Algorithms and Datastructures

Big-Oh notation
September 5, 2023

iCIS | Software Science
Radboud University

Runtime Complexity

Big-Oh Notation

Analyzing the runtime complexity of programs

Question

- Why is quick sort better than bubble sort?

Question

- Why is quick sort better than bubble sort?
- Because quick sort runs faster.

Question

- Why is quick sort better than bubble sort?
- Because quick sort runs faster.
- More difficult: is quick sort better than merge sort?
- General question: what measures do we have to determine how good an algorithm is?

Good algorithms

- Functional correctness: is the output of the algorithm correct?
- Runtime efficiency: how long does execution of the algorithm take?
- Space efficiency: how much memory does the algorithm use?
- Energy efficiency: how much energy does the algorithm consume?

Good algorithms

- Functional correctness: is the output of the algorithm correct?
- **Runtime efficiency**: how long does execution of the algorithm take?
- Space efficiency: how much memory does the algorithm use?
- Energy efficiency: how much energy does the algorithm consume?

Topic of today: analyzing the runtime efficiency of algorithms.

Outline

Runtime Complexity

Big-Oh Notation

Analyzing the runtime complexity of programs

Analyzing Execution Time

Before we can analyze algorithms, we need to fix a formal setting.

- Model of computation
- Resource cost of operation

Analyzing Execution Time: Model of Computation

We use **random access machines**.

- Generic computer with single processor and no concurrency
- Algorithms are its programs

There are primitive operations

- Arithmetic operations: addition, multiplication, ...
- Data manipulation: storing, reading, ...
- Control flow: if-then-else, loops, ...

The primitive operations operate in constant time

Execution Time

The execution time depends on the **size** of the input

- For sorting an array: the number of elements
- Finding maximal common subsequence of two strings: sum of the lengths of the strings

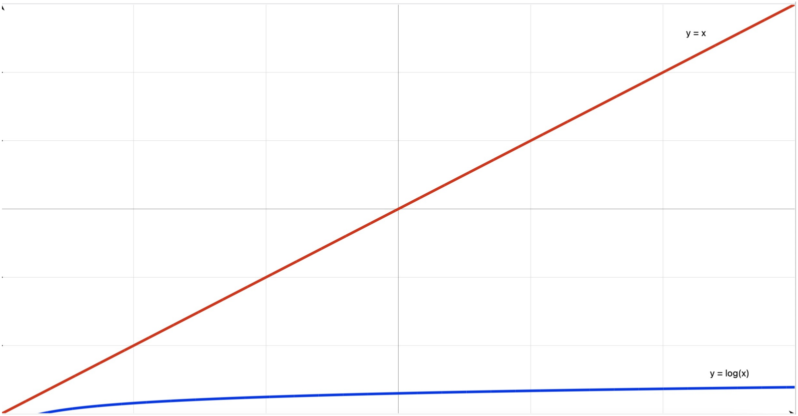
Execution time: number of primitive operations executed (depends on the size of the input).
Note: this is machine independent

Let's compare some runtime complexities

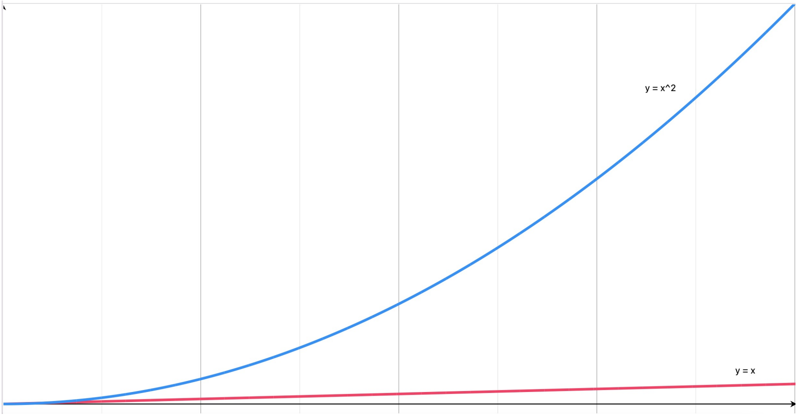
Let's compare some algorithms with

- $T(n) = \log(n)$ (binary search)
- $T(n) = n$ (linear search)
- $T(n) = n^2$ (bubble sort)
- $T(n) = 2^n$ (naive implementation of Fibonacci)

