CSC110 Lecture 22: Properties of Asymptotic Notation and Basic 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
  - But please take some time off 😴
- Next week is **reading week**!
  - No lecture, tutorial, or office hours
- Note: there **is** a tutorial tomorrow

Definitions from last class

Let $f, g : \mathbb{N} \to \mathbb{R}^{\geq 0}$.

$$g \in \mathcal{O}(f) : \exists c, n_0 \in \mathbb{R}^+,\ \forall n \in \mathbb{N},\ n \geq n_0 \Rightarrow g(n) \leq c \cdot f(n)$$
$$g \in \Omega(f) : \exists c, n_0 \in \mathbb{R}^+,\ \forall n \in \mathbb{N},\ n \geq n_0 \Rightarrow g(n) \geq c \cdot f(n)$$
$$g \in \Theta(f) : \exists c_1, c_2, n_0 \in \mathbb{R}^+,\ \forall n \in \mathbb{N},\ n \geq n_0 \Rightarrow c_1 \cdot f(n) \leq g(n) \leq c_2 \cdot f(n)$$

Today you'll learn to...

1. Compare different elementary functions using asymptotic notation.
2. State and prove useful properties about Big-O, Omega, and Theta asymptotic notation.
3. Perform a running-time analysis on simple functions, including ones containing for loops.

Comparing Elementary Functions (continued from last class)

Elementary Function Growth Hierarchy Theorem

For all $a, b \in \mathbb{R}^+$, the following statements are true:

1. If $a > 1$ and $b > 1$, then $\log_a n \in \Theta(\log_b n)$.
    - E.g., $\log_2 n \in \Theta(\log_{100} n)$

2. If $a < b$, then $n^a \in \mathcal{O}(n^b)$ and $n^a \notin \Omega(n^b)$.
    - E.g., $n^2 \in \mathcal{O}(n^{100})$ and $n^2 \notin \Omega(n^{100})$

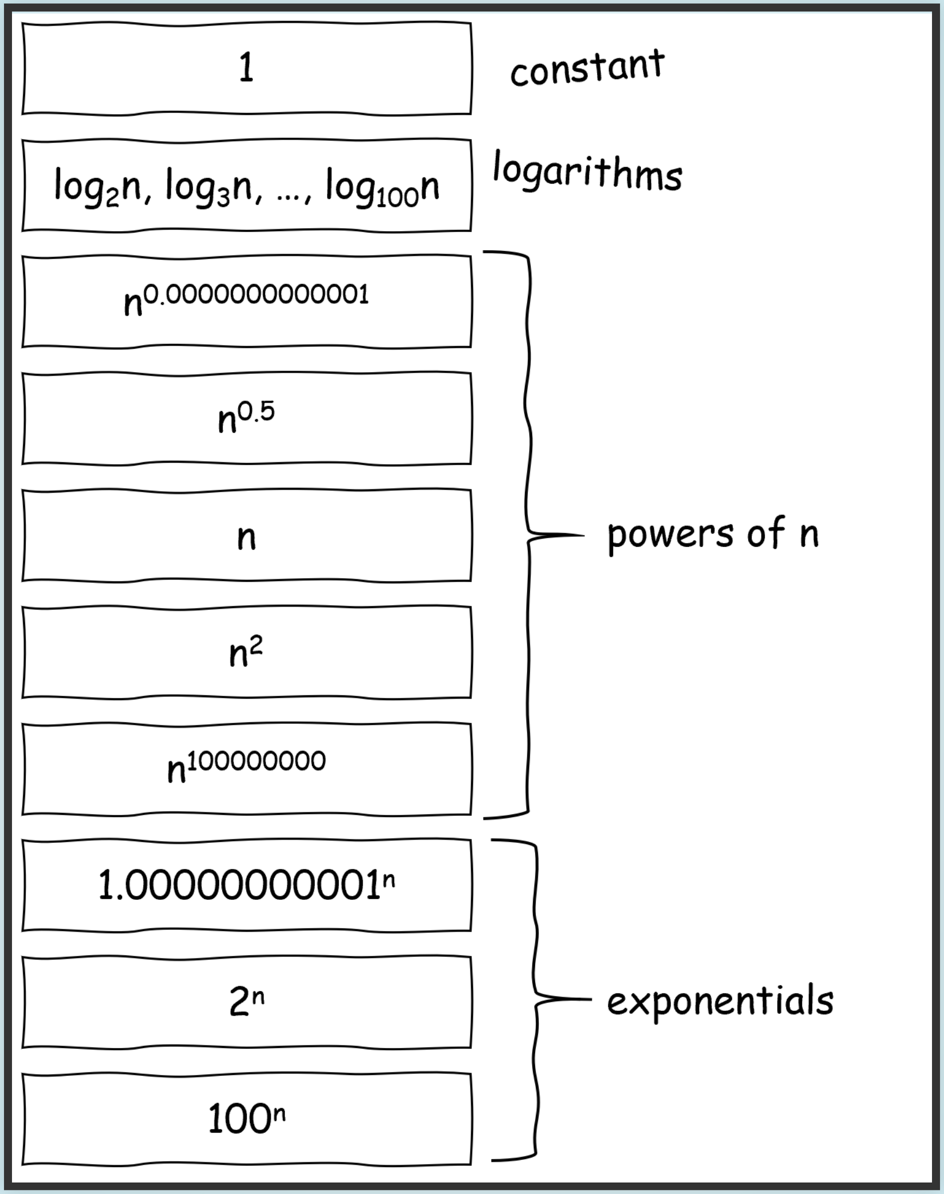
3. If $a < b$, then $a^n \in \mathcal{O}(b^n)$ and $a^n \notin \Omega(b^n)$.
    - E.g., $2^n \in \mathcal{O}(100^n)$ and $2^n \notin \Omega(100^n)$

