Formal Big-O Notation

Making Our Intuition Precise

Prof. Antonio Khalil Moretti

SCIS 313: Data Structures and Algorithm Analysis

Spelman College

What We Observed Last Time

From our experiments, we noticed:

- $n^2 - 3n + 2$ and n^2 looked nearly identical for large n
- We said both are $O(n^2)$
- Constants and lower-order terms became negligible

But what does $O(n^2)$ *really* mean mathematically?

Today: We'll make our intuition precise (without getting too scary!)

The Intuitive Idea

Before We Get Formal

"$f(n) = O(g(n))$" means

"f grows no faster than g"

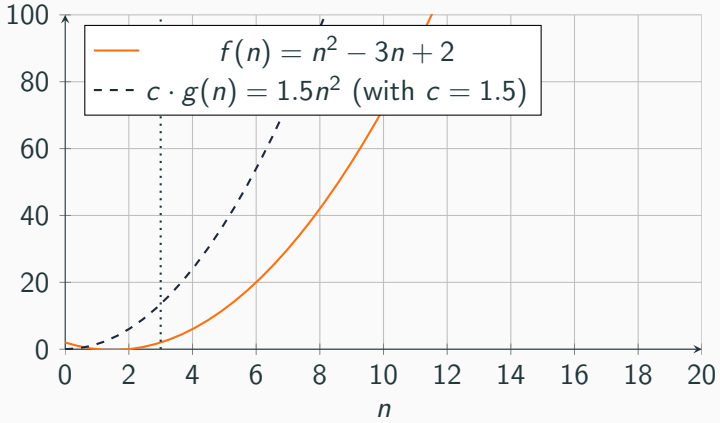
More precisely:

- For large enough n
- $f(n)$ is bounded by some multiple of $g(n)$
- We can always make $f(n) \leq c \cdot g(n)$ by picking c big enough

Key insight: We only care about what happens for *large* n

Visualizing Big-O

$f(n) = O(g(n))$ **means** f **is eventually bounded by** $c \cdot g(n)$



For all $n \geq n_0$, we have $f(n) \leq c \cdot g(n)$

The Formal Definition

Don't panic - we'll break this down!

Definition: $f(n) = O(g(n))$ means:
There exist positive constants c and n_0 such that

$$0 \leq f(n) \leq c \cdot g(n)$$

for all $n \geq n_0$

Translation:

- Pick *any* constants $c > 0$ and $n_0 > 0$ that work
- After some point (n_0), f is always below $c \cdot g$
- We only care about large n (ignore small values)

Example 1: Proving $3n + 5 = O(n)$

Step-by-step proof

Claim: $3n + 5 = O(n)$

Proof: We need to find c and n_0 such that $3n + 5 \leq c \cdot n$ for all $n \geq n_0$

Approach: Let's try $c = 4$ and $n_0 = 5$

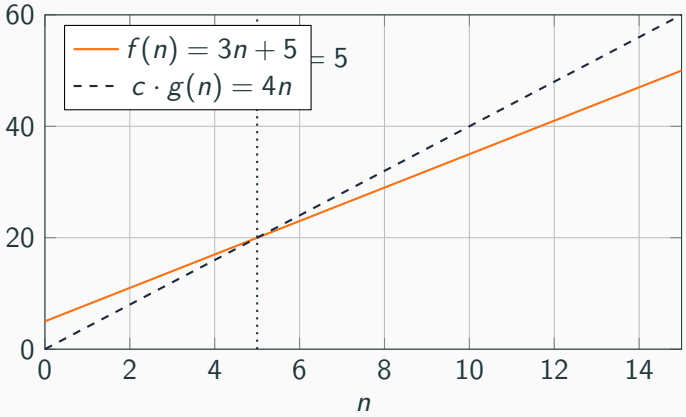
$$\begin{aligned} 3n + 5 &\leq 4n \quad \text{for all } n \geq 5 \\ 5 &\leq 4n - 3n \\ 5 &\leq n \end{aligned}$$

This is true for all $n \geq 5$! ✓

Therefore: $3n + 5 = O(n)$ with $c = 4$ and $n_0 = 5$

Example 1: Visualized

$3n + 5 \leq 4n$ **for** $n \geq 5$



Note: We could have chosen different values! For example, $c = 8$ and $n_0 = 1$ also works.

