CSE 250
Data Structures

Dr. Eric Mikida
epmikida@buffalo.edu

Dr. Oliver Kennedy
okennedy@buffalo.edu

212 Capen Hall

Day 04
Runtime Analysis
Textbook Ch. 7.3-7.4

Announcements

- Dr. Kennedy will be giving lecture on Friday and Monday
- PA 0 is due Friday
- Start PA 1 early!

From Lecture 01...

Option 1
- Very fast Prepend, Get First
- Very slow Get Nth

Option 2
- Very fast Get Nth, Get First
- Very slow Prepend

Option 3
- Very fast Get Nth, Get First
- Occasionally slow Prepend

From Lecture 01...

Option 1

- Very fast Prepend, Get First
- Very slow Get Nth

Option 2

- Very fast Get Nth, Get First
- Very slow Prepend

Option 3

- Very fast Get Nth, Get First
- Occasionally slow Prepend

What is fast? slow?

Attempt #1: Wall-clock time?

- What is fast?
  - 10s? 100ms? 10ns?
  - …it depends on the task
- Algorithm vs Implemtation
  - Compare Grace Hopper's implementation to yours
- What machine are you running on?
  - Your old laptop? A lab machine? The newest, shiniest processor?
- What bottlenecks exist? CPU vs IO vs Memory vs Network…

Attempt #1: Wall-clock time?

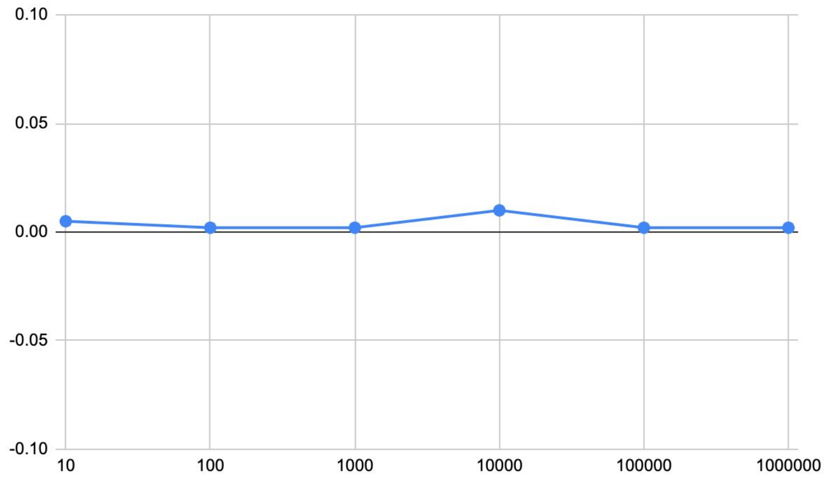
- What is fast?
  - 10s? 100ms? 10ns?
  - …it depends on the task
- Algorithm vs Implