# CRACKING YOUR TECHNICAL INTERVIEW!

By: Juan Peñuela

Jefferson Li

KNIGHT HACKS

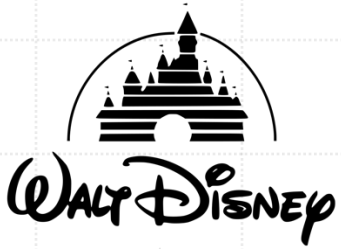
# JUAN PEÑUELA

- Senior in C.S.
- 2x Intern at USAA
- Incoming SWE Analyst at BNY

# JEFFERSON LI

- Third-year CS major
- Previously interned at NYSE | ICE
- Incoming SWE intern at Microsoft
- Fun fact: I speak three languages :)

# OUR INTERVIEW EXPERIENCES



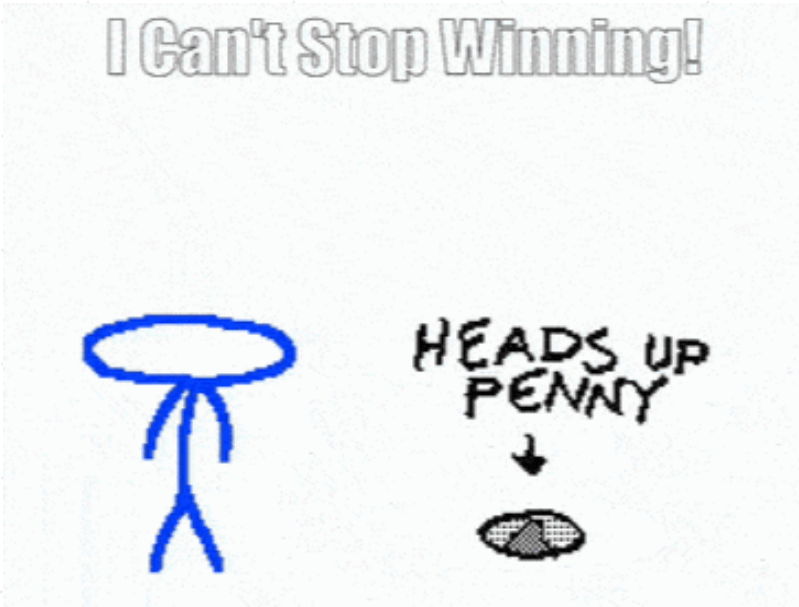






# WHAT YOU WILL LEARN TODAY!

- Why do coding/technical interviews exist?
- How companies pick their interview questions
- Clearing out some SWE Interview myths
- Structure of Technical Interviews
- Tackling the behavioral section of the interview process
- How to pick the best language for the technical part
- How we use runtime (Big Oh) in the real world
- You will be able to solve LeetCode questions!



# WHY DO THEY EXIST?

Problem solving skills are valuable, not just coding skills.

Some interviewers relate basic DSA knowledge with how well you can solve real-world problems at the workplace.

When candidates communicate their thought process to the interviewers, it is seen as a valuable skill.

# HOW DO COMPANIES SELECT THEIR QUESTIONS?

- **The Truth:** The only thing that makes a question "recent" on sites like LeetCode, is the fact that a random interviewer happened to pick it, as there is no set list of questions for a company; rather, ***each interviewer picks their own.***

- ***But… How do Interviewers assess my performance in coding questions?***

# IT'S ALL RELATIVE!

- Interviewers evaluate candidate's performance relative to each other. For this reason, is not a bad thing if you have a hard question, since every other applicant had it too.

- In other words, your performance is not evaluated against the question you were given, but rather, against the average performance of other candidates who had the same question.

# GENERAL STRUCTURE OF A TECHNICAL INTERVIEW

- 45 minutes interview per round
- 2-3 rounds
- Normally leet code easy-medium level questions
- Often the first 15-minutes would be resume/behavioral problems.
- Easy: 10-15 minutes ; Medium: 20-25 minutes

# BEHAVIORAL SECTION

The behavioral portion of a technical interview is just as important as the coding part

Often take place the first 10-15 minutes of the interview

Common questions include:

- Scenario-based question: "Tell me about a time you had a conflict with your teammate"
- Resume-based question: "Can you go more in-depth about your role at NYSE"
- Personal-based question: "What is your biggest weakness, and how do you go about it"

# WHAT SHOULD I DO?

Small talks

Find common grounds: culture, hobbies, sports, or interests

The STAR method

# THE STAR METHOD

**Situation**: During an internship, a teammate struggled with deadlines due to unfamiliarity with key technologies, creating a bottleneck for the team.

**Task**: Resolve the conflict by supporting the teammate's growth while ensuring the team met deadlines.

# THE STAR METHOD

**Action**: Held one-on-one sessions to mentor the teammate and redistributed tasks temporarily to balance the workload.

**Result**: The teammate gained confidence, the team met its deadlines, and the experience strengthened collaboration and problem-solving skills.

I'll start
leetcode 3 weeks
before the interview

I'll start
leetcode 1 week
before the interview

I got some
time tomorrow
before the interview

Unemployed

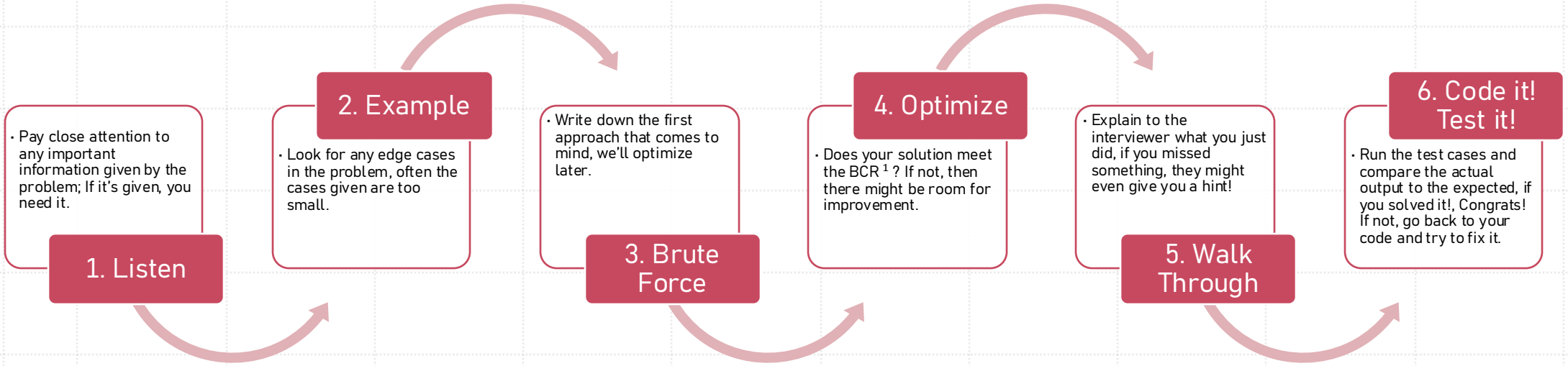
# HOW TO PREPARE FOR CODING INTERVIEWS


NO

# ABOUT PICKING A LANGUAGE

- Well… Most interviewers aren't picky about languages, they're more interested in how well you solve the problems than how well you know a language.
- Prevalence: Choose a relatively common language that you feel comfortable and understand (e.g. Python or Java)
- Avoid languages that will make you do extra work (e.g. C, which forces you to deal with memory allocation)

# STRATEGIES TO SOLVE CODING PROBLEMS




1. BCR (Best conceivable runtime) the theoretical lowest possible time complexity that an algorithm can achieve for solving a given problem. It serves as a benchmark for optimal efficiency.

# EVALUATING THE SPEED OF YOUR CODE

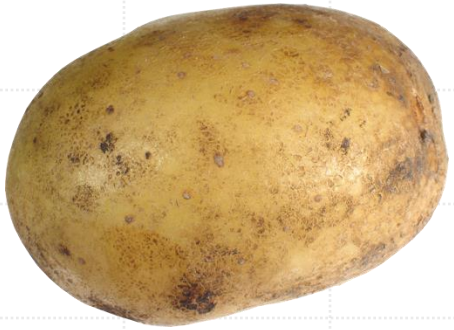
*You*

- Program executes in 0.07ms

```
int foo(int array)
{
   int sum = 0;

   for (int i = 0; i < n; i++)
      sum += array[i];

   return sum;
}
```

*Your friend*

- Program executes in 1.2ms



# THE IDEA OF BIG OH

*Building the Idea*

- We account for the number of steps/instructions executed in a piece of code:

1. Assignment Statements (variables)
2. Comparisons
3. Arithmetic Operations

And then we write our code in terms of a function

```
int foo(int array)
{
   int sum = 0;

   for (int i = 0; i < n; i++)
      sum += array[i];

   return sum;
}
```

| Statement | Execution (where n = array.length) |
|---|---|
| sum = 0 | executed once |
| i = 0 | executed once |
| i < n | executed (n + 1) times (**not** n times!) |
| i++ | executed n times |
| sum += . . . | executed n times |

