

CSC110 Lecture 25: Worst-Case Running Time Analysis

David Liu, Department of Computer Science

Navigation tip for web slides: press ? to see keyboard navigation controls.

Announcements and Today's Plan

Announcements

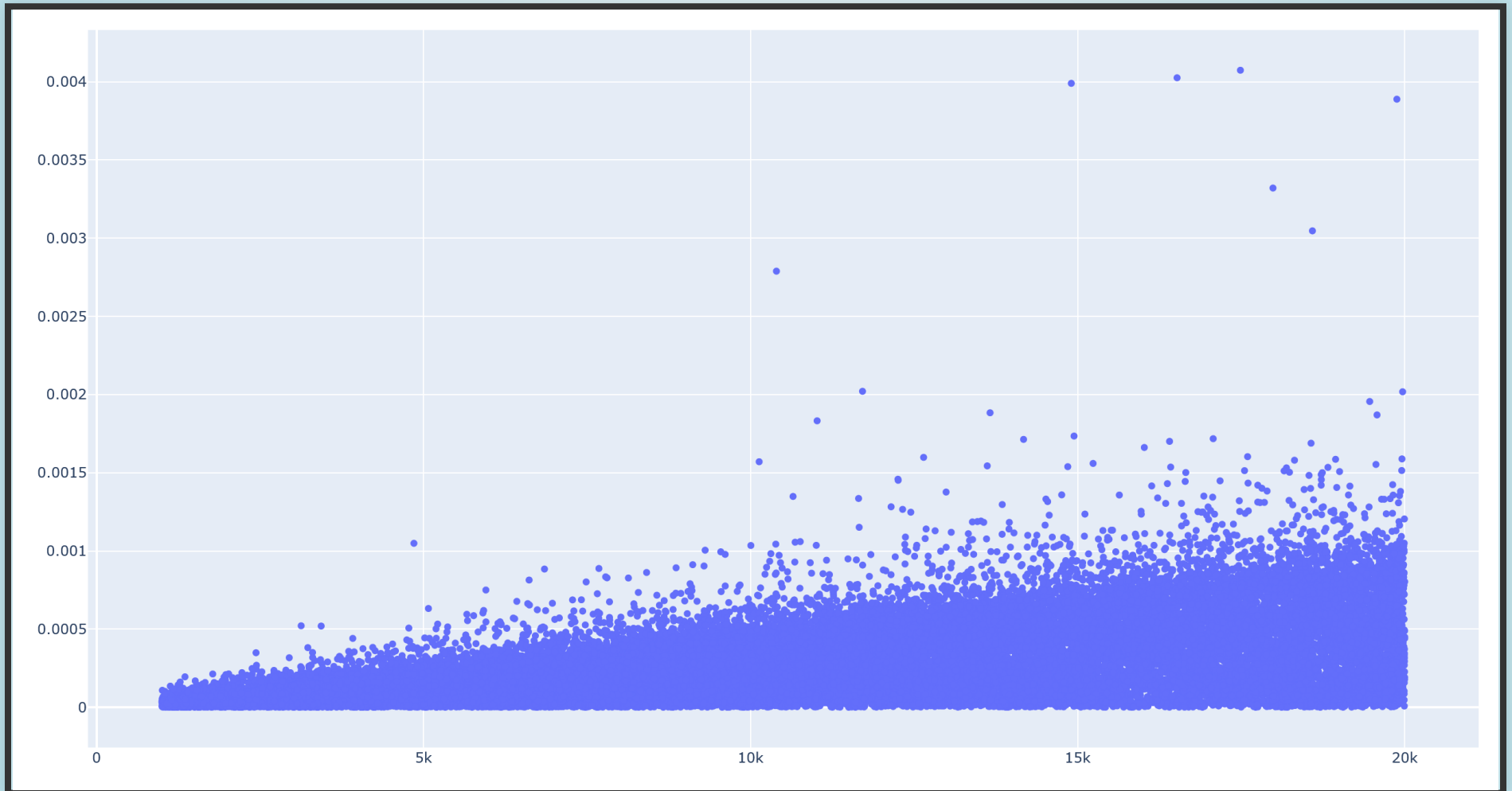
- Assignment 4 has been [posted](#)
 - Check out the [A4 FAQ \(+ corrections\)](#)
 - [Additional TA office hours](#)
 - Review [advice on academic integrity](#)
- Prep 10 (due next Monday) is the [last prep](#) (no preps in Weeks 11/12)
- We'll be posting the Term Test 3 Info Page later today

Today you'll learn to...

