Hash Tables & Big O

Week 2

Your Instructor: Paulina

- grew up in Bay Area; now lives in New York City
- path: UC Berkeley→ Codepath → Uber
  - Dec 2014: graduated from UC Berkeley (Computer Science)
  - Feb 2015: completed CodePath's Android Bootcamp
  - March 2015 - May 2020: Software Engineer @ Uber
    (past internships: StubHub, Trulia)

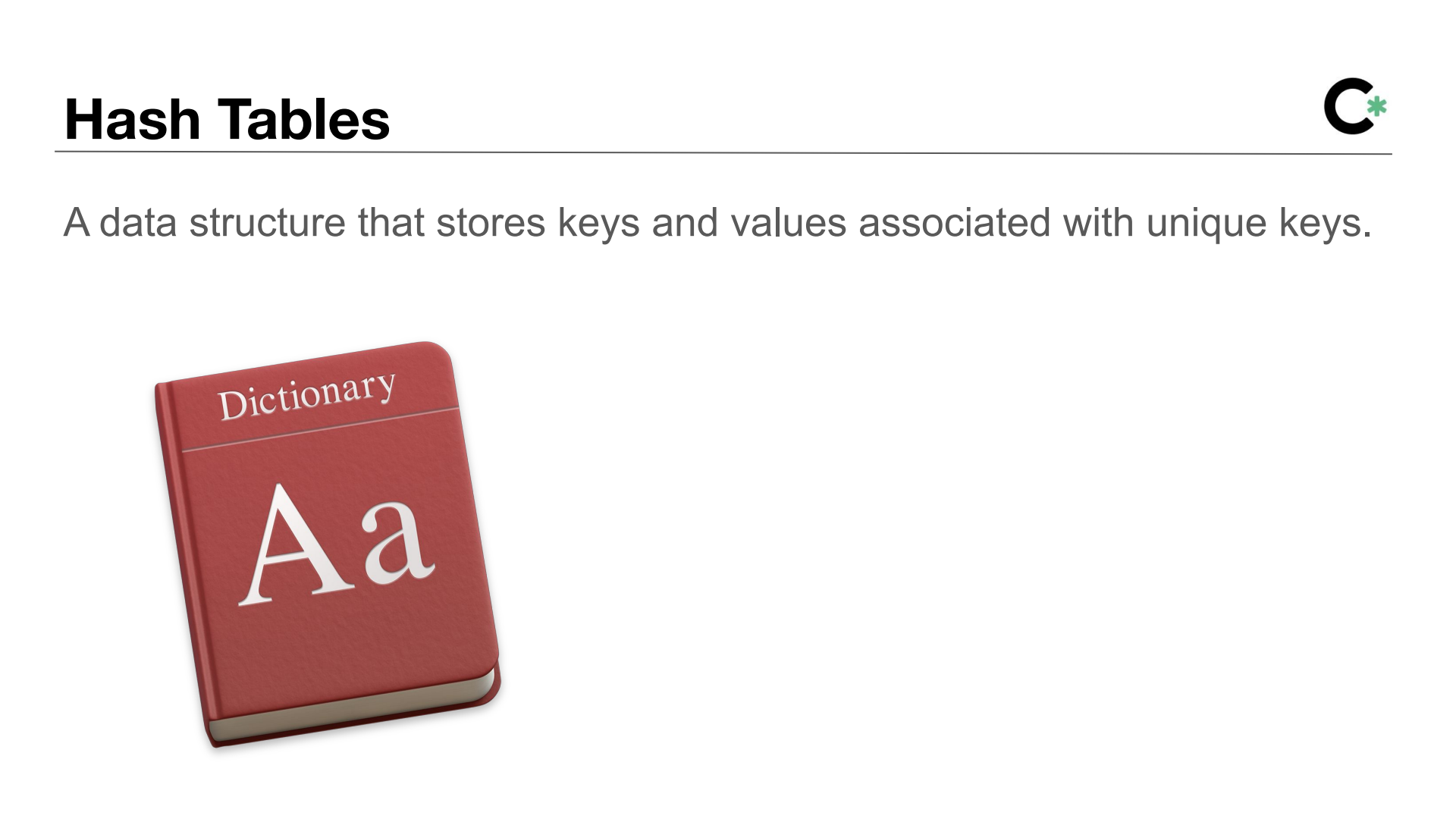
Let's connect!