Why Constants Don't Matter

The power of Big-O

Question: Is $100n = O(n)$?

Answer: Yes! Pick $c = 100$ and $n_0 = 1$

$$100n \leq 100n \quad \text{for all } n \geq 1$$

General principle:

- Any constant factor can be absorbed into c
- $5n = O(n)$, $1000n = O(n)$, $0.001n = O(n)$
- This is why we drop constant multipliers!

Example 2: Proving $5n^2 + 100n + 3 = O(n^2)$

Dealing with multiple terms

Claim: $5n^2 + 100n + 3 = O(n^2)$

Proof: We need $5n^2 + 100n + 3 \leq c \cdot n^2$ for large n

For $n \geq 1$:

$$100n \leq 100n^2 \quad \text{(multiply both sides by } n\text{)}$$
$$3 \leq 3n^2 \quad \text{(since } n \geq 1\text{)}$$

Therefore:

$$\begin{aligned} 5n^2 + 100n + 3 &\leq 5n^2 + 100n^2 + 3n^2 \\ &= 108n^2 \end{aligned}$$

Works with $c = 108$ and $n_0 = 1$

Why Lower-Order Terms Don't Matter

From the previous example: $5n^2 + 100n + 3 = O(n^2)$

Key insight:

- For large n, the n^2 term dominates
- At $n = 1000$:
  - $5n^2 = 5,000,000$
  - $100n = 100,000$ (only 2% of total)
  - $3 \approx 0$ (negligible)
- The $100n$ and 3 become insignificant

General rule: Keep only the highest-degree term

Big-Ω: Lower Bounds

The flip side of Big-O

Definition: $f(n) = \Omega(g(n))$ means:
There exist constants $c > 0$ and $n_0 > 0$ such that

$$f(n) \geq c \cdot g(n)$$

for all $n \geq n_0$

Interpretation:

- O: upper bound ("grows no faster than")
- Ω: lower bound ("grows at least as fast as")

Example: $3n^2 + 5n = \Omega(n^2)$ because $3n^2 + 5n \geq 3n^2$ for all $n \geq 0$

Big-Θ: Tight Bounds

When upper and lower bounds match

Definition: $f(n) = \Theta(g(n))$ means:
$f(n) = O(g(n))$ **and** $f(n) = \Omega(g(n))$

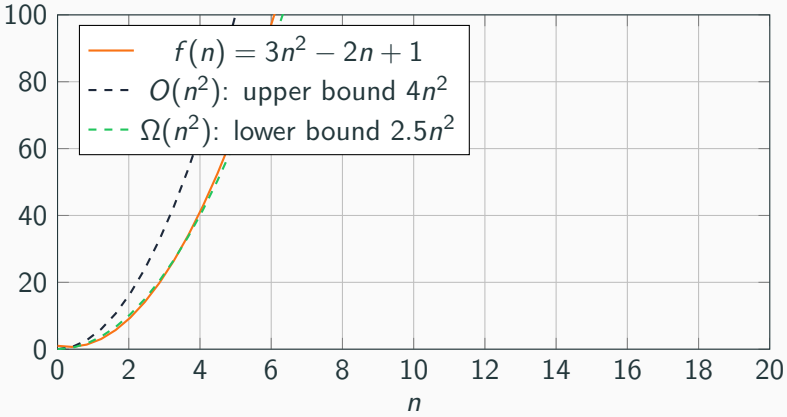
What this means:

- f and g grow at the same rate
- f is sandwiched between two multiples of g
- Most precise characterization

Example: $3n^2 + 5n = \Theta(n^2)$

- Upper: $3n^2 + 5n \leq 8n^2$ for $n \geq 1$
- Lower: $3n^2 + 5n \geq 3n^2$ for $n \geq 0$

O, Ω, and Θ Visualized



$f(n)$ is squeezed between the bounds $\Rightarrow f(n) = \Theta(n^2)$

Which Notation Do We Use?

