CS2040S

Recitation 1
AY20/21S2

Greeting!

My full name is Nguyen Thi Mai Huong /hʊəːŋ/ (흐엉), but people call me Mia for short

My email: maihuongnt@u.nus.edu.sg

First things first

- Please rename yourself according to your Coursemology name, we will be running a script to mark attendance therefore require exact string matching
- **Please turn on your video**: I want to see your pretty hand handsome faces. Talking with a blank screen can be very depressing.
- **Please unmute yourself and speak** when you have questions, I usually don’t check the chat.

ProTips

- Use repl.it for quick experiments with Java
- Jin prepared a java cheatsheet here, you can check this out. Make sure you give the repo a star if you find it helpful.
- Are you a visual learner? Use Visualgo (developed by Lecturer Steven Halim) to augment your algorithmic journey!

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

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

```
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 $n/2$.

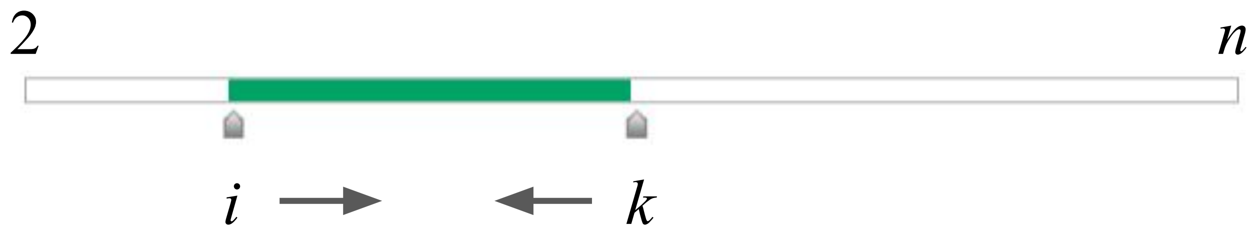
With some careful thinking, you'll realize we can stop at $\sqrt{n}$.

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

```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;
}
```

Is there a bug?

Problem 1.b.

```
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

```
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.