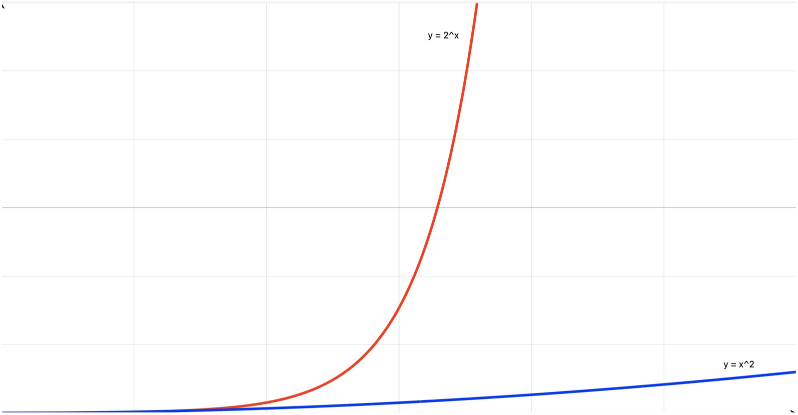
Comparison between runtime complexities



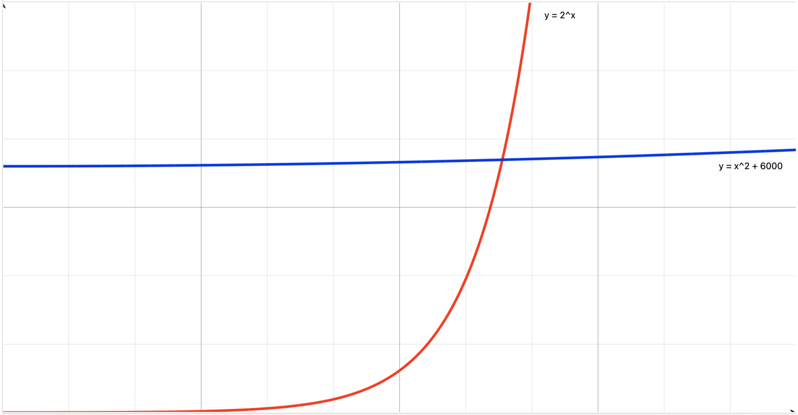
Comparison between runtime complexities



Comparison between runtime complexities



Comparison between runtime complexities



Comparison between runtime complexities

	$\log(n)$	n	$n \cdot \log(n)$	n^2	2^n
10	≈ 3.3	10	≈ 33.2	100	1024
100	≈ 6.6	100	≈ 664.4	10000	$\approx 1.27 \cdot 10^{30}$
1000	≈ 9.9	1000	≈ 9965.8	1000000	really a lot

Think

- Binary search: $T(n) = \log(n)$
- Linear search: $T(n) = n$
- Merge sort: $T(n) = n \cdot \log(n)$
- Bubble sort: $T(n) = n^2$
- Fibonacci (naive implementation): $T(n) = 2^n$

It is about growth

- In practice: the input can be quite large (big databases)
- In those cases, it is the **growth** that determines the runtime efficiency, not the constants

So: we want to have a methods to compare functions based on how they grow if the input becomes larger and larger.