LinkedIn: linkedin.com/in/mpaulinar/
Instagram: @poww.lina
Email: m.paulina.ramos@gmail.com

Hash Tables

A data structure that stores keys and values associated with unique keys.

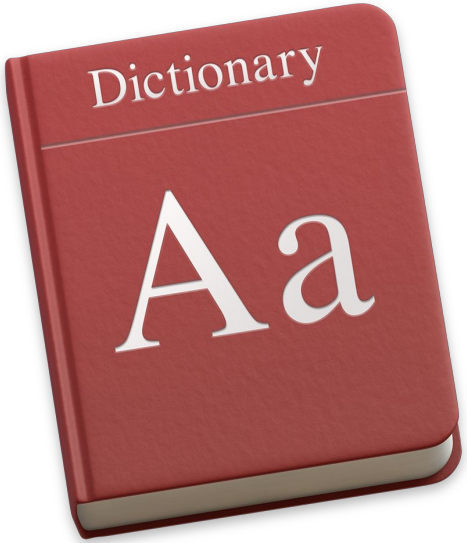
Hash Tables



A data structure that stores keys and values associated with unique keys.

Hash Tables

A data structure that stores keys and values associated with unique keys.



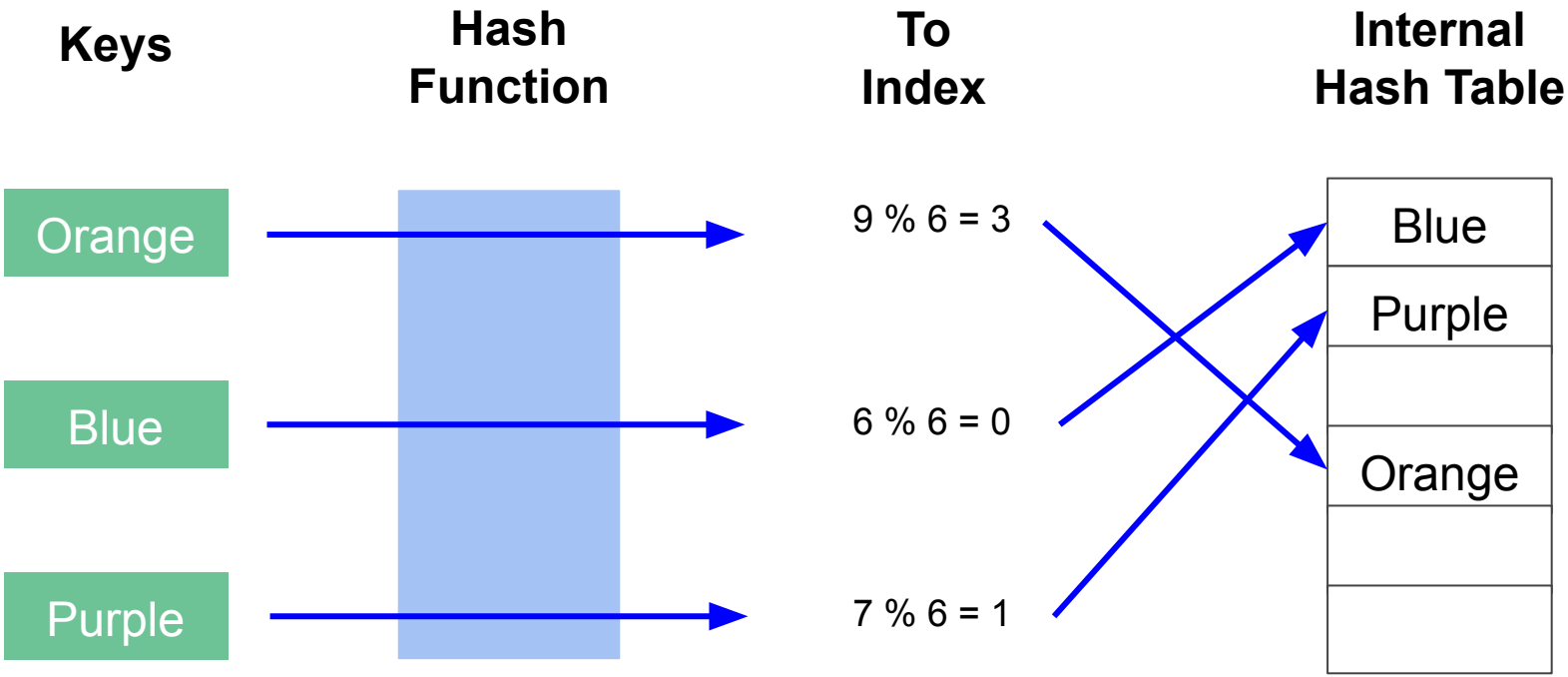
- {key: value, key2: value2...}
- Unordered