Theorem (function growth hierarchy)

4. If $a > 1$, then $1 \in \mathcal{O}(\log_a n)$ and $1 \notin \Omega(\log_a n)$.
   - Note: 1 means the constant function $g(n) = 1$ for all $n \in \mathbb{N}$

5. If $a > 1$, then $\log_a n \in \mathcal{O}(n^b)$ and $\log_a n \notin \Omega(n^b)$.
   - E.g., $\log_2 n \in \mathcal{O}(n^{0.0000000001})$ and $\log_2 n \notin \Omega(n^{0.0000000001})$

6. If $b > 1$, then $n^a \in \mathcal{O}(b^n)$ and $n^a \notin \Omega(b^n)$.
   - E.g., $n^{10000} \in \mathcal{O}(1.0000001^n)$ and $n^{10000} \notin \Omega(1.0000001^n)$


1
constant
$\log_2 n, \log_3 n, \ldots, \log_{100} n$
logarithms
$n^{0.0000000000001}$
$n^{0.5}$
n
n^2
$n^{100000000}$
powers of n
1.00000000001^n
2^n
100^n
exponentials

Properties of Big-O, Omega, and Theta

A few useful properties

For all functions $f, g, h : \mathbb{N} \to \mathbb{R}^{\geq 0}$ the following are True:

1. $f \in \Theta(f)$
2. $f \in \mathcal{O}(g) \Leftrightarrow g \in \Omega(f)$
3. $f \in \mathcal{O}(g) \wedge g \in \mathcal{O}(h) \Rightarrow f \in \mathcal{O}(h)$ (transitivity of Big-O)
4. $f \in \Omega(g) \wedge g \in \Omega(h) \Rightarrow f \in \Omega(h)$ (transitivity of Omega)
5. $f \in \Theta(g) \wedge g \in \Theta(h) \Rightarrow f \in \Theta(h)$ (transitivity of Theta)

Constant scaling

For all $f : \mathbb{N} \to \mathbb{R}^{\geq 0}$ and $a \in \mathbb{R}^+$, $a \cdot f \in \Theta(f)$.

Examples:

- $100n^2 \in \Theta(n^2)$
- $50 \cdot 2^n \in \Theta(2^n)$
- $7 \in \Theta(1)$

Sum of Functions

For all $f, g : \mathbb{N} \to \mathbb{R}^{\geq 0}$, if $g \in \mathcal{O}(f)$ then $f + g \in \Theta(f)$.

For example, since $n \in \mathcal{O}(n^2)$, we know $n^2 + n \in \Theta(n^2)$.

Note: an expanded version of this theorem is written in Section 9.3.

Example: applying these properties

Prove that $100n^2 + 9\log n \in \Theta(n^2)$.

Proof.

We know $100n^2 \in \Theta(n^2)$, by the Constant Scaling Theorem.

We know $9\log n \in \Theta(\log n)$ (Constant Scaling).

And we know $\log n \in \mathcal{O}(n^2)$ (Elementary Function Growth Hierarchy).

So $9\log n \in \mathcal{O}(n^2)$ (transitivity of Big-O).

So then $100n^2 + 9\log n \in \Theta(n^2)$, by the Sum of Functions Theorem.

Example proof of a property

Prove that: $\forall f, g : \mathbb{N} \to \mathbb{R}^{\geq 0},\ g \in \mathcal{O}(f) \Rightarrow f \in \Omega(g)$.

$$\begin{aligned}
&\forall f, g : \mathbb{N} \to \mathbb{R}^{\geq 0}, \\
&\quad \big(\exists c_1, n_1 \in \mathbb{R}^+,\ \forall n \in \mathbb{N},\ n \geq n_1 \Rightarrow g(n) \leq c_1 \cdot f(n)\big) \Rightarrow \\
&\quad \big(\exists c_2, n_2 \in \mathbb{R}^+,\ \forall n \in \mathbb{N},\ n \geq n_2 \Rightarrow f(n) \geq c_2 \cdot g(n)\big)
\end{aligned}$$

Let $f, g : \mathbb{N} \to \mathbb{R}^{\geq 0}$.

Assume there exist $c_1, n_1 \in \mathbb{R}^+$ such that
$\forall n \in \mathbb{N},\ n \geq n_1 \Rightarrow g(n) \leq c_1 \cdot f(n)$.

Let $c_2 =$ _____. Let $n_2 =$ _____. Let $n \in \mathbb{N}$ and assume $n \geq n_2$.

We will prove that $f(n) \geq c_2 g(n)$.

Rough work

Given: $g(n) \leq c_1 \cdot f(n)$, when $n \geq n_1$

Want: $f(n) \geq \_\_\_\_ \cdot g(n)$, when $n \geq \_\_\_\_$

$$f(n) \geq \frac{1}{c_1} \cdot g(n), \text{ when } n \geq n_1$$

Take $c_2 = \frac{1}{c_1}$, and $n_2 = n_1$.

Let $f, g : \mathbb{N} \to \mathbb{R}^{\geq 0}$.

Assume there exist $c_1, n_1 \in \mathbb{R}^+$ such that
$\forall n \in \mathbb{N},\ n \geq n_1 \Rightarrow g(n) \leq c_1 \cdot f(n)$.

Let $c_2 = \frac{1}{c_1}$. Let $n_2 = n_1$. Let $n \in \mathbb{N}$ and assume $n \geq n_2$.

We will prove that $f(n) \geq c_2 g(n)$.

Since $n \geq n_2 = n_1$, we know $g(n) \leq c_1 \cdot f(n)$ (from our Big-O assumption).

Then dividing by c_1, we have:

$$\frac{1}{c_1} \cdot g(n) \leq f(n)$$

$$c_2 \cdot g(n) \leq f(n) \qquad \left(\text{since } c_2 = \frac{1}{c_1}\right)$$

Exercise 1: Properties of asymptotic growth

Running-Time Analysis

How long does a program take to run?

Problem: different computers run at different speeds

Solution: Count “basic operations” (or “steps”), not milliseconds or other units of time

Problem: if a program operates on more data, it naturally takes longer

Solution: Measure running time as a function of input size

Problem: Small differences in lines of code may impact running time measurement

Solution Allow for “approximate” counting

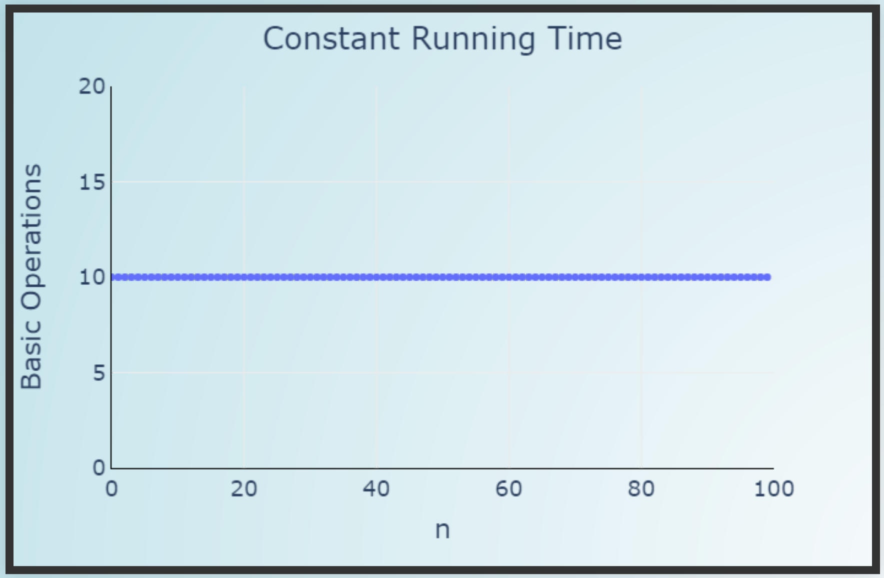
Definition: The running-time function

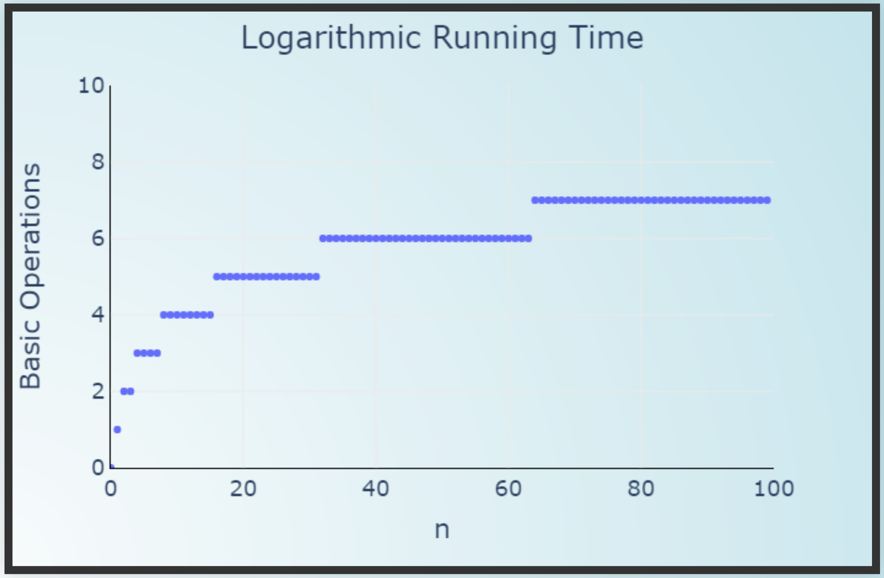
Let `func` be an algorithm (e.g., Python function). We define the **running-time function of `func`** as $RT_{func} : \mathbb{N} \to \mathbb{R}^{\geq 0}$, where

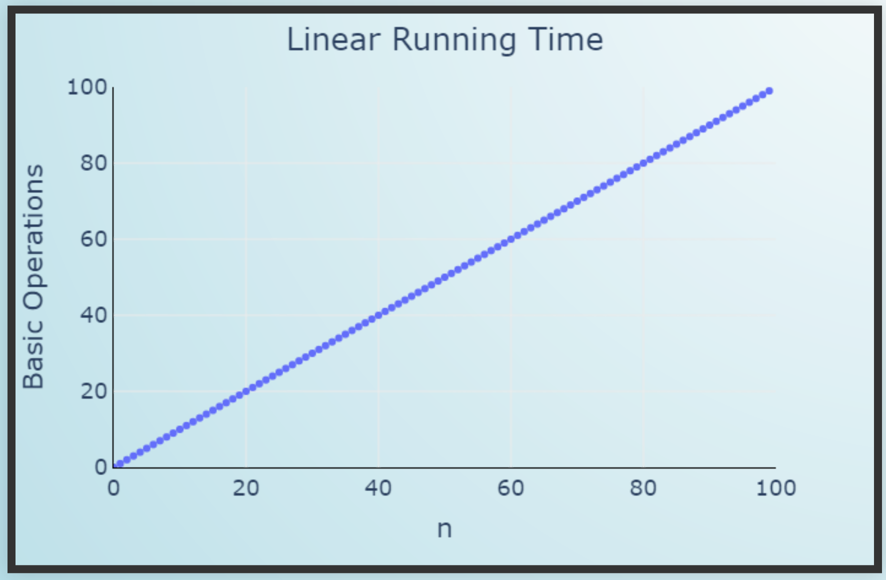
$RT_{func}(n)$ = number of steps `func` takes on an input of size n