Practical guidance

In practice:

- **Big-O** is most common
  - Talking about worst-case performance
  - "This algorithm is $O(n \log n)$"
  - Technically an upper bound
- **Big-Θ** is most precise
  - When we know exact growth rate
  - "Bubble sort is $\Theta(n^2)$"
- **Big-Ω** is less common
  - For lower bounds / impossibility results
  - "Sorting requires $\Omega(n \log n)$ comparisons"

Often: People say O when they mean Θ (technically sloppy but understood)

Practice: Prove $n \log n = O(n^2)$

Task: Show that $n \log n = O(n^2)$

Hint: For $n \geq 2$, we have $\log n < n$

Practice: Prove $n \log n = O(n^2)$

Task: Show that $n \log n = O(n^2)$

Hint: For $n \geq 2$, we have $\log n < n$

Solution:

$$n \log n < n \cdot n \quad (\text{since } \log n < n)$$
$$n \log n < n^2$$

Therefore $n \log n = O(n^2)$ with $c = 1$ and $n_0 = 2$

Practice: Is $n^2 = O(n)$?

Question: Can we prove $n^2 = O(n)$?

We would need: $n^2 \leq c \cdot n$ for some c and all $n \geq n_0$

Practice: Is $n^2 = O(n)$?

Question: Can we prove $n^2 = O(n)$?

We would need: $n^2 \leq c \cdot n$ for some c and all $n \geq n_0$

Answer: No!

Dividing both sides by n: we'd need $n \leq c$

But this can't be true for *all* $n \geq n_0$ (no matter what c we pick)

Lesson: You can't "shrink" to a smaller complexity class

Little-o and Little-ω

Strict bounds (brief mention)

Sometimes we need *strict* bounds:

- **Little-o:** $f(n) = o(g(n))$ means f grows *strictly slower* than g
  - $n = o(n^2)$ ✓
  - $n^2 \neq o(n^2)$ ✗ (not strictly slower than itself)
- **Little-ω:** $f(n) = \omega(g(n))$ means f grows *strictly faster* than g
  - $n^2 = \omega(n)$ ✓

Analogy with numbers:

- O is like $\leq$, o is like $<$
- Ω is like $\geq$, ω is like $>$
- Θ is like $=$

Summary of Asymptotic Notations

Notation	Meaning	Analogy	Example
O	Upper bound	$\leq$	$n = O(n^2)$
Ω	Lower bound	$\geq$	$n^2 = \Omega(n)$
Θ	Tight bound	$=$	$3n^2 = \Theta(n^2)$
o	Strict upper	$<$	$n = o(n^2)$
ω	Strict lower	$>$	$n^2 = \omega(n)$

Most important for this class: O, Ω, and Θ

Common Mistakes to Avoid

Mistake 1: Saying "This algorithm is $O(n)$ or $O(n^2)$"

- ✗ It's one or the other, not both!
- Better: "worst case is $O(n^2)$, best case is $O(n)$"

Mistake 2: Confusing O with Θ

- $n = O(n^2)$ is technically true (but not tight)
- $n = \Theta(n)$ is the precise statement

Mistake 3: Using "exponential" for anything that grows fast

- n^2 is polynomial, not exponential
- Exponential means 2^n, e^n, etc.

Mistake 4: Forgetting that $n^2 \neq O(n)$

- Can't bound a faster-growing function by a slower one

Connecting to Our Experiments

Last time we measured:

- Bubble sort: $\sim$59 ms for $n = 15,000$
- Standard sort: $\sim$6 ms for $n = 100,000$

Now we can *prove* mathematically:

- Bubble sort has 2 nested loops $\Rightarrow \Theta(n^2)$ operations
- Standard sort uses divide-and-conquer $\Rightarrow \Theta(n \log n)$ operations

Empirical + Mathematical = Complete understanding

What's Next

Building on This Foundation

Today we learned:

- Formal definition of Big-O, Ω, and Θ
- How to prove complexity bounds
- Why constants and lower-order terms don't matter
- Common notations and their meanings

Next class:

- Analyzing code systematically
- Counting operations in loops
- Analyzing recursive algorithms
- Best/worst/average case analysis

Homework: Practice proving complexity bounds (problems posted on Canvas)

Logarithms

The inverse of exponents

Question: $2^{?} = 8$

Answer: $? = 3$ because $2^3 = 8$

Logarithm notation: $\log_2 8 = 3$

In words: "log base 2 of 8 equals 3"

General definition:

$$\log_b x = y \quad \text{means} \quad b^y = x$$

Examples:

- $\log_2 16 = 4$ because $2^4 = 16$
- $\log_{10} 100 = 2$ because $10^2 = 100$
- $\log_2 1024 = 10$ because $2^{10} = 1024$

What Does a Logarithm Tell Us?