**Total Runtime:** T(n) = 3n + 3

# LET'S PRACTICE!

*What's the runtime of the following pieces of code?*

1. EASY

```
public static int foo1(int n)
{
   int x = 0;

   for (int i = 0; i <= 100 * n; i++)
      x++;

   return x;
}
```

O(n)

2. EASY

```
public static int foo1a(int n)
{
   int x = 0;

   for (int i = 1; i <= n; i++)
      x++;
   for (int i = 1; i <= n; i++)
      x++;

   return x;
}
```

O(n)

3. EASY

```
public static int foo2(int n)
{
   int x = 0;

   for (int i = 1; i <= n; i++)
      for (int j = 1; j <= n; j++)
         // How many times does the following line get executed?
         x++;

   return x;
}
```

O(n^2)

4. HARD!

```
public static int foo(int n)
{
   int sum = 0;

   for (int i = 0; i < 100; i++)
   {
      // Assume foo3(n) is an O(log n) function.
      sum += foo3(n);

      // Assume foo1(n) is an O(n) function.
      sum += foo1(n);
   }

   return sum;
}
```

O(n)

5. MEDIUM

```
public static int complicatedTripleFoo(int n)
{
   int x = 0;

   for (int i = 0; i <= n; i++)
      for (int j = 0; j <= n; j++)
         for (int k = n; k <= n; k++)
            x++;

   return x;
}
```

O(n^2)

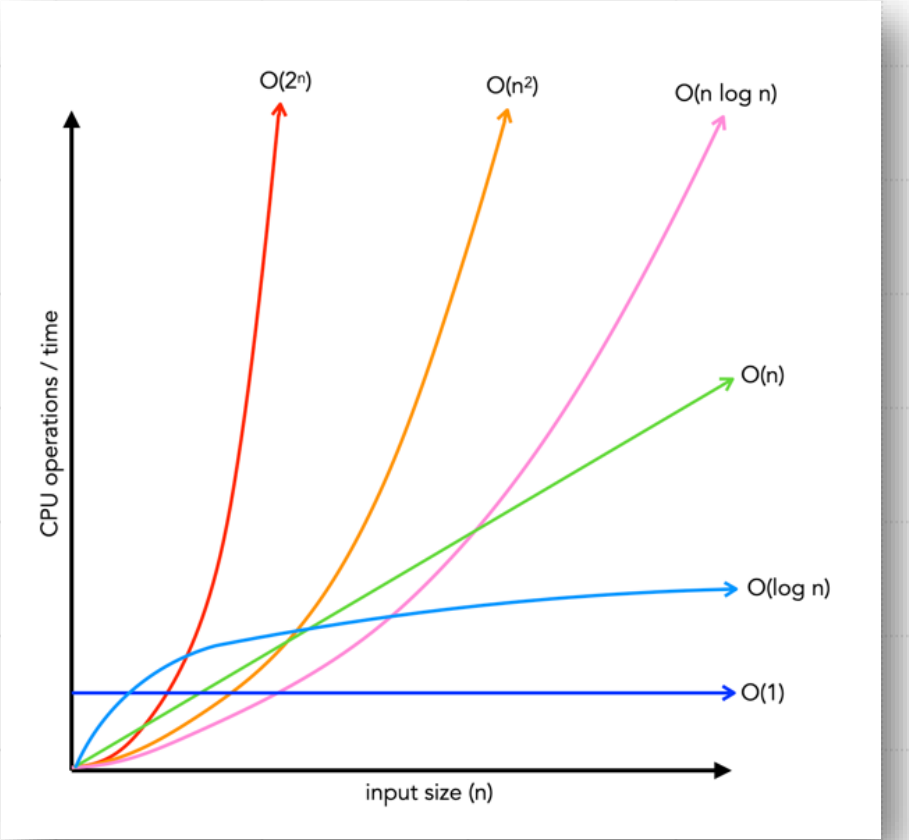
6. HARD!

```
public static int evenMoreFoo(int n)
{
   int sum = 0;

   for (int i = 0; i < n; i++)
   {
      // Assume foo3(n) is an O(log n) method that returns n * n.
      sum += foo3(n);

      // Assume foo1(n) is an O(n) method that returns n * n.
      sum += foo1(n);
   }

   return sum;
}
```

O(n^2)