Hash Tables

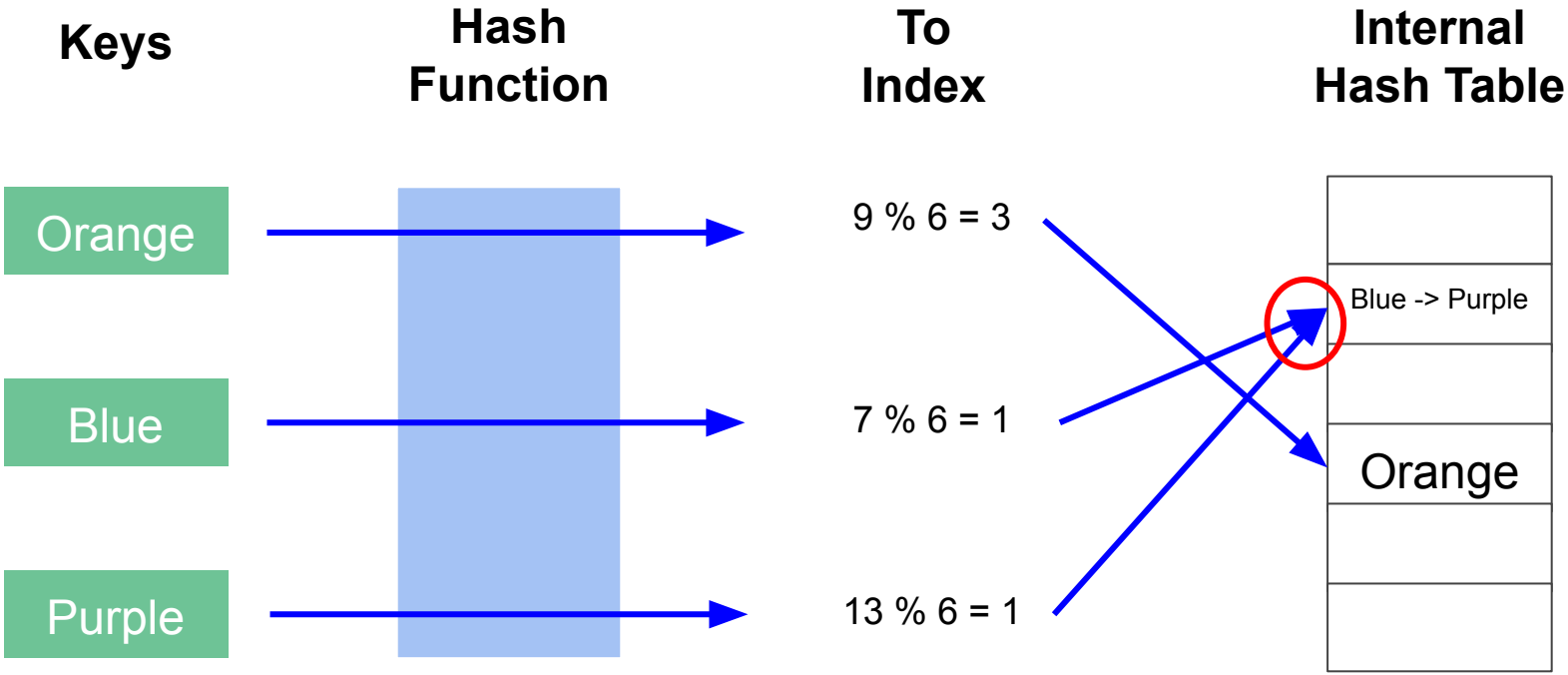

```java
Hashtable<String, Integer> topTestScores =
    new Hashtable<String, Integer>();
topTestScores.put("John", 99);
topTestScores.put("Sandra", 100);
topTestScores.put("Sammy", 85);
// Prints 100
System.out.println(topTestScores.get("Sandra"));
```


```python
topTestScores =
    {'John': 99, 'Sandra': 100, 'Sammy': 85}
# Prints 85
print topTestScores['Sammy']
```

Hash Function



Collisions


Keys
Hash Function
To Index
Internal Hash Table
Orange
Blue
Purple
9 % 6 = 3
7 % 6 = 1
13 % 6 = 1
Blue -> Purple
Orange

Big O

Efficiency: How well the function scales with the size of the dataset.