Logarithm answers the question:

"How many times do I need to multiply the base to get this number?"

Examples:

- $\log_2 1024 = 10$
  "How many times do I double 1 to get 1024?" $\rightarrow$ 10 times
- $\log_2 16 = 4$
  "How many times do I double 1 to get 16?" $\rightarrow$ 4 times

Or equivalently:

"How many times do I need to halve this number to get down to 1?"

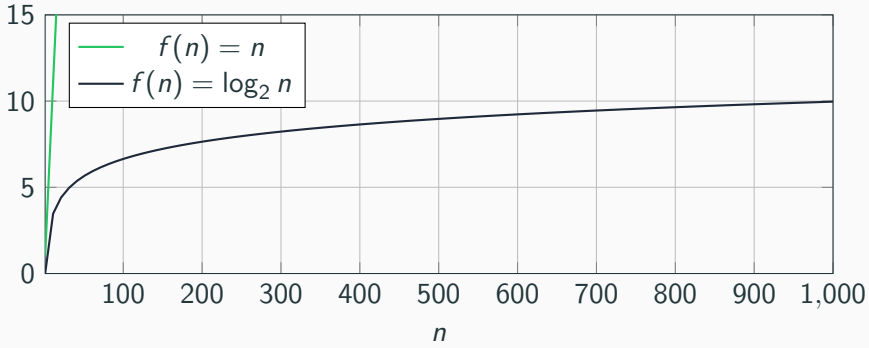
Why Logarithms Matter in Computer Science

Logarithms appear whenever we:

- **Divide a problem in half repeatedly**
  Binary search: 1000 items $\to$ 500 $\to$ 250 $\to$ 125 $\to$ ... $\to$ 1
  Takes $\log_2 1000 \approx 10$ steps
- **Double something repeatedly**
  How many times can we double before reaching n?
  Answer: $\log_2 n$ times
- **Work with trees**
  Height of balanced binary tree with n nodes?
  Answer: $\log_2 n$

Visualizing Logarithmic Growth

It grows very slowly

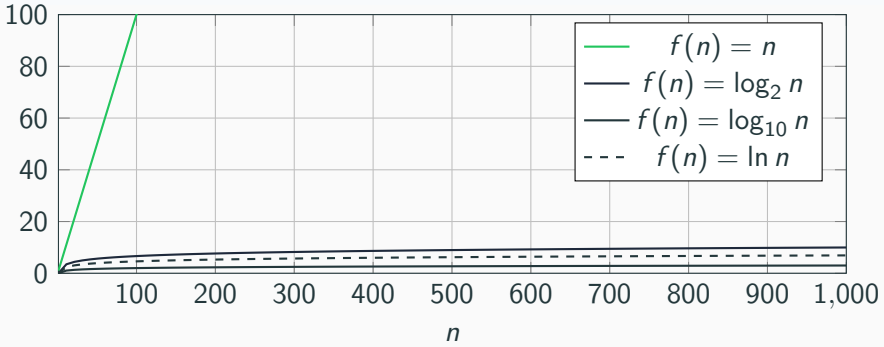


Notice: When $n = 1000$, $\log_2 n \approx 10$

Logarithmic growth is MUCH slower than linear!

Visualizing Logarithmic Growth

The base doesn't matter as much as you'd think



Key Insight: The log curves differ only by a **constant factor**.

Since $\log_a n = \frac{\ln n}{\ln a}$, changing the base just rescales by $\frac{1}{\ln a}$.

In Big-O notation, constant factors don't matter so

$O(\log_2 n) = O(\log_{10} n) = O(\log n)$.

Important Logarithm Properties

Key properties to know:

1. $\log_b(xy) = \log_b x + \log_b y$
   Multiplying inside = Adding outside
2. $\log_b(x^k) = k \log_b x$
   Power inside = Multiply outside
3. $\log_b 1 = 0$ for any base b
   Because $b^0 = 1$
4. $\log_b b = 1$ for any base b
   Because $b^1 = b$
5. **Change of base:** $\log_a x = \frac{\log_b x}{\log_b a}$
   This is why the base doesn't matter in Big-O!

Why Logarithm Base Doesn't Matter in Big-O

Mathematical fact:

$$\log_2 n = \frac{\log_{10} n}{\log_{10} 2} \approx 3.32 \cdot \log_{10} n$$

The 3.32 is just a constant!

In Big-O: We drop constants, so:

$$\log_2 n = O(\log_{10} n) = O(\ln n) = O(\log n)$$

Takeaway: When we write $O(\log n)$, we don't need to specify the base.

Common bases:

- $\log_2$ (base 2) — Most common in CS
- $\log_{10}$ (base 10) — Common in science
- ln (base $e \approx 2.718$) — Natural logarithm, used in calculus

Quick Practice: Logarithms

Try these (no calculator):

1. $\log_2 32 = ?$
2. $\log_2 1 = ?$
3. $\log_{10} 1000 = ?$
4. Approximately, $\log_2 1,000,000 = ?$

Hint for #4: $2^{10} = 1024 \approx 1000$

Solutions

1. $\log_2 32 = 5$ (because $2^5 = 32$)
2. $\log_2 1 = 0$ (because $2^0 = 1$)
3. $\log_{10} 1000 = 3$ (because $10^3 = 1000$)
4. $\log_2 1,000,000 \approx 20$

   Because $1,000,000 = 1000 \times 1000 \approx 2^{10} \times 2^{10} = 2^{20}$

Pro tip: $2^{10} \approx 1000$ is a useful approximation to remember!

In-Class Practice: Proving Asymptotic Bounds

Practice Problems to Try

Tip: Explicitly choose constants c and n_0.

In-Class Problem 1

Prove that

$$7n + 20 = O(n).$$

In-Class Problem 1

Prove that

$$7n + 20 = O(n).$$

Solution.

For $n \geq 1$, we have $20 \leq 20n$, so

$$7n + 20 \leq 7n + 20n = 27n.$$

Choose $c = 27$ and $n_0 = 1$. □

In-Class Problem 2

Prove that

$$n^2 = \Omega(n).$$

In-Class Problem 2

Prove that

$$n^2 = \Omega(n).$$

Solution.

For all $n \geq 1$,

$$n^2 \geq n.$$

Choose $c = 1$ and $n_0 = 1$. □

In-Class Problem 3

Prove that

$$4n^2 + 6n + 1 = \Theta(n^2).$$

In-Class Problem 3

Prove that

$$4n^2 + 6n + 1 = \Theta(n^2).$$

Upper bound:

For $n \geq 1$, $6n \leq 6n^2$ and $1 \leq n^2$, so

$$4n^2 + 6n + 1 \leq 11n^2.$$

In-Class Problem 3

Prove that

$$4n^2 + 6n + 1 = \Theta(n^2).$$

Upper bound:

For $n \geq 1$, $6n \leq 6n^2$ and $1 \leq n^2$, so

$$4n^2 + 6n + 1 \leq 11n^2.$$

Lower bound:

For all $n \geq 1$,

$$4n^2 + 6n + 1 \geq 4n^2.$$

In-Class Problem 3

Prove that

$$4n^2 + 6n + 1 = \Theta(n^2).$$

Upper bound:

For $n \geq 1$, $6n \leq 6n^2$ and $1 \leq n^2$, so

$$4n^2 + 6n + 1 \leq 11n^2.$$

Lower bound:

For all $n \geq 1$,

$$4n^2 + 6n + 1 \geq 4n^2.$$

Choose $c_1 = 4$, $c_2 = 11$, $n_0 = 1$. □

In-Class Problem 4

True or False:

$$3n^2 - n = \Theta(n^2)$$

In-Class Problem 4

True or False:

$$3n^2 - n = \Theta(n^2)$$

Answer: True.

In-Class Problem 4

True or False:

$$3n^2 - n = \Theta(n^2)$$

Answer: True.

For $n \geq 1$, $-n \leq 0$, so

$$3n^2 - n \leq 3n^2.$$

Also,

$$3n^2 - n \geq 2n^2 \quad \text{for } n \geq 1.$$

In-Class Problem 4

True or False:

$$3n^2 - n = \Theta(n^2)$$

Answer: True.

For $n \geq 1$, $-n \leq 0$, so

$$3n^2 - n \leq 3n^2.$$

Also,

$$3n^2 - n \geq 2n^2 \quad \text{for } n \geq 1.$$

Choose $c_1 = 2$, $c_2 = 3$, $n_0 = 1$. □

In-Class Problem 5

Prove that

$$\log n = O(n).$$

In-Class Problem 5

Prove that

$$\log n = O(n).$$

Solution.

