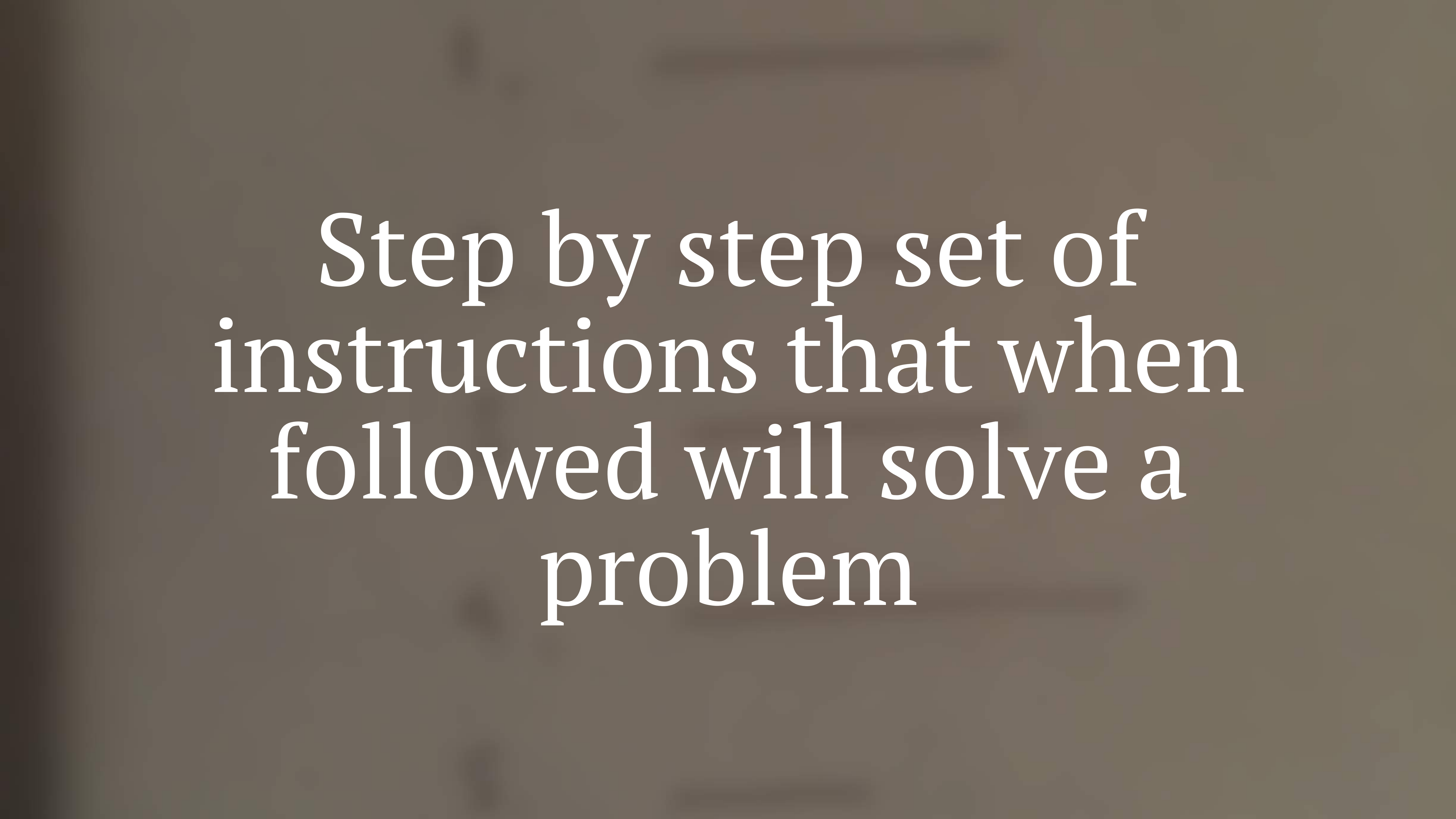
Algorithms and time complexity

What is an algorithm?



Step by step set of instructions that when followed will solve a problem

Find the largest distance between 2 numbers

[2,6,3,19,5,7]

Multiple approaches that take more or less operations

How many operations => Time complexity

How the size of the problem changes the amount of space and time taken to solve the problem using a particular algorithm

Why does it matter?

Computer has limited resources (limited space and limited processing power)

We'll focus on time complexity

Creating algorithms is a process of optimizing

For our challenge - finding the largest distance between 2 numbers

[2,6,3,19,5,7]

As we increase the number of inputs (the 'size of the problem') - how will the time taken change?

Approach 1

Algorithm: Compare all the numbers to each other and return the largest difference

How many operations?

Approach 1

Algorithm: Compare all the numbers to each other and return the largest difference

How many operations?

n*n => n^2

Quadratic time

Approach 2

Algorithm: Find the smallest number, find the largest number and return their difference

How many operations?

Approach 2

Algorithm: Find the smallest number, find the largest number and return their difference

How many operations?

2*5

Linear time

Approach 3

Assume we now know the list is sorted

Algorithm: Return difference between first and last element of list

How many operations?

Approach 3

Assume we now know the list is sorted

Algorithm: Return difference between first and last element of list

How many operations?

3 (always)

Constant time

Approach 1: Compare all the numbers to each other and return the largest difference n^2

Approach 2: Find the smallest number, find the largest number and return their difference 2n

Approach 3: (Assuming sorted list) Return difference between first and last element of list 3

How can we compare these?

Big-O notation describes the performance or complexity of an algorithm in terms of execution time

O(1)
O(n)
O(n^2)

Why not O(3) or O(2n)?

Big-O generalizes

Difference in time complexity matters only at big numbers for n

50n or **2**n^2 when n = 1,000

50 * 1,000 => 50,000

2 * (1,000^2) => 2,000,0**00**

(100 times bigger)

Binary Search - Phone Book

List of 1,000,000 names in alphabetical order

n = 1,000,000

How to find "Will Sentance"?

Logarithmic O(log n)

Summary

1. Constant: O(1) e.g. myArray[35]
2. Logarithmic: O(log n) e.g. binary search
3. Linear: O(n) e.g for loop
4. Quadratic: O(n^2) e.g. for loop inside a for loop
5. Exponential: O(constant^n) e.g. breaking a password by guessing every possible combination of letters