- Analyse the **worst-case running time** of an algorithm.
- Identify algorithms for which worst-case analysis is appropriate.
- Identify built-in functions and data type operations for which worst-case analysis is appropriate.

Searching in a list

item in lst



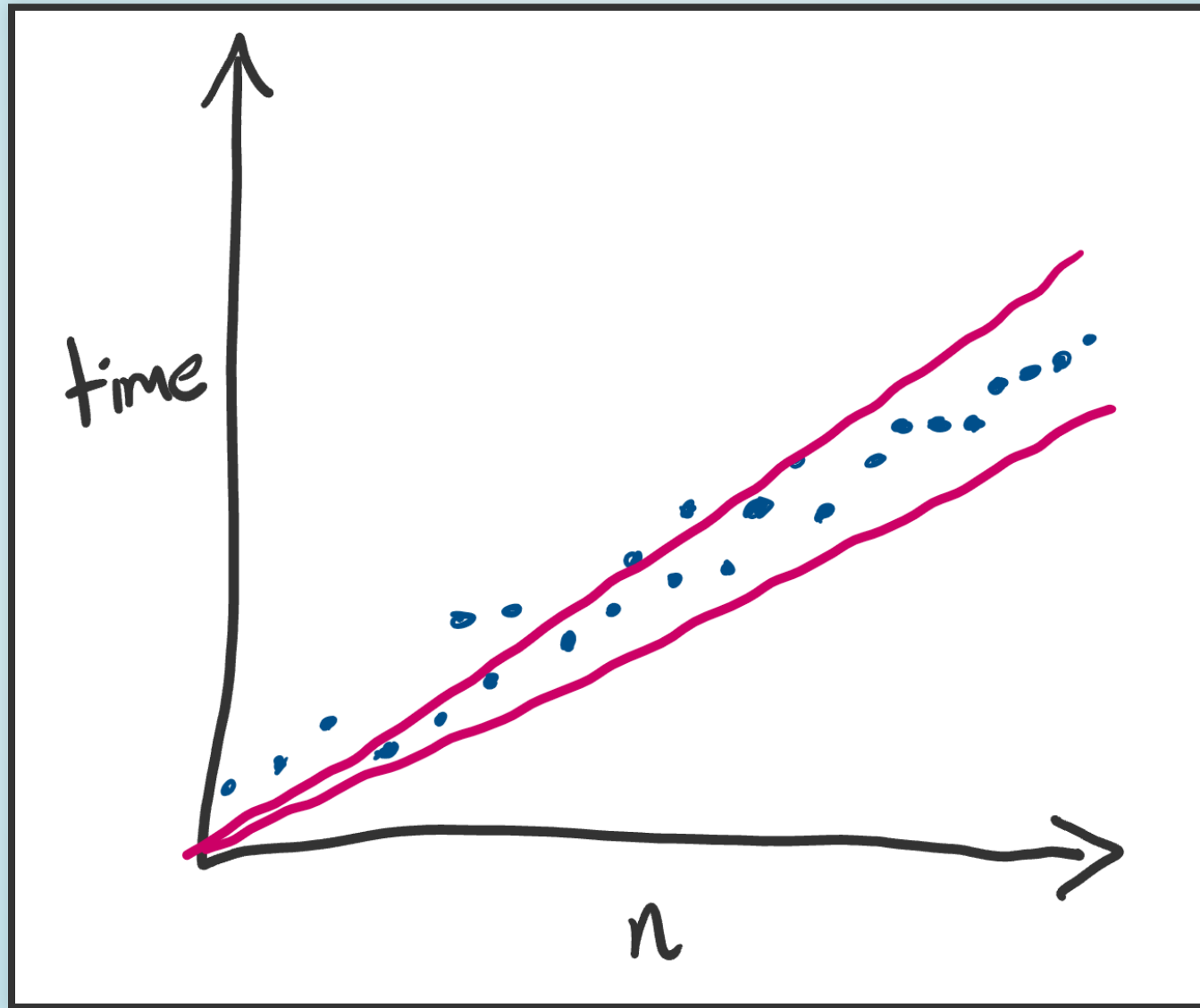
```
def search(lst: list[int], item: int) -> bool:
    """Return whether item is in lst."""
    for x in lst:
        if x == item:
            return True

    return False
```