For all $n \geq 2$, $\log n \leq n$.

In-Class Problem 5

Prove that

$$\log n = O(n).$$

Solution.

For all $n \geq 2$, $\log n \leq n$.

Choose $c = 1$ and $n_0 = 2$. □

In-Class Problem 6

Prove that

$$5n^3 + n^2 = O(n^3).$$

In-Class Problem 6

Prove that

$$5n^3 + n^2 = O(n^3).$$

Solution.

For $n \geq 1$, $n^2 \leq n^3$, so

$$5n^3 + n^2 \leq 6n^3.$$

Choose $c = 6$ and $n_0 = 1$. □

In-Class Problem 7

Prove that

$$10n^2 + 4n = \Omega(n^2).$$

In-Class Problem 7

Prove that

$$10n^2 + 4n = \Omega(n^2).$$

Solution.

For all $n \geq 1$,

$$10n^2 + 4n \geq 10n^2.$$

Choose $c = 10$ and $n_0 = 1$. □

In-Class Problem 8

Prove that

$$2n \log n + n = \Theta(n \log n).$$

In-Class Problem 8

Prove that

$$2n \log n + n = \Theta(n \log n).$$

Upper bound:

For $n \geq 2$, $n \leq n \log n$, so

$$2n \log n + n \leq 3n \log n.$$

In-Class Problem 8

Prove that

$$2n \log n + n = \Theta(n \log n).$$

Upper bound:

For $n \geq 2$, $n \leq n \log n$, so

$$2n \log n + n \leq 3n \log n.$$

Lower bound:

For all $n \geq 1$,

$$2n \log n + n \geq 2n \log n.$$

In-Class Problem 8

Prove that

$$2n \log n + n = \Theta(n \log n).$$

Upper bound:

For $n \geq 2$, $n \leq n \log n$, so

$$2n \log n + n \leq 3n \log n.$$

Lower bound:

For all $n \geq 1$,

$$2n \log n + n \geq 2n \log n.$$

Choose $c_1 = 2$, $c_2 = 3$, $n_0 = 2$. □

Homework Solutions

Problem 1 Solution

Prove that $2n^2 + 10n + 5 = O(n^2)$

Goal: Find c and n_0 such that $2n^2 + 10n + 5 \leq c \cdot n^2$ for all $n \geq n_0$.

Key idea: For $n \geq 1$, we can bound each term by something involving n^2:

$$10n \leq 10n^2 \quad (\text{since } n \leq n^2 \text{ for } n \geq 1)$$

$$5 \leq 5n^2 \quad (\text{since } 1 \leq n^2 \text{ for } n \geq 1)$$

Putting it together:

$$2n^2 + 10n + 5 \leq 2n^2 + 10n^2 + 5n^2 = 17n^2$$

So $c = 17$ and $n_0 = 1$ works. ■

Problem 2 Solution

Prove that $n^3 = \Omega(n^2)$

Goal: Find c and n_0 such that $n^3 \geq c \cdot n^2$ for all $n \geq n_0$.

Key idea: We can simplify by dividing both sides by n^2 (valid for $n \geq 1$):

$$n^3 \geq c \cdot n^2 \quad \iff \quad n \geq c$$

So if we pick $c = 1$, we just need $n \geq 1$. **Formally:** For all $n \geq 1$:

$$n^3 = n \cdot n^2 \geq 1 \cdot n^2$$

So $c = 1$ and $n_0 = 1$ works. ■

Problem 3 Solution

Prove that $5n + 3 = \Theta(n)$

Goal: Find c_1, c_2, n_0 such that $c_1 \cdot n \leq 5n + 3 \leq c_2 \cdot n$ for all $n \geq n_0$.

We need to prove **both** directions. **Upper bound ($O(n)$):** For $n \geq 1$, $3 \leq 3n$, so:

$$5n + 3 \leq 5n + 3n = 8n$$

Lower bound ($\Omega(n)$): For all $n \geq 1$:

$$5n + 3 \geq 5n \geq 5 \cdot n$$

So $c_1 = 5$, $c_2 = 8$, and $n_0 = 1$ works. ■

Problem 4 Solution

True or False: $n^2 + n = \Theta(n^2)$

Answer: TRUE

Upper bound ($O(n^2)$): For $n \geq 1$, $n \leq n^2$, so:

$$n^2 + n \leq n^2 + n^2 = 2n^2$$

Lower bound ($\Omega(n^2)$): For all $n \geq 1$:

$$n^2 + n \geq n^2 = 1 \cdot n^2$$

So $c_1 = 1$, $c_2 = 2$, and $n_0 = 1$ works. ■

Problem 5 Solution

True or False: $\log n = O(\sqrt{n})$

Answer: TRUE

Goal: Find c and n_0 such that $\log n \leq c \cdot \sqrt{n}$ for all $n \geq n_0$. We were given that