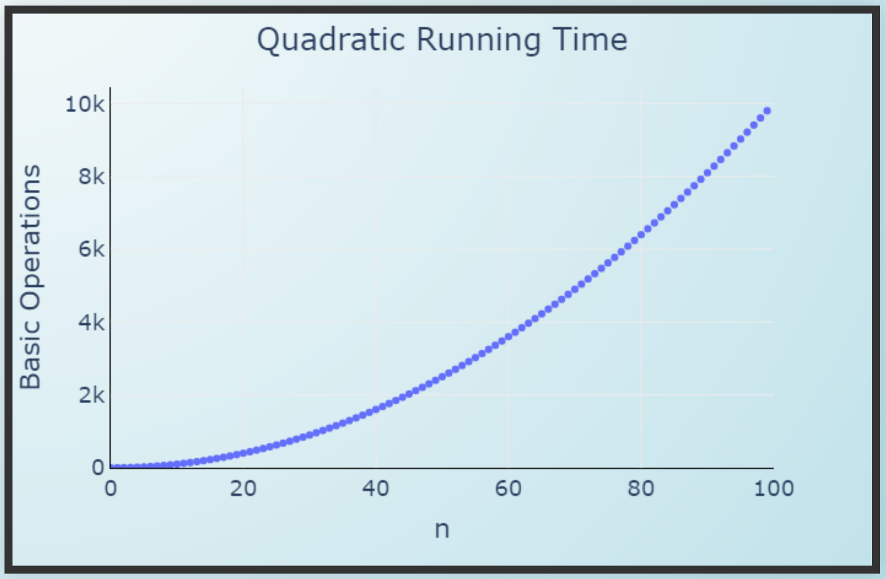
Goal of a **running-time analysis** of `func`:

1. First, find a formula for RT_{func} (in terms of n) by analysing the code.
2. Find an elementary function f such that $RT_{func} \in \Theta(f)$.


Constant Running Time
Basic Operations
0
5
10
15
20
n
0
20
40
60
80
100


Logarithmic Running Time
Basic Operations
0
2
4
6
8
10
n
0
20
40
60
80
100


Linear Running Time
Basic Operations
0
20
40
60
80
100
n
0
20
40
60
80
100


Quadratic Running Time
Basic Operations
0
2k
4k
6k
8k
10k
n
0
20
40
60
80
100

What is a step?

A **basic operation** (or **step**) is a block of code whose running time does not depend on the size of the function's input.

We say that these are "constant time" operations.

Examples of basic operations

- Arithmetic and comparison operations on numbers (`+`, `<`)
- Assignment statements (all data types)
- Calling `print` on numbers; calling `len` on collections
- Returning from a function

Example 1

Analyse the running time of the following function.

```
def f(numbers: list[int]) -> bool:
    x = len(numbers) + 1
    y = x * 3
    return len(numbers) + y > 9000
```

Analysis. Let n be the length of the input list `numbers`.

All expressions and statements in the body of `f` are constant time operations, so we count the whole body as 1 step.

So $RT_f(n) = 1$, which is $\Theta(1)$. `f` is a constant time function.

Note: Even if we count this function as 3 steps, we still get $\Theta(1)$!

Example 2

```
def print_items(numbers: list[int]) -> None:
    for number in numbers:
        print(number)
```

A for loop represents repeated code. To analyse its running time, we need to add up the steps taken by all iterations of the loop.

Procedure for analysing the running time of a for loop:

1. Determine the number of iterations.
2. Determine the number of steps in each iteration.

Example 2

```
def print_items(numbers: list[int]) -> None:
    for number in numbers:
        print(number)
```

Analysis. Let n be the length of the input `numbers`.

- The for loop has n iterations
- A single iteration takes 1 step (since calling `print` on a number is constant time)

So the total number of steps is $n \cdot 1 = n$, which is $\Theta(n)$.

Example 3

```
def my_sum(numbers: list[int]) -> int:
    sum_so_far = 0

    for number in numbers:
        sum_so_far = sum_so_far + number

    return sum_so_far
```

When you see a mixture of constant-time and non-constant-time statements, calculate the number of steps for each statement separately and add the result.

```
def my_sum(numbers: list[int]) -> int:          # Line 1
    sum_so_far = 0                              # Line 2
                                                # Line 3
    for number in numbers:                      # Line 4
        sum_so_far = sum_so_far + number        # Line 5
                                                # Line 6
    return sum_so_far                           # Line 7
```

Analysis. Let n be the length of the input list `numbers`.

1. (Line 2) `sum_so_far = 0` takes 1 step (constant time)
2. (Line 4-5) the for loop takes n steps, because:
    - it takes n iterations
    - each iteration takes 1 step (constant time)
3. (Line 7) `return sum_so_far` takes 1 step (constant time)

So the total running time is $1 + n + 1 = n + 2$ steps, which is $\Theta(n)$.

Exercise 2: Analysing running time of for loops

Summary

Today you learned to...

1. Compare different elementary functions using asymptotic notation.
2. State and prove useful properties about Big-O, Omega, and Theta asymptotic notation.
3. Perform a running-time analysis on simple functions, including ones containing for loops.

Homework

- Readings:
  - From today: 9.3, 9.5
  - For after reading week: 9.6, 9.7, 9.8, 9.9
- Prep 9 due Monday November 14 (after Reading Week)
- Assignment 4 has been posted
- **Enjoy Reading Week!** 🙌

David's realization of the week