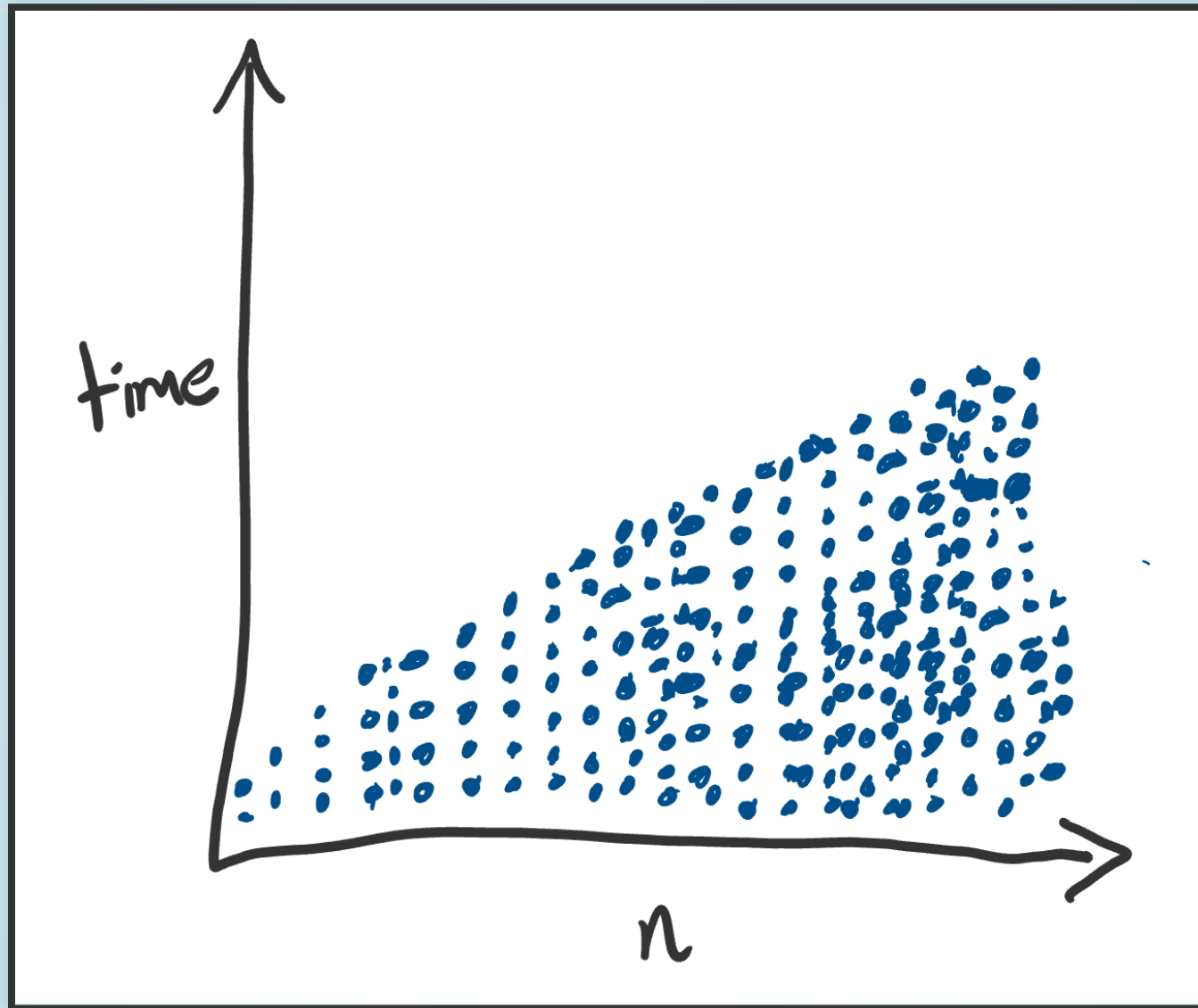
The running time of `search` doesn't just depend on the length of `lst`, it also depends on the elements in `lst`.