Outline

Runtime Complexity

Big-Oh Notation

Analyzing the runtime complexity of programs

Definition

Definition (Big Oh)
Suppose, we have $f, g : \mathbb{N} \to \mathbb{R}^+$. Then we say $f \in \mathcal{O}(g)$ if there is $c \in \mathbb{R}$ and $n_0 \in \mathbb{N}$ such that $c > 0$ and for all $n \geq n_0$ we have $f(n) \leq c \cdot g(n)$.

This means: the growth of f is bounded by g.
We also say g is an **asymptotic upper bound** for f.

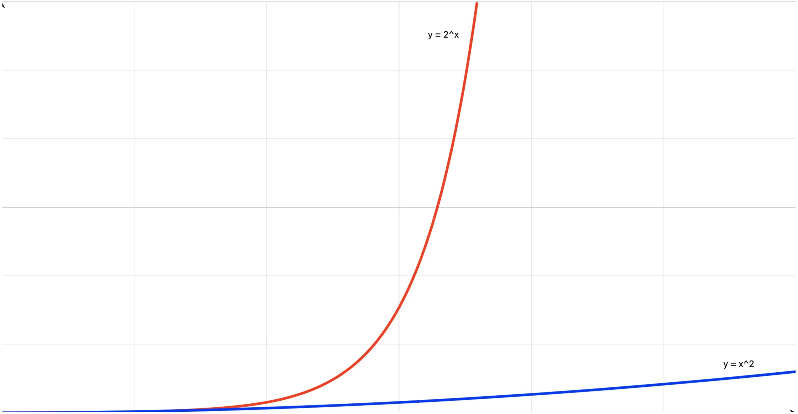
Definition

Definition (Big Oh)
Suppose, we have $f, g : \mathbb{N} \to \mathbb{R}^+$. Then we say $f \in \mathcal{O}(g)$ if there is $c \in \mathbb{R}$ and $n_0 \in \mathbb{N}$ such that $c > 0$ and for all $n \geq n_0$ we have $f(n) \leq c \cdot g(n)$.

This means: the growth of f is bounded by g.
We also say g is an **asymptotic upper bound** for f.
Note: some people write $f = \mathcal{O}(g)$, but I won't do that