Big O

Efficiency: How well the function scales with the size of the dataset.

Time: Runtime, Time Complexity, Time Constraints

- How long a function takes to complete

Big O

Efficiency: How well the function scales with the size of the dataset.

Time: Runtime, Time Complexity, Time Constraints

- How long a function takes to complete

Space: Extra Memory, Space Complexity, Memory Constraint

- How much memory a function consumes while executing

Big O

n: Size of the algorithm’s input

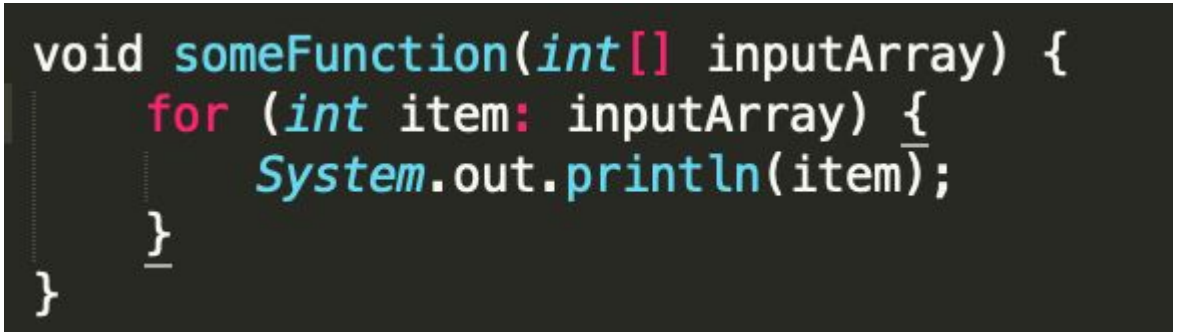
Big O

n: Size of the algorithm’s input

x: number of things algorithm does or creates. O(x * n)

- O(2n) -> O(n)
- Constants are dropped

Big O


void someFunction(int[] inputArray) {
for (int item: inputArray) {
System.out.println(item);
}
}

Big O

Time: ??

Space: ??

```
void someFunction(int[] inputArray) {
    for (int item: inputArray) {
        System.out.println(item);
    }
}
```

Big O

Time: O(n)

Space: ??

```
void someFunction(int[] inputArray) {
    for (int item: inputArray) {
        System.out.println(item);
    }
}
```

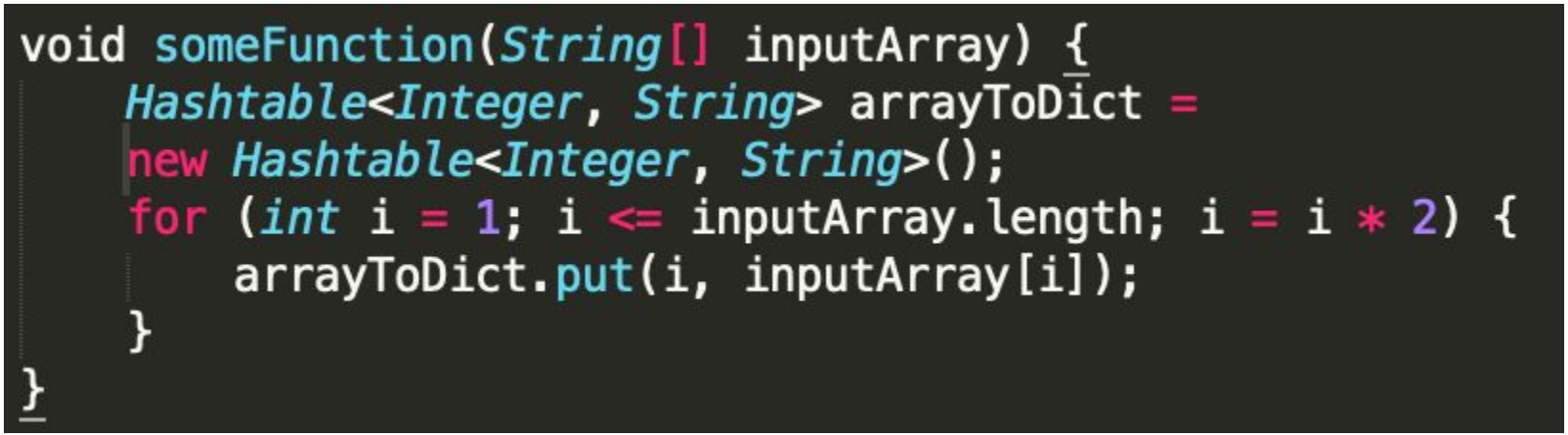
Big O

Time: O(n)