```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;
}
```

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

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

Try it! Expanding the sequence

$$
\begin{aligned}
T(n) &= T(2) + T(3) + \cdots + T(\sqrt{n}) + \sqrt{n} \\
&= \sum_{i=2}^{n^{1/2}} T(i) + n^{1/2} \\
&\le n^{1/2} T(n^{1/2}) + n^{1/2} \\
&= n^{1/2}(n^{1/4} T(n^{1/4}) + n^{1/4}) + n^{1/2} \\
&= n^{3/4} T(n^{1/4}) + n^{3/4} + n^{1/2} \\
&= n^{3/4}(n^{3/16} T(n^{1/16}) + n^{3/16} + n^{1/8}) + n^{3/4} + n^{1/2} \\
&= n^{15/16} T(n^{1/16}) + n^{15/16} + n^{7/8} + n^{3/4} + n^{1/2}
\end{aligned}
$$

since $\sum_{i=2}^{n^{1/2}} T(i) \le n^{1/2} \times T(n^{1/2})$

• • •


A FEW
MOMENTS LATER

Expanding the sequence

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

$$= \sum_{i=2}^{n^{1/2}} T(i) + n^{1/2}$$

$$\leq n^{1/2} T(n^{1/2}) + n^{1/2}$$

since $\sum_{i=2}^{n^{1/2}} T(i) \leq n^{1/2} \times T(n^{1/2})$

$$= n^{1/2}(n^{1/4} T(n^{1/4}) + n^{1/4}) + n^{1/2}$$

$$= n^{3/4} T(n^{1/4}) + n^{3/4} + n^{1/2}$$

$$= n^{3/4}(n^{3/16} T(n^{1/16}) + n^{3/16} + n^{1/8}) + n^{3/4} + n^{1/2}$$

$$= n^{15/16} T(n^{1/16}) + n^{15/16} + n^{7/8} + n^{3/4} + n^{1/2}$$

$$= nT(1) + \cdots + n^{31/32} + n^{15/16} + n^{7/8} + n^{3/4} + n^{1/2}$$

Continue expanding leftwards till infinity...

$$= nT(1) + \cdots + n^{1-1/32} + n^{1-1/16} + n^{1-1/8} + n^{1-1/4} + n^{1-1/2}$$

$$= nT(1) + \cdots + \frac{n}{n^{1/32}} + \frac{n}{n^{1/16}} + \frac{n}{n^{1/8}} + \frac{n}{n^{1/4}} + \frac{n}{n^{1/2}}$$

What's the pattern here?

Spot the pattern

$$nT(1) + \cdots + \frac{n}{n^{1/32}} + \frac{n}{n^{1/16}} + \frac{n}{n^{1/8}} + \frac{n}{n^{1/4}} + \frac{n}{n^{1/2}}$$

Numerators are all n. As we progress from left to right, the denominator gets squared.

Okay, but what is $T(1)$?

According to our algorithm, $T(1)$ would be caught by the first if-statement so it takes *some* constant time.

Trick: We are free to assign it a constant amount that is convenient for our analysis!

Let's assign $T(1) = ½$ time.

Since it's an *infinite sequence*, we are now ready to **rewrite** it from left to right ;)

Re-writing from left to write

$$
\begin{aligned}
T(n) &= \frac{n}{2} + \frac{n}{2^2} + \frac{n}{2^4} + \frac{n}{2^8} + \dots \\
&= n \sum_{0}^{\infty} \frac{1}{2^{2^i}} \\
&= nL && \text{where } L \approx 0.81642 \\
&= O(n)
\end{aligned}
$$

Q.E.D

This derivation was adapted from:
https://math.stackexchange.com/questions/1330747/solving-the-recurrence-tn-sqrt-n-t-sqrtn-sqrtn

Problem 1.d.

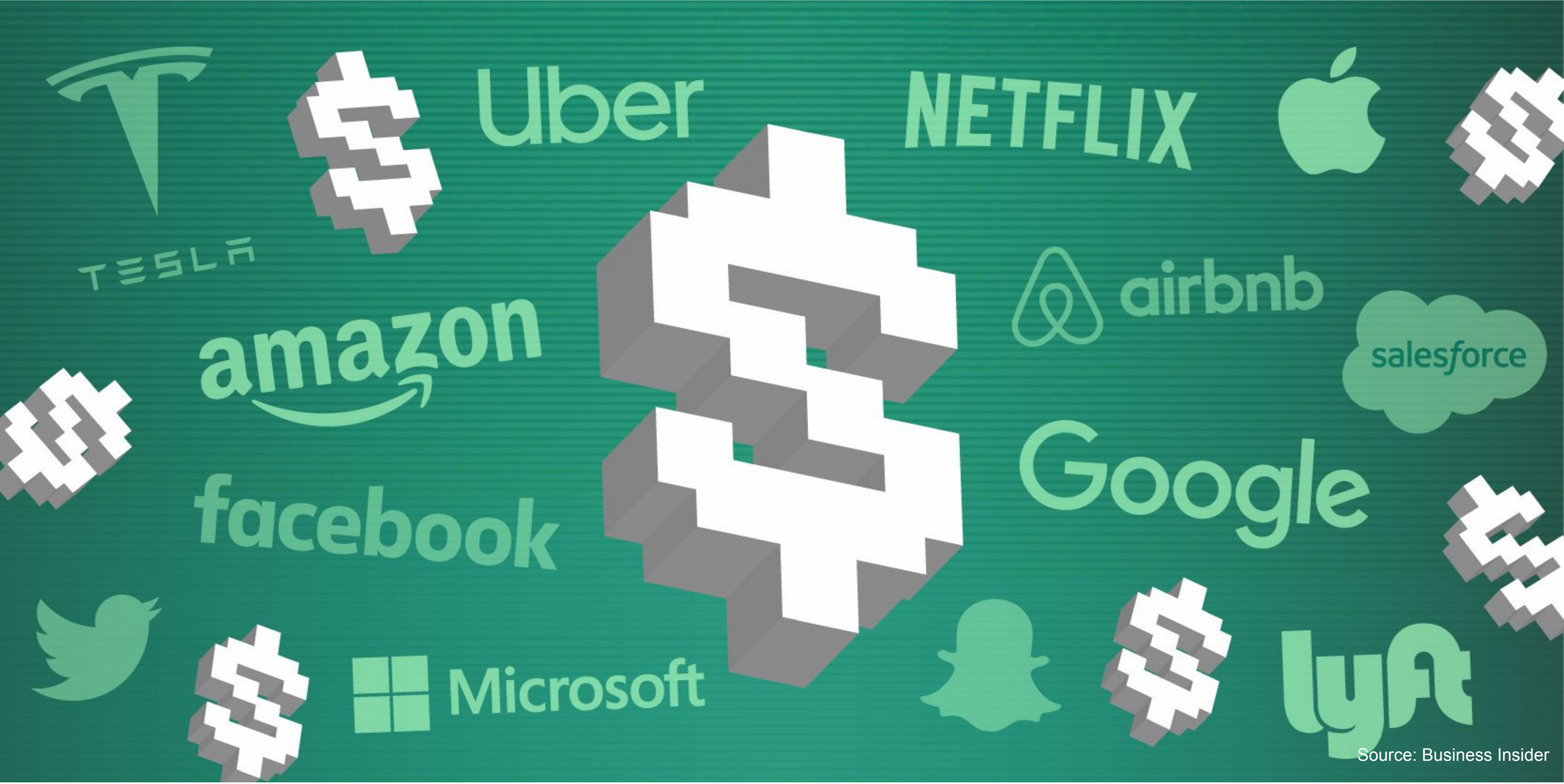
1. Is $\log_2(n) = O(\log(n))$?
2. Is $2^{2n} = O(2^n)$?

Recall

By definition:

$$\mathrm{f}(n) = \mathrm{O}(\mathrm{g}(n)) \quad \text{iff.} \quad \mathrm{f}(n) \leqslant k \cdot \mathrm{g}(n) + c$$

For some values k (non-zero) and c (can be zero)



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:

1. n random number generation: n operations
2. Salary + Random offset: n operations
3. Adding person value to the previous sum: $n-1$ operations (This is not needed for person 1)
4. Taking out random offset: n operations
5. Division: 1 operation

So $4n+1$ 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?

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

Attempt 3

Changes to attempt 2:

- We denote the maximum salary in the group a m
- Leader first picks an impossibly large number $P = m \times n$ (which everyone can agree on)
- Leader then picks a secret random initial offset r in the range $[1, P]$