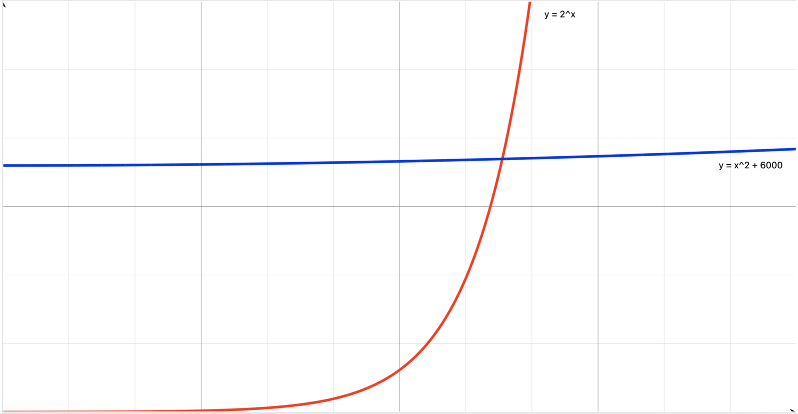
Example

$n^2 \in \mathcal{O}(2^n)$



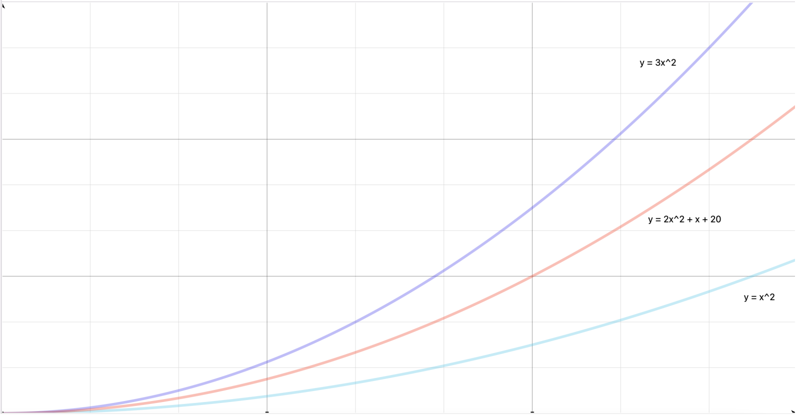
Example

$n^2 + 6000 \in \mathcal{O}(2^n)$



Example

$2 \cdot n^2 + n + 20 \in \mathcal{O}(n^2)$



Example

We have:

$$2 \cdot n^2 + 3 \in \mathcal{O}(n^2)$$

Take:

$$c = 3, \qquad n_0 = 3$$

Then for $n \geq 3$ we have

$$\begin{aligned} 2 \cdot n^2 + 3 &\leq 2 \cdot n^2 + n^2 \\ &= 3 \cdot n^2 \end{aligned}$$

Example

We have:

$$2 \cdot n^2 + 3 \in \mathcal{O}(n^2)$$

Take:

$$c = 3, \qquad n_0 = 3$$

Then for $n \geq 3$ we have

$$\begin{aligned} 2 \cdot n^2 + 3 &\leq 2 \cdot n^2 + n^2 \\ &= 3 \cdot n^2 \end{aligned}$$

Observation: big Oh allows us to **suppress constant factors.**

Example

We have:

$$n^2 + n \in \mathcal{O}(n^2)$$

Take:

$$c = 2, \qquad n_0 = 1$$

Then for $n \geq 1$ we have

$$\begin{aligned} n^2 + n &\leq n^2 + n^2 \\ &= 2 \cdot n^2 \end{aligned}$$

Example

We have:

$$n^2 + n \in \mathcal{O}(n^2)$$

Take:

$$c = 2, \qquad n_0 = 1$$

Then for $n \geq 1$ we have

$$\begin{aligned} n^2 + n &\leq n^2 + n^2 \\ &= 2 \cdot n^2 \end{aligned}$$

Observation: big Oh allows us to **suppress lower-order terms.**

Example

We have:

$$n^2 + 2 \cdot n + 3 \in \mathcal{O}(n^2)$$

Take:

$$c = 4, \qquad n_0 = 3$$

Then for $n \geq 3$, we have

$$\begin{aligned} n^2 + 2 \cdot n + 3 &\leq n^2 + 2 \cdot n^2 + n^2 \\ &= 4 \cdot n^2 \end{aligned}$$

Example

We have:

$$n^2 + 2 \cdot n + 3 \in \mathcal{O}(n^2)$$

Take:

$$c = 4, \qquad n_0 = 3$$

Then for $n \geq 3$, we have

$$\begin{aligned} n^2 + 2 \cdot n + 3 &\leq n^2 + 2 \cdot n^2 + n^2 \\ &= 4 \cdot n^2 \end{aligned}$$

Observation: big Oh allows us to **suppress constants and lower-order terms**.

Nonexample

We have

$$n^2 \notin \mathcal{O}(n)$$

Suppose, we have c and n_0 such that for all $n \geq n_0$

$$c \cdot n \geq n^2$$

Take

$$n = c + n_0 + 1$$

Then $n \geq n_0$. We also have

$$\begin{aligned} n^2 &= (c + n_0 + 1)^2 \\ &= c^2 + 2 \cdot c \cdot n_0 + n_0^2 + 2 \cdot (n_0 + c) + 1 \\ &> c^2 + c \cdot n_0 + c \\ &= c \cdot (c + n_0 + 1) = c \cdot n \end{aligned}$$

Contradiction!