# COMMON TIME COMPLEXITIES

- **Definition**: Time complexity measures the amount of time an algorithm takes to complete as a function of input size (n).
- **Common Time Complexities**:
- **O(1)**: Constant Time – *Example: Accessing an array element.*
- **O(log n)**: Logarithmic Time – *Example: Binary search.*
- **O(n)**: Linear Time – Example*: Iterating through an array.*
- **O(n²)**: Quadratic Time – *Example: Nested loops.*
- **O(2ⁿ)**: Exponential Time – *Example: Solving the Tower of Hanoi.*



# LET'S TRY SOME INTERVIEW QUESTIONS!!!

| BEGINNER | INTERMEDIATE |
| --- | --- |
| 1. **Is Unique**: Implement a method to determine if a string has all unique characters.<br>***Follow up**: Can you solve it without using additional data structures?* | **1. Zero Matrix**: Given a $M \times N$ matrix, write an algorithm such that if an element [i][j] == 0, then the entire row and column is set to 0. |
| **2. Check Permutation**: Given two strings, write a method to check if string 1 is a permutation of string 2. Example:<br><br>Input: "knightro", "troknight"<br>Output: True | **2. String Compression:** Compress the strings by counting how many of the same character there are in a row. Example:<br><br>Input: "aabcccccaaa"<br>Output: "a2b1c5a3" |

# BEGINNER SOLUTIONS 1: IS UNIQUE

- One Option is to create an array of 256 Booleans (Due to ASCII characters), then iterate through the string, and for each character, mark that index as true. If the flag was already true, then return false.

- If we can't use additional data structures, there's other approaches we can use:
  - 1. Compare each character against each other $O(n^2)$
  - 2. Sort the string and check neighboring characters.

```java
public static boolean isUnique(String str)
{
    // Handle edge cases

    // Create boolean array
    boolean[] charSet = new boolean[256];

    // Iterate through string
    for (int i = 0; i < str.length(); i++)
    {
        int current = str.charAt(i);

        // Character was found earlier
        if (charSet[current])
            return false;
        // Otherwise mark it
        else
            charSet[current] = true;
    }
    // If the code made it here, return true
    return true;
}
```

# INTERMEDIATE SOLUTIONS 1: ZERO MATRIX

- You could simply iterate through the matrix and each time you find a 0, set the whole row and column to 0, but if you do this, you will eventually end up with a matrix full of 0s.
- One way to get around this is to create a secondary matrix, which flags which spaces need to be 0, then proceed from there. This approach also guarantees that we will have a $O(M \times N)$ runtime.
- What's the runtime of this approach?

# BEGINNER SOLUTIONS 2: CHECK PERMUTATION

- The easiest solution is to sort both strings, and if they match, then they are permutations of each other.
- We can also check the character count on each string and see if they match, using a hash map or in python, the function Counter()
- ***Follow up: What's the BCR of this question?***

```java
public static boolean checkPermutation(String str1, String str2)
{
    // If the strings are not the same length, they can't be
    // each other's permutation
    if (str1.length() != str2.length())
        return false;

    // Sort and compare the strings
    char[] charArray1 = str1.toCharArray();
    char[] charArray2 = str2.toCharArray();

    Arrays.sort(charArray1);
    Arrays.sort(charArray2);

    return Arrays.equals(charArray1, charArray2);
}
```

# INTERMEDIATE SOLUTIONS 2: STRING COMPRESSION

- **Option 1**: Check if the current character is the same as the next character. If not, add its compressed version to the result. This works but is not efficient.
- **Option 2**: Using a map to count the number of characters, then building the string based on the keys and values.
- **Option 3**: Using a sliding window approach, where the fast pointer increases while the character remains the same; counting and building the string at the same time.
- What other solutions did you come up with?

# MENTALITY MANAGEMENT

- It's essential to stay calm and collected throughout the interview
- Find common grounds with the interviewer
- Remember, it's a CONVERSATION, not a test
- Keeping a learning attitude
- It's OK to ask for hints
- Rubber duck method

# Questions?

# LET'S CONNECT!

**Juan Peñuela**

Former Software Engineer Intern at USAA &
Computer Science Student at University of Central...

**Jefferson Li**

Incoming SWE intern @ Microsoft | Prev @ ICE
Mortgage Technology | CS Major @ UCF

# REFERENCES


- *McDowell, Gayle Laakmann. Cracking the Coding Interview: 189 Programming Questions and Solutions. 6th ed., CareerCup, 2015.*