Running time is not a **function** of input size: multiple inputs of the same size can have different running times!

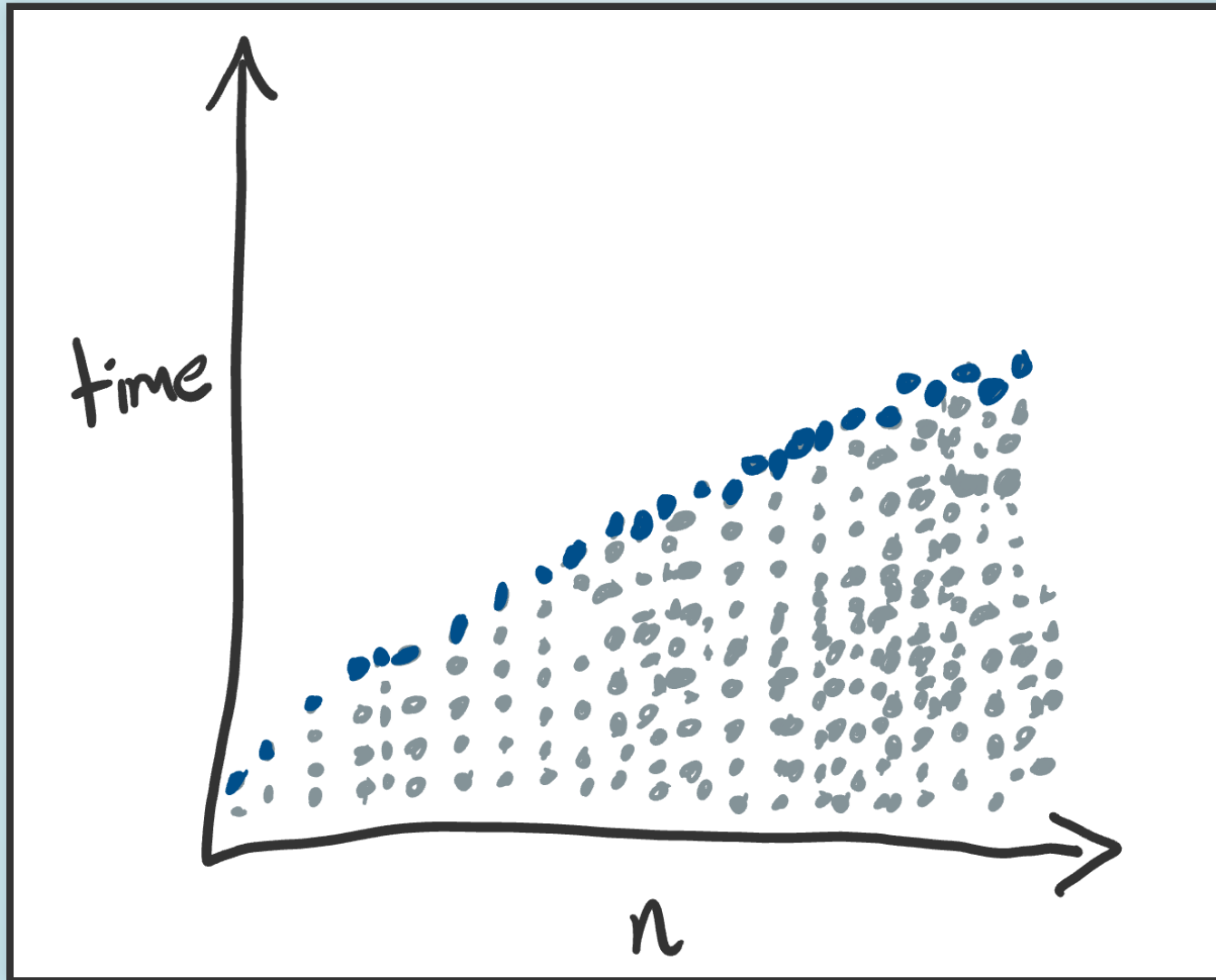
Running times up until now...