Properties

General properties:

- $f \in \mathcal{O}(f)$
- If $f \in \mathcal{O}(g)$ and $g \in \mathcal{O}(h)$, then $f \in \mathcal{O}(h)$

Properties

General properties:

- $f \in \mathcal{O}(f)$
- If $f \in \mathcal{O}(g)$ and $g \in \mathcal{O}(h)$, then $f \in \mathcal{O}(h)$
- If $f \in \mathcal{O}(g)$, then $c \cdot f \in \mathcal{O}(g)$

Properties

General properties:

- $f \in \mathcal{O}(f)$
- If $f \in \mathcal{O}(g)$ and $g \in \mathcal{O}(h)$, then $f \in \mathcal{O}(h)$
- If $f \in \mathcal{O}(g)$, then $c \cdot f \in \mathcal{O}(g)$
- If $f_1, f_2 \in \mathcal{O}(g)$, then $f_1 + f_2 \in \mathcal{O}(g)$

Properties

General properties:

- $f \in \mathcal{O}(f)$
- If $f \in \mathcal{O}(g)$ and $g \in \mathcal{O}(h)$, then $f \in \mathcal{O}(h)$
- If $f \in \mathcal{O}(g)$, then $c \cdot f \in \mathcal{O}(g)$
- If $f_1, f_2 \in \mathcal{O}(g)$, then $f_1 + f_2 \in \mathcal{O}(g)$

Examples:

- $n \in \mathcal{O}(n^2)$
- If $k_1 \leq k_2$, then $n^{k_1} \in \mathcal{O}(n^{k_2})$

Properties

General properties:

- $f \in \mathcal{O}(f)$
- If $f \in \mathcal{O}(g)$ and $g \in \mathcal{O}(h)$, then $f \in \mathcal{O}(h)$
- If $f \in \mathcal{O}(g)$, then $c \cdot f \in \mathcal{O}(g)$
- If $f_1, f_2 \in \mathcal{O}(g)$, then $f_1 + f_2 \in \mathcal{O}(g)$

Examples:

- $n \in \mathcal{O}(n^2)$
- If $k_1 \leq k_2$, then $n^{k_1} \in \mathcal{O}(n^{k_2})$
- $\log(n) \in \mathcal{O}(n)$
- $n^k \in \mathcal{O}(2^n)$

Properties

General properties:

- $f \in \mathcal{O}(f)$
- If $f \in \mathcal{O}(g)$ and $g \in \mathcal{O}(h)$, then $f \in \mathcal{O}(h)$
- If $f \in \mathcal{O}(g)$, then $c \cdot f \in \mathcal{O}(g)$
- If $f_1, f_2 \in \mathcal{O}(g)$, then $f_1 + f_2 \in \mathcal{O}(g)$

Examples:

- $n \in \mathcal{O}(n^2)$
- If $k_1 \leq k_2$, then $n^{k_1} \in \mathcal{O}(n^{k_2})$
- $\log(n) \in \mathcal{O}(n)$
- $n^k \in \mathcal{O}(2^n)$
- $n \in \mathcal{O}(n \cdot \log(n))$
- $n \cdot \log(n) \in \mathcal{O}(n^2)$

Example

Show that

$$n^2 + 3 \cdot n + 5 \in \mathcal{O}(n^2)$$

Note that

- $n^2 \in \mathcal{O}(n^2)$
- $n \in \mathcal{O}(n^2)$
- So: $3 \cdot n \in \mathcal{O}(n^2)$
- So: $n^2 + 3 \cdot n \in \mathcal{O}(n^2)$

Example

Show that

$$n^2 + 3 \cdot n + 5 \in \mathcal{O}(n^2)$$

Note that

- $n^2 \in \mathcal{O}(n^2)$
- $n \in \mathcal{O}(n^2)$
- So: $3 \cdot n \in \mathcal{O}(n^2)$
- So: $n^2 + 3 \cdot n \in \mathcal{O}(n^2)$
- $5 \in \mathcal{O}(n^2)$
- So: $n^2 + 3 \cdot n + 5 \in \mathcal{O}(n^2)$

Other Notation

Definition
Suppose, we have $f, g : \mathbb{N} \to \mathbb{R}^+$. Then we say $f \in \Omega(g)$ if there is $c \in \mathbb{R}$ and $n_0 \in \mathbb{N}$ such that $c > 0$ and for all $n \geq n_0$ we have $f(n) \geq c \cdot g(n)$.

Proposition
We have $f \in \Omega(g)$ if and only if $g \in \mathcal{O}(f)$.

Other Notation

Definition
Suppose, we have $f, g : \mathbb{N} \to \mathbb{R}^+$. Then we say $f \in \Omega(g)$ if there is $c \in \mathbb{R}$ and $n_0 \in \mathbb{N}$ such that $c > 0$ and for all $n \geq n_0$ we have $f(n) \geq c \cdot g(n)$.

Proposition
We have $f \in \Omega(g)$ if and only if $g \in \mathcal{O}(f)$.

So: this says that f grows at least as fast as g.
We also say g is an **asymptotic lower bound** for f.

Other Notation

Definition
Suppose, we have $f, g : \mathbb{N} \to \mathbb{R}^+$. Then we say $f \in \Omega(g)$ if there is $c \in \mathbb{R}$ and $n_0 \in \mathbb{N}$ such that $c > 0$ and for all $n \geq n_0$ we have $f(n) \geq c \cdot g(n)$.

Proposition
We have $f \in \Omega(g)$ if and only if $g \in \mathcal{O}(f)$.

So: this says that f grows at least as fast as g.
We also say g is an **asymptotic lower bound** for f.

Definition
We say $f \in \Theta(g)$ if $f \in \mathcal{O}(g)$ and $f \in \Omega(g)$.

True or false?

$5 \cdot n \in \Omega(n)$

True or false?

$5 \cdot n \in \Omega(n)$

True! We have $n \in \mathcal{O}(5 \cdot n)$, so we also have $5 \cdot n \in \Omega(n)$

True or false?

$5 \cdot n \in \Omega(n)$

True! We have $n \in \mathcal{O}(5 \cdot n)$, so we also have $5 \cdot n \in \Omega(n)$

$n \in \Omega(n^2)$

True or false?

$5 \cdot n \in \Omega(n)$

True! We have $n \in \mathcal{O}(5 \cdot n)$, so we also have $5 \cdot n \in \Omega(n)$

$n \in \Omega(n^2)$

False! This means that $n^2 \in \mathcal{O}(n)$, which is false.

True or false?

$5 \cdot n \in \Omega(n)$

True! We have $n \in \mathcal{O}(5 \cdot n)$, so we also have $5 \cdot n \in \Omega(n)$

$n \in \Omega(n^2)$

False! This means that $n^2 \in \mathcal{O}(n)$, which is false.

If $f \in \Theta(g)$, then $g \in \Theta(f)$

True or false?

$5 \cdot n \in \Omega(n)$

True! We have $n \in \mathcal{O}(5 \cdot n)$, so we also have $5 \cdot n \in \Omega(n)$

$n \in \Omega(n^2)$

False! This means that $n^2 \in \mathcal{O}(n)$, which is false.

If $f \in \Theta(g)$, then $g \in \Theta(f)$

True! Note that both $f \in \Theta(g)$ and $g \in \Theta(f)$ mean that $f \in \mathcal{O}(g)$ and $g \in \mathcal{O}(f)$.

True or false?

$5 \cdot n \in \Omega(n)$

True! We have $n \in \mathcal{O}(5 \cdot n)$, so we also have $5 \cdot n \in \Omega(n)$

$n \in \Omega(n^2)$

False! This means that $n^2 \in \mathcal{O}(n)$, which is false.

If $f \in \Theta(g)$, then $g \in \Theta(f)$

True! Note that both $f \in \Theta(g)$ and $g \in \Theta(f)$ mean that $f \in \mathcal{O}(g)$ and $g \in \mathcal{O}(f)$.

