;
}
```

Test yourself!

What is the worst-case?

Test yourself!

What is the worst-case?

Answer: When n is prime!

Problem 1.a.

```java
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. $O(1)$ time
2. $O(n)$ time
3. $O(i)$ time

Version 1

```
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;
}
```

Can we do better?

Test yourself!

Do we really need to test divisibility from 2 to $n-1$?

Test yourself!

Do we really need to test divisibility from 2 to $n-1$?

Answer: No, at simple glance you should realize that we can stop at $n/2$.

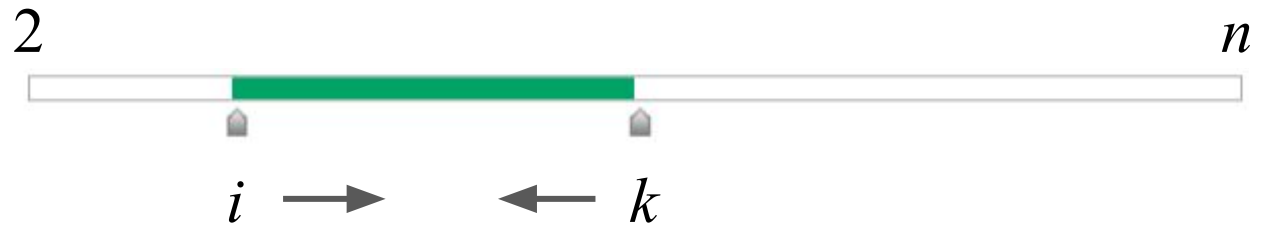
With some careful thinking, you'll realize we can stop at $\sqrt{n}$.

If you have taken or are taking CS1231, you will learn that the largest possible factor of a number can only be its root.

Why?

As we vary i from 2 to $n-1$, at which point does the relative magnitudes of k and i flip?

$$k \times i = n$$



They'll cross at $\sqrt{n}$, after which they end up exploring in each other previously explored ranges.

Version 2

```
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;
}
```

Is there a bug?

Problem 1.b.

```java
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. $O(n)$ time
2. $O(i)$ time

Version 3: Sieve of Eratosthenes

```java
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;
}
```

Problem 1.c.

```
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;
}
```

What is the order of growth for `isPrime3` assuming the order of growth for `isDivisible(n,i)`is $O(1)$ time

Problem 1.c.

```
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;
}
```

$O(\sqrt{n})$ iterations

$T(i) + O(1)$ operations

Problem 1.c.

```
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;
}
```

$$T(n) = T(2) + T(3) + \cdots + T(\sqrt{n}) + \sqrt{n}$$

$$= \sum_{i=2}^{\sqrt{n}} T(i) + \sqrt{n}$$

$$\leq \sqrt{n}T(\sqrt{n}) + \sqrt{n}$$

Problem 1.c.

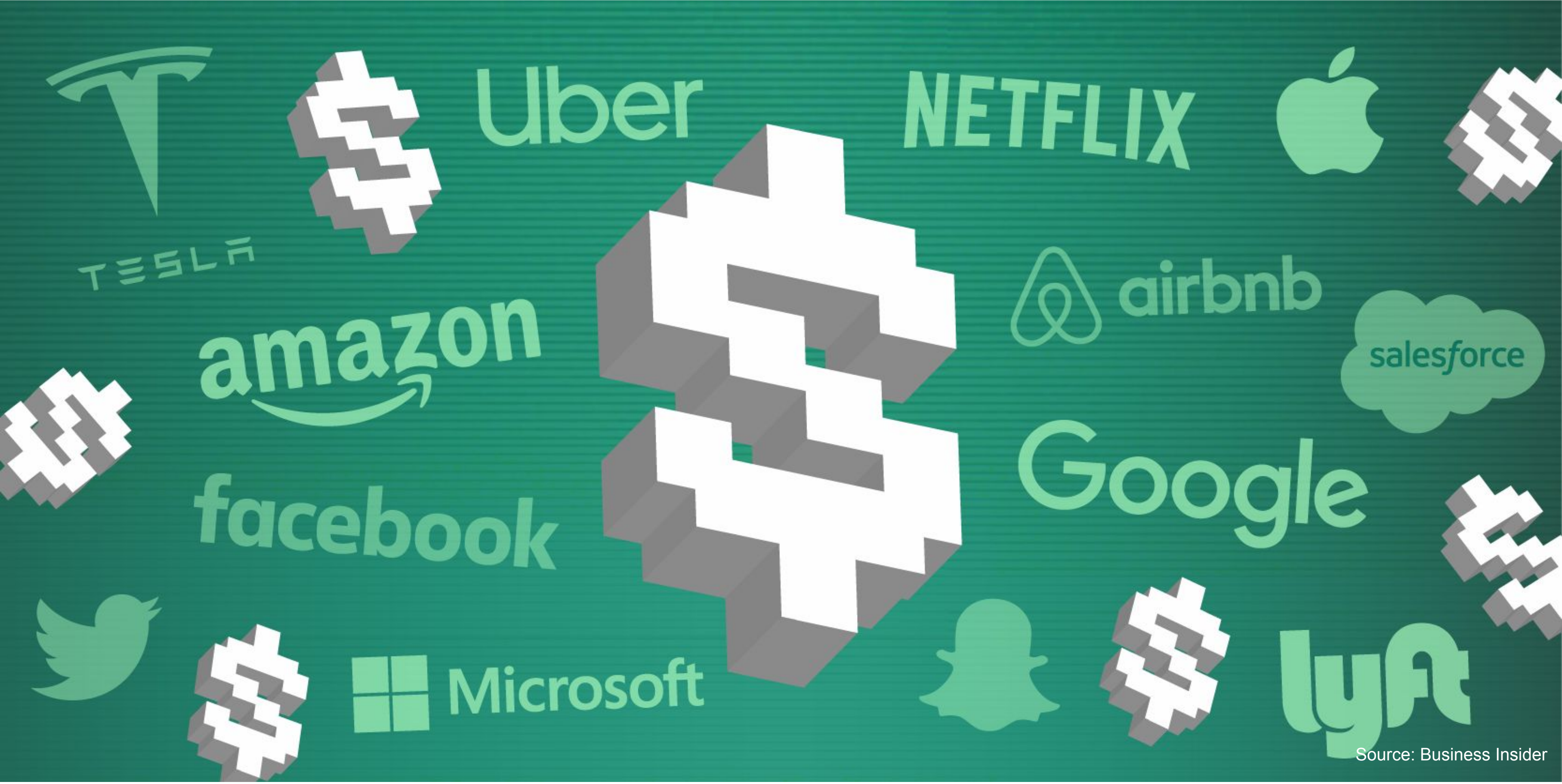
```
public static boolean isPrime3(int n) {
  if (n<2) {
    return false;
  }
  for (int i=2;i<=sqrt(n);i++){
    if (isPrime3(i) && isDivisible(n,i)) {
      return false;
    }
  }
  return true;
}
```

$$T(n) = T(2) + T(3) + \cdots + T(\sqrt{n}) + \sqrt{n}$$

$$= \sum_{i=2}^{\sqrt{n}} T(i) + \sqrt{n}$$

$$\leq \sqrt{n}T(\sqrt{n}) + \sqrt{n}$$

$$T(n) = O(n)$$



Source: Business Insider

Question 2: How much do you make?

Background

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!

Hint

Source: https://www.ultimatecampresource.com/camp-games/circle-games/

Attempt 1

- Pick a leader L who starts and ends the procedure
- Everyone preselect a random offset r (what range?)
- Accumulated sum is propagated clockwise/anticlockwise (just be consistent)

First round:

- L adds its salary $s_L +$ its random offset r_L and pass down the sum to the next person i in sequence
- i receives the sum, adds to it its salary $s_i +$ its offset r_i and pass down the sum to the next person in sequence
- This continues until we end at L

Attempt 1

Second round:

- Same as first round except for each person i, instead of adding their salary s_L $+$ random offset r_L, they simply deduct their random offset r_L from the sum and pass it down
- This continues until we end at L

Finally,

Leader divides total sum by the total number of people n to obtain the mean and shares that with everyone.

Test yourself!

What is the number of operations needed for this proposal?

Test yourself!

What is the number of operations needed for this proposal?

Answer: n random number generations, n plus operations, n minus operations, 1 division operation.

So $3n+1$ operations total.

Can we do *faster*?

But..

- It doesn't really ensure privacy, because how much you add/subtract from the value means something
  - $i-1$ and $i+1$ person can collude and figure out the salary of i^{th} person
  - First round: They figured out salary of i plus its offset
  - Second round: They figured offset of i
- From what space should you choose your random offset?

Test yourself!

Doesn’t aggregate amount already hide individual information?

Why are we using random offsets at all?

Test yourself!

Doesn’t aggregate amount already hide individual information?

Why are we using random offsets at all?

Answer: The person immediately next to the leader will always figure out the leader’s salary! We just want to overcome this (for now). So we need random offsets, but do we need n random offsets?

Attempt 2

Changes to attempt 1:

- Only leader picks an initial random offset
- Only 1 round
- When the round ends with the leader, it simply subtract off its secret initial offset and divide it by n

Test yourself!

What is the number of operations needed for this proposal?

Test yourself!

What is the number of operations needed for this proposal?

Answer: 1 random number generations, n plus operations, 1 minus operations, 1 division operation.

So $n+3$ operations total.

This is 3 times faster!

But..

- $i-1$ and $i+1$ person can still collude and figure out the salary of i^{th} person