Space: O(1)

```
void someFunction(int[] inputArray) {
    for (int item: inputArray) {
        System.out.println(item);
    }
}
```

Big O


void someFunction(String[] inputArray) {
Hashtable<Integer, String> arrayToDict =
new Hashtable<Integer, String>();
for (int i = 1; i <= inputArray.length; i = i * 2) {
arrayToDict.put(i, inputArray[i]);
}
}

Big O

```
void someFunction(String[] inputArray) {
    Hashtable<Integer, String> arrayToDict =
    new Hashtable<Integer, String>();
    for (int i = 1; i <= inputArray.length; i = i * 2) {
        arrayToDict.put(i, inputArray[i]);
    }
}
```

Time: ??

Space: ??

Big O

```
void someFunction(String[] inputArray) {
    Hashtable<Integer, String> arrayToDict =
    new Hashtable<Integer, String>();
    for (int i = 1; i <= inputArray.length; i = i * 2) {
        arrayToDict.put(i, inputArray[i]);
    }
}
```

Time: O(log n) Space: ??

Big O

```
void someFunction(String[] inputArray) {
    Hashtable<Integer, String> arrayToDict =
    new Hashtable<Integer, String>();
    for (int i = 1; i <= inputArray.length; i = i * 2) {
        arrayToDict.put(i, inputArray[i]);
    }
}
```

Time: O(log n) Space: O(log n)

Hash Tables - Runtimes

	Lookup	**Insert**	**Delete**
Best Case	O(1)	O(1)	O(1)
Worst Case	O(n)	O(n)	O(n)

Big O

Read this!

https://guides.codepath.org/compsci/Big-O-Complexity-Analysis

Hash Tables - Pros & Cons

PROS

1. Fast Search (Lookup)
2. Fast Insertions
3. Fast Deletion
4. Synchronized & Thread-safe

CONS

1. Unordered
2. Takes up memory

Hash Tables - useful for caching :)

Related Data Structures

PROPERTY	**HashTable**	**HashMap**	**HashSet**	**LinkedHashMap**
iteration order	random	random	random	ordered (by insertion)
best-case runtime complexity	O(1)	O(1)	O(1)	O(1)
null keys/values allowed?	no	yes	yes	yes
duplicate values?	yes (but not keys)	yes (but not keys)	no	yes (but not keys)
synchronized & thread-safe?	yes	no	no	no

Hash Tables - Common Problems

- Find duplicates
- Sums
- Find subsequence
- LRU (Least Recently Used) Cache

UMPIRE

Understand

Match

Pseudocode / **P**lan

Implement

Review

Evaluate

UMPIRE

Understand

Match

Pseudocode / **P**lan

Implement

Review

Evaluate

UMPIRE

Understand

- Ask clarifying questions
- Verify inputs and outputs
- State edge cases
- Time or space constraints

Match

- Find a data structure that might help solve the problem
- Are there any special techniques that can be used

Pseudocode / **P**lan

- Talk through different solutions and their tradeoffs
  - Talk with interviewer to decide on which one to try first
  - If only one solution come to mind, continue on with it
- This would be the start of your code
- Create any helper functions (just the stubs are fine)
- Write the high level steps in words
  - For example: loop through array

Isomorphic Strings

Given two strings s and t, determine if they are isomorphic.

Two strings are isomorphic if the characters in **s** can be replaced to get **t**. No two characters may map to the same character, but a character may map to itself.

Understand

Input and Output

- s = “foo”, t = “bar” -> false
- s = “mom”, t = “dad” -> true
- s = “test”, t = “boo” -> false
- s = “”, t = “” -> false
- s = “mop”, t= “dad” -> false

Complexity

- Are there any run time constraints? *Let’s say there isn’t one for now.*
- Can we use extra memory? *yes*

Clarifying Questions

- Will the inputs be valid strings? *yes*

Match

Special techniques and/or data structures that we can use to solve this problem?