$\log n < \sqrt{n}$ for all $n \geq 2$.
That means $\log n \leq 1 \cdot \sqrt{n}$ for all $n \geq 2$. So $c = 1$ and $n_0 = 2$ works. ■ **But wait —**

why is $\log n < \sqrt{n}$ **true?** Let's build some intuition on the next slide.

Problem 5 Solution (cont.)

Why is $\log n < \sqrt{n}$? — Step 1: Substitute

Let's set $n = 2^k$. This is just a substitution to make the math cleaner. **What happens to $\log n$?**

$$\log n = \log(2^k) = k \log 2$$

By the change of base formula, $\frac{\log(2^k)}{\log 2} = \log_2(2^k) = k$, so:

$$\log_2 n = k$$

What happens to $\sqrt{n}$?

$$\sqrt{n} = \sqrt{2^k} = 2^{k/2}$$

So our question becomes: **Is $k < 2^{k/2}$?**

Problem 5 Solution (cont.)

Why is $\log n < \sqrt{n}$? — Step 2: Simplify

We want to show: $k < 2^{k/2}$ **Square both sides** (both sides are positive, so this is safe):

$$k^2 < \left(2^{k/2}\right)^2 = 2^k$$

So now we just need to show: $k^2 < 2^k$ This is a much simpler question! Let's check small values and then show it holds forever.

Problem 5 Solution (cont.)

Why is $\log n < \sqrt{n}$? — Step 3: Check small cases

Let's just compute and compare:

k	k^2	2^k	$k^2 < 2^k$?
1	1	2	✓
2	4	4	= (not yet)
3	9	8	
4	16	16	= (not yet)
5	25	32	✓
6	36	64	✓

The inequality holds starting at $k = 5$. Now we need to show it **stays true** for all $k \geq 5$.

Problem 5 Solution (cont.)

Why is $\log n < \sqrt{n}$? — Step 4: Show the gap keeps growing

When we go from k to $k+1$, what happens to each side? 2^k gets multiplied by 2:

$$2^{k+1} = 2 \cdot 2^k \quad \Longrightarrow \quad \text{it doubles}$$

k^2 increases by $2k+1$:

$$(k+1)^2 = k^2 + 2k + 1 \quad \Longrightarrow \quad \text{it grows by } 2k+1$$

At $k = 5$: $2^k = 32$ and $2k+1 = 11$. Since $32 > 11$, the doubling outpaces the $+2k+1$ increase.

For $k > 5$, 2^k only gets bigger and $2k+1$ grows much slower, so the gap keeps growing. ■

Problem 6 Solution

Which is a tighter characterization?

Statement A: This algorithm is $O(n^2)$ vs. **Statement B:** This algorithm is $\Theta(n \log n)$

Statement B is tighter. Here's why: $O(n^2)$ is an **upper bound only**. It tells us the algorithm is "no worse than n^2," but it could actually be much faster. For example, an $O(n^2)$ algorithm might actually run in $\Theta(n)$ time. $\Theta(n \log n)$ is a **tight bound**. It tells us the algorithm is *exactly* $n \log n$ growth — not faster, not slower. **"Tighter"** means the bound gives us *more precise* information about the actual growth rate.

Problem 7 Solution

Why is "Bubble sort is $O(n^{100})$" technically correct but useless?

It's technically correct because: Bubble sort runs in $O(n^2)$ time, and $n^2 \leq n^{100}$ for $n \geq 1$. Since Big-O is an upper bound, any larger function is also a valid upper bound.
It's useless because: It tells us almost nothing. Almost *every* algorithm is $O(n^{100})$.

Saying this gives us no way to compare or distinguish algorithms. **A more useful characterization:**

$$\text{Bubble sort} = \Theta(n^2)$$

This tells us the **exact** growth rate. We prefer tight bounds because they let us actually *compare* algorithms and make informed decisions about which to use.