- Now, the last person will always figure out the initial offset chosen by the leader ⇒ can collude with the first person to figure out leader's salary
- This is still not privacy preserving..
- But it's certainly faster!

Did I ever tell you I'm super-fast at math?
Yeah? Okay, what's 912 X 43?
741!
Dude, that's not even close!
Fast, though.
imgflip.com

Test yourself!

What is a straightforward metric we can use for privacy preservation here?

Test yourself!

What is a straightforward metric we can use for privacy preservation here?

Answer:

The number of colluding individuals it takes to ruin the game!

With this metric, the past 3 algorithms all have privacy preservation of degree 2 (the higher the better).

Discussion point

Your fellow classmate also came up with an alternative definition for privacy preservation by suggesting to use the distance between the “best guess” of a victim’s salary (by an adversary) from the victim’s actual salary.

This is perfectly reasonable! Realize this metric might drive a different category of solutions entirely! We just need to define how to derive the “best guess”.

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.

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?

Challenge

Can we come up with a solution where privacy preservation only fails when EVERYONE ELSE colludes?

I.E an algorithm of privacy preservation degree $n - 1$

Pause and think for a minute.. Is it even possible :O

Key idea

- Each person partitions up its own salary into n chunks of random sizes
- Keeps one chunk, distributes the remaining $n - 1$ chunk among the $n - 1$ others
- Total sum of salaries is obtained by summing up all the $n \times n$ partial chunks

Other possible ideas: statistical approach

What if we aren't so interested in the *exact* mean? We just want an approximate value with some reasonable guarantees that it is close to the true value? E.g. can tolerate $\pm\$10$ answer.

Take a look at the beans in a jar experiment: 160 people were able to guess the number of jelly beans in a jar accurate to .1% when their answers were averaged. Out of 4510 beans, the average guess was 4514. If you have enough participants, you can leverage on the Law of Large Numbers!

Other possible ideas: statistical approach

Some of you have also proposed a similar idea: Everyone samples from some gaussian distribution centered at zero mean and add it to their own salaries as *noise*. Doing so will *perturb* their individual salaries and when we take the overall mean, the effects of the noise would be *cancelled out* (since the mean due to noise is zero).

Mathematically, suppose the random variable sampled from the distribution for salaries is S and the random variable sampled from the noise distribution is kZ where k is some non-zero constant and Z sampled from the standard normal distribution, then:

$$\begin{aligned}\mathbb{E}[\![S + kZ]\!] &= \mathbb{E}[\![S]\!] + k\mathbb{E}[\![Z]\!] \\ &= \mathbb{E}[\![S]\!] \qquad \text{since } \mathbb{E}[\![Z]\!] \text{ is zero}\end{aligned}$$

Lesson learnt

1. How you (re)define the problem
2. Determines what are the metrics of success
3. Which in turn drives the solution

Steps 1-3 are iterated again each time we develop *deeper problem insights* and challenge *hidden assumptions* from prior attempts. It takes many of such iterations to attain “good” solutions!