- Since a character from s will be mapped to a character from t, we can use hash tables to do the unique mapping
- It looks like we would need to loop through both strings to create that map
    - We can convert the string into an array of characters
    - We can loop through the string and get the character at that index.

Pseudocode / Plan

Different ways to solve this problem? Trade Offs?

- Simple solution:
  - (1) Loop through characters of s
  - (2) Keep a reference to the first mapped character of t
  - (3) Loop through t to see if there are other occurrences that map to the same s character for that index.
  - *Time complexity is O(n ^ 2); Space complexity is O(1)*

Pseudocode / Plan

Different ways to solve this problem? Trade Offs?

- Time efficient solution: Create a hash map, loop through s and t, if s's character doesn't exist in the map, create one with value of t's character. If it does exist, look to see if the value matches t's current character.
    - *Example:* s = "foo", t = "bar" -> false

Pseudocode / Plan

Different ways to solve this problem? Trade Offs?

- Time efficient solution: Create a hash map, loop through s and t, if s's character doesn't exist in the map, create one with value of t's character. If it does exist, look to see if the value matches t's current character.
  - *Example:* s = "foo", t = "bar" -> false
  - *Time complexity is O(n); Space complexity is O(n) where n is the length of the string.*

Pseudocode / Plan

1. Create a map where the key is characters in s and the value is the unique mapping of the character from t. Create a set to keep track of visited t chars.
2. Loop through s and t, return false if the key exists but the value is different
3. If not: if the t char is in set, return false. else, enter the key/value to the map
4. Return true at the end of the loop

Pseudocode / Plan

1. Create a map where the key is characters in s and the value is the unique mapping of the character from t. Create a set to keep track of visited t chars.
2. Loop through s and t, return false if the key exists but the value is different
3. If not: if the t char is in set, return false. else, enter the key/value to the map
4. Return true at the end of the loop

Let's draw out the map through the loop: s = "foo", t = "bar"

1. {f : b}
2. {f : b, o : a}
3. {f : b, o : a} -> o is already there and the value is not r, so return false

Group Exercise

Group Exercise

- Groups of 5 - 6: **Understanding**, **Matching**, and **Pseudocoding/Planning**
- Per question, one person is the driver who will lead the following:

Group Exercise

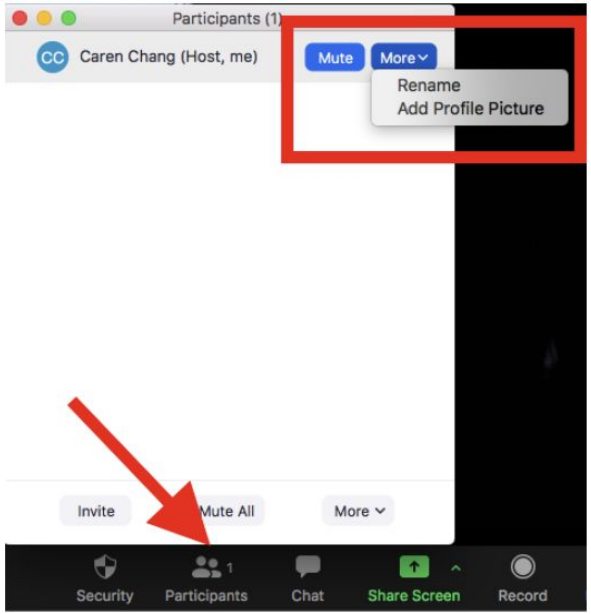
- Groups of 5 - 6: **Understanding**, **Matching**, and **Pseudocoding/Planing**
- Per question, one person is the driver who will lead the following:
  1. List 3 input/output examples
  2. Ask any clarifying question, the rest of the team can answer and decide on the answer (this is arbitrary)
  3. Verbally discuss 1-2 ways to solve the problem and discuss tradeoffs.
  4. Think through helper methods that might help solve the problem
  5. Write the pseudocode

Group Exercise

Reminder: prepend your pod number to your name! Find your pod number here.

So for example, if Caren was in Group 1, she would make her Zoom display name to `1 - Caren`.