Running times of search



Worst-case running time



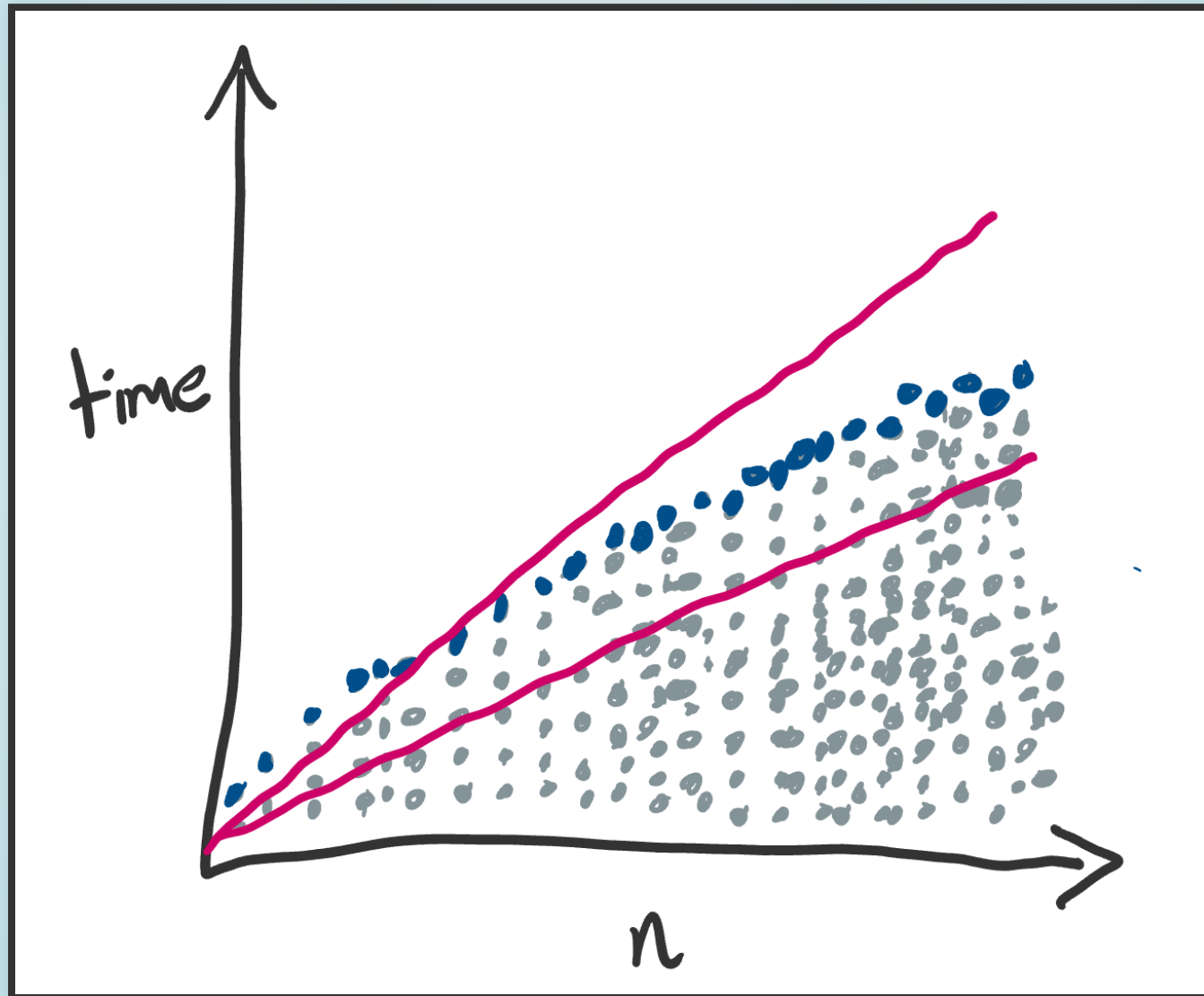
Let `func` be an algorithm, and \mathcal{I}_n be the set of inputs to `func` of size n (where $n \in \mathbb{N}$).