- Leader calculates $acc = (s_L + r) \bmod P$ and pass it on to the next person
- Every person adds their salary into acc and pass it to the next person as usual
- When it ends, leader calculates $((acc - r) \bmod P) / n$

Attempt 3

Example

1. 3 person with salaries $\$25,000$, $\$30,000$, $\$6,000$
2. Everyone agrees that $P = 100,000$ is sufficiently big as it is greater than $3 \times 30,000$
3. Leader chooses a secret random offset between 0 and $100,000$, say $90,423$
4. Leader calculates $(25,000 + 90,423) \bmod 100,000$, obtaining $15,423$. Notice that every value between 0 and $100,000$ is equally likely after the modulo!
5. Leader secretly pass that value to person two, who then calculates $15,423 + 30,000 = 45,423$, and pass it on to the third person
6. Third person calculates $45,423 + 6,000 = 51,423$ and pass it back to leader
7. Leader calculates $((51,423 - 90,423) \bmod 100,000)$, obtaining $61,000$
8. Finally $61,000 / 3$ is revealed to everyone

Test yourself!

What is the significance of choosing the large number P?

Test yourself!

What is the significance of choosing the large number P?

Answer: Think of it as a P-hours clockface and the accumulated sum is time. The leader randomly advance the start time by $r \in [1, P]$ hours. Each person then advances the time equals to its salary. At the end of the round, the leader unwinds the clock back by r hours to obtain the true sum.

This also explains why P must be so large, because the true sum must fit within the chosen clockface, if not it will wrap-around and we won't know how many times that happened. Note that the "fake sum" will wrap around at most once.

But..

- People still can collude the same way as attempt 2
- The possibility of collusion obviously damages privacy preservation but how much is too much?
- We haven’t explicitly tackled that problem because we haven’t specified a metric for privacy preservation

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?

Pause and think for a minute..

Key idea

- Each person generates random numbers that all add up to their own salary
- Give one such random number to each other person in the circle, keeps one piece.
- Each of them will sum n pieces and they will pass all the partial sums around in a circle to obtain the total sum of salaries.

Test yourself!

If you can give your values to $n-1$ people simultaneously and can you receive numbers from $n-1$ people simultaneously, what is the time complexity for conducting all the sending and receiving?

Test yourself!

If you can give your values to $n-1$ people simultaneously and can you receive numbers from $n-1$ people simultaneously, what is the time complexity for conducting all the sending and receiving?

Answer: $\mathrm{O}(1)$. Can we really achieve this in practice? If you can only receive one piece of information at a time, is there a lower bound of $\Omega(n)$?

In Practice

Can we compute the average salary of everyone in $O(1)$?

Hint: Do you still have to go one round to sum up everyone’s amounts after the sending/receiving exchange?

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:

$$\mathbb{E}[\![S+kZ]\!] = \mathbb{E}[\![S]\!] + k\mathbb{E}[\![Z]\!]$$

$$= \mathbb{E}[\![S]\!] \qquad \text{since } \mathbb{E}[\![Z]\!] \text{ is zero}$$

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* from prior attempts. It takes many of such iterations to attain good solutions!