You can change your Zoom display name by going to `Participants` -> Find your name and hover over it -> Click on `More` -> `Rename`



Discuss Together

Kth frequent Number

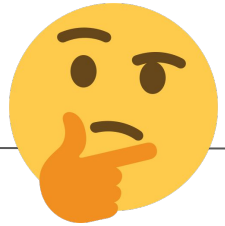
Inputs to try

- 0
- Null
- empty
- input of length 1 (e.g. string, array)
- negative integers
- SUPERRR large input
- duplicate inputs (don’t assume elements are unique!)
- unanticipated symbols / characters
- last digit of input number is 0

Ask yourself...

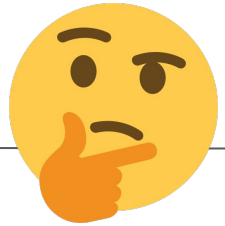
- What happens as input increases?
- Can I achieve this in one pass or two passes through the dataset?
- Can I solve this without creating another data structure? (Can I do it in-place?)
- Am I doing this operation for every element in the dataset?
  - If you are, that's N operations done N times, so $O(N^2)$
- What's the cost incurred of *each item* passing through my function?

Ask yourself...



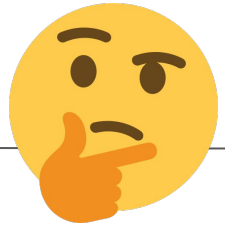
- What happens as input increases?
- Can I achieve this in one pass or two passes through the dataset?
- Can I solve this without creating another data structure? (Can I do it in-place?)
- Am I doing this operation for every element in the dataset?
  - If you are, that's N operations done N times, so $O(N^2)$
- What's the cost incurred of *each item* passing through my function?

Ask yourself...



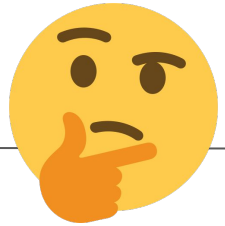
- What happens as input increases?
- Can I achieve this in one pass or two passes through the dataset?
- Can I solve this without creating another data structure? (Can I do it in-place?)
- Am I doing this operation for every element in the dataset?
  - If you are, that's N operations done N times, so $O(N^2)$
- What's the cost incurred of *each item* passing through my function?

Ask yourself...



- What happens as input increases?
- Can I achieve this in one pass or two passes through the dataset?
- Can I solve this without creating another data structure? (Can I do it in-place?)
- Am I doing this operation for every element in the dataset?
  - If you are, that's N operations done N times, so $O(N^2)$
- What's the cost incurred of *each item* passing through my function?

Ask yourself...



- What happens as input increases?
- Can I achieve this in one pass or two passes through the dataset?
- Can I solve this without creating another data structure? (Can I do it in-place?)
- Am I doing this operation for every element in the dataset?
  - If you are, that's N operations done N times, so $O(N^2)$
- What's the cost incurred of *each item* passing through my function?

More Tips

- Look for repeat work.
  - If your current solution goes through the same data multiple times, you might be doing unnecessary repeat work. See if you can save time by looking through the data just once!
- A function with two for-loops *usually* has a runtime of $O(n^2)$

More Tips

- Look for repeat work.
  - If your current solution goes through the same data multiple times, you might be doing unnecessary repeat work. See if you can save time by looking through the data just once!
- A function with two for-loops *usually* has a runtime of $O(n^2)$

More Interview Tips!

Tip:	How To Verbalize It:
💡Repeat the problem statement back to your interviewer.	*“To make sure I understand what you’re asking, you want me to count the number of times a word appears in a sentence and return that value in linear time?”*
💡If you need to assume something, make sure to verbally check if it’s a correct assumption.	*“Can I assume that the input won’t have any negative integers?”*
💡Clearly verbalize your approach to your interviewer before actually writing code to validate that you’re on the right track.	*“Here’s the approach I’m considering [...] do you want me to implement it?”*
💡Make it clear where you are. State what you know, what you’re trying to do, and highlight the gap between the two. The clearer you are in expressing exactly where you’re stuck, the easier it is for your interviewer to help you.	*“So far, I have two pointers traversing the list where one pointer advances faster than the other. However, it’s unclear to me exactly how many steps ahead the faster pointer such that the slower pointer will point to the beginning of the cycle by the time it catches up…”*

Guest Panel on Saturday!

submit questions here

HackerRank Week 2 due by

Monday, Tuesday, June 16th @ 11:59pm PST


Questions?