We define the **worst-case running time** of `func` as:

$$WC_{\text{func}}(n) = \max \{ \text{running time of } \mathbf{func}(x) \mid x \in \mathcal{I}_n \}$$

WC_{func} is a function, and so we can use Big-O/Omega/Theta to describe its growth rate!

Worst-case running time



Worst-case running-time analysis

Goal of a **worst-case running-time analysis** of `func`:

*Find an elementary function f such that
 $WC_{\text{func}} \in \Theta(f)$.*

$WC_{\text{func}}(n)$ is the maximum of a set of numbers.

How do we know what this maximum is?

Interlude

What is the **maximum age** of the people in this room?

What is the **maximum height** of the people in this room?

Goal: analyse the worst-case running time of `search`.

```
def search(lst: list[int], item: int) -> bool:
    """Return whether item is in lst."""
    for x in lst:
        if x == item:
            return True

    return False
```

Intuition: the maximum runtime occurs when `item` is not in `lst`. In this case, (roughly) n steps happen, where $n = \text{len}(\text{lst})$.

In a worst-case running-time analysis, we **don't** try to compute WC_{func} exactly, since it is hard in general to find an exact “maximum running time”.

Instead, we find **matching upper and lower bounds** on the running time:

1. Find an elementary function f such that $WC_{\text{func}} \in \mathcal{O}(f)$
2. Then, show that $WC_{\text{func}} \in \Omega(f)$
3. Conclude that $WC_{\text{func}} \in \Theta(f)$

Finding an upper bound on the worst-case running time

f is an **upper bound** on WC_{func} when

- $\forall n \in \mathbb{N}, WC_{\text{func}}(n) \leq f(n)$

i.e.,

- $\forall n \in \mathbb{N}, \max \{\text{running time of } \text{func}(x) \mid x \in \mathcal{I}_n\} \leq f(n)$

i.e.,

- $\forall n \in \mathbb{N}, \forall x \in \mathcal{I}_n, \text{running time of } \text{func}(x) \leq f(n)$

Finding an upper bound on the worst-case running time

$$\forall n \in \mathbb{N}, \forall x \in \mathcal{I}_n, \text{ running time of } \mathbf{func}(x) \leq f(n)$$

To find an upper bound on the worst-case running time of `func`, we:

- Pick an **arbitrary** n
- Pick an **arbitrary** input x of size n
- Find an upper bound on the running time of `func` (x) .

```
def search(lst: list[int], item: int) -> bool:
    """Return whether item is in lst."""
    for x in lst:
        if x == item:
            return True

    return False
```

Worst-case analysis (upper bound). Let $n \in \mathbb{N}$, and let `lst` be an **arbitrary** list of length n , and let `item` be an arbitrary `int`.

The for loop takes **at most** n iterations, and each iteration takes 1 step (constant time), for a total of **at most** n steps.

The `return False` either happens or doesn't; it takes **at most** 1 step.

The total running time is **at most** $n + 1$ steps, which is $\mathcal{O}(n)$.

Finding a lower bound on the worst-case running time

f is a **lower bound** on WC_{func} when

- $\forall n \in \mathbb{N}, WC_{\text{func}}(n) \geq f(n)$

i.e.,

- $\forall n \in \mathbb{N}, \max \{\text{running time of } \text{func}(x) \mid x \in \mathcal{I}_n\} \geq f(n)$

i.e.,

- $\forall n \in \mathbb{N}, \exists x \in \mathcal{I}_n, \text{running time of } \text{func}(x) \geq f(n)$

Finding a lower bound on the worst-case running time

$$\forall n \in \mathbb{N}, \exists x \in \mathcal{I}_n, \text{ running time of } \text{func}(x) \geq f(n)$$

To find a **lower bound on the worst-case running time of `func`**, we:

- Pick an **arbitrary** n
- Pick a **specific** input x of size n
- Find a lower bound on the running time of `func` (x).
 - Or, usually we can find an **exact** running time of `func` (x).

```
def search(lst: list[int], item: int) -> bool:
    """Return whether item is in lst."""
    for x in lst:
        if x == item:
            return True

    return False
```

Worst-case analysis (lower bound).