$n^2 + 2 \cdot n + 20 \in \Theta(n^2)$

True or false?

$5 \cdot n \in \Omega(n)$

True! We have $n \in \mathcal{O}(5 \cdot n)$, so we also have $5 \cdot n \in \Omega(n)$

$n \in \Omega(n^2)$

False! This means that $n^2 \in \mathcal{O}(n)$, which is false.

If $f \in \Theta(g)$, then $g \in \Theta(f)$

True! Note that both $f \in \Theta(g)$ and $g \in \Theta(f)$ mean that $f \in \mathcal{O}(g)$ and $g \in \mathcal{O}(f)$.

$n^2 + 2 \cdot n + 20 \in \Theta(n^2)$

True!

Outline

Runtime Complexity

Big-Oh Notation

Analyzing the runtime complexity of programs

Linear Search

We consider the following program

```
bool isIn(int xs[], int x) {
    int i = 0;
    while (i < xs.length()) {
        if(x == xs[i]) {
            return true;
        }
        i++;
    }
    return false;
}
```

Denote its time complexity **in the worst case** by T.
Goal: $T \in \mathcal{O}(n)$.

Linear Search with Annotated Costs

Step 1: annotate the cost per line

```
bool isIn(int xs[], int x) {
    int i = 0;                    // Cost: c_1
    while (i < xs.length()) {     // Cost: c_2
        if(x == xs[i]) {          // Cost: c_3
            return true;          // Cost: c_4
        }
        i++;                      // Cost: c_5
    }
    return false;                 // Cost: c_6
}
```

Linear Search with Annotated Costs and Repetitions

Step 2: annotate the number of repetitions per line

```
bool isIn(int xs[], int x) {
    int i = 0;                        // Cost: c_1, repetitions: 1
    while (i < xs.length()) {        // Cost: c_2, repetitions: n
        if(x == xs[i]) {             // Cost: c_3, repetitions: n
            return true;             // Cost: c_4, repetitions: n
        }
        i++;                         // Cost: c_5, repetitions: n
    }
    return false;                    // Cost: c_6, repetitions: 1
}
```

Here *n* is the length of the array xs.
Note: in the worst case (the element is not found), the loop has *n* repetitions.

The Total Cost of Linear Search

Step 3: add all the costs and take repetitions into account

$$T(n) = c_1 + n \cdot (c_2 + c_3 + c_4 + c_5) + c_6$$

The Total Cost of Linear Search

Step 3: add all the costs and take repetitions into account

$$T(n) = c_1 + n \cdot (c_2 + c_3 + c_4 + c_5) + c_6$$

Note: in the best case, the element we are looking for, is the first element of the array.
In that case, we would have

$$T(n) = c_1 + c_2 + c_3 + c_4$$

The Time Complexity of Linear Search

Step 4: prove what we wanted to show

$$c_1 + n \cdot (c_2 + c_3 + c_4 + c_5) + c_6 \in \mathcal{O}(n)$$

Simpler, show that

$$a \cdot n + b \in \mathcal{O}(n)$$

This holds.

- Take $c = a + 1$ and $n_0 = b$.
- Or you can use the properties given before

Why is the worst case interesting?

- It gives an upper bound that holds for **every** input
- It is realistic and often the average case is closer to the worst case
- Worst case is simpler than the average case

Average case: difficult to determine. For example, one needs to determine the probability that an input is already sorted

Conclusion

Take aways from this lecture:

- Many factors determine the quality of algorithms, including **run-time efficiency**
- We compare run-time efficiency by looking at the **growth** of functions
- To characterize the growth of functions, we use the **Big-Oh notation**
- The Big-Oh notation allows us to **surpress constants and lower-order terms**
- Often the **worst case** is the most interesting case

Reading material: Chapter 1 and 2 in Roughgarden