Let $n \in \mathbb{N}$. Let `lst = [1, 2, ..., n]` and `item = 0`.

The for loop takes n iterations (the if condition is never `True`). Each iteration takes 1 step, for a total of n steps.

The `return False` executes and takes 1 step.

The total running time is $n + 1$ steps, which is $\Theta(n)$.

Putting it together

First, we proved that $WC_{func} \in \mathcal{O}(n)$.

Second, we found an **input family** (set of inputs, one for each $n \in \mathbb{N}$) whose running time is $\Theta(n)$. This told us that $WC_{func} \in \Omega(n)$.

Putting these two parts together, we can conclude that $WC_{func} \in \Theta(n)$.

Warning meme



Exercise 1: Worst-case running time analysis practice

Exercise 2: Lists vs. sets!

any and all revisited (birefly)

`any` and `all` are implemented using **early returns**:

- `any` can stop as soon as it encounters a `True`
- `all` can stop as soon as it encounters a `False`

Their worst-case running time is $\Theta(n)$, where n is the size of the input collection.

Demo: `any` and `all` with comprehensions

See Course Notes for details!

A trickier worst-case analysis

Definitions

A **palindrome** is a string that is the same when reversed.

- E.g., 'abba', 'davad', 'b'.

A **prefix** of a string s is a string that appears at the beginning of s .

- E.g., 'abc' is a prefix of 'abcdefgh'.

A **palindrome prefix** of a string s is a prefix of s that is a palindrome.

- E.g., 'abba' is a palindrome prefix of 'abbaceb'.

Problem: given a string *s*, return the length of the longest palindrome prefix of *s*.

- E.g., given 'abbaceb', return 4.

```
def palindrome_prefix(s: str) -> int:
    n = len(s)
    for prefix_length in range(n, 0, -1): # goes from n down to
        # Check whether s[0:prefix_length] is a palindrome
        is_palindrome = ...

        # If a palindrome prefix is found, return the current len
    if is_palindrome:
        return prefix_length
```

```
def palindrome_prefix(s: str) -> int:
    n = len(s)
    for prefix_length in range(n, 0, -1): # goes from n down to
        # Check whether s[0:prefix_length] is a palindrome
        is_palindrome = all(s[i] == s[prefix_length - 1 - i]
                             for i in range(0, prefix_length))

        # If a palindrome prefix is found, return the current len
        if is_palindrome:
            return prefix_length
```

We can show that the worst-case running time is $\mathcal{O}(n^2)$, where n is the length of s . (Exercise!)

To prove a matching lower bound, we need to find an input family whose runtime is $\Theta(n^2)$.

Finding a “maximum” input family

Let $n \in \mathbb{N}$.

1. Attempt 1: Let $s = 'aaa \dots a'$ repeated n times.

- `all` call takes n steps, and then returns `True`
- The for loop only iterates once!
- $\Theta(n)$ running time

2. Attempt 2: Let $s = abcabcabc \dots$ (abc repeated for n characters).

- The for loop iterates n times (no prefix is a palindrome).
- But the `all` call only takes 1 or 2 steps before returning `False`.
- So again, $\Theta(n)$ running time!

See Course Notes for a discussion of a “good enough” input family!

Summary

Today you learned to...

- Analyse the worst-case running time of an algorithm.
- Identify algorithms for which worst-case analysis is appropriate.
- Identify built-in functions and data type operations for which worst-case analysis is appropriate.

Homework

- Readings:
 - From today: 9.8
 - Next week: Chapter 10
- Assignment 4 due next week!
- Prep 10 to be released shortly
 - Prep 10 is the **last prep** (no preps in Weeks 11/12)
- Term Test 3 info page to be released shortly

Bonus warning meme

**STUDENT WHO DIDN'T PAY ATTENTION TODAY:
CAN'T WE JUST TELL WHAT THE WORST CASE IS?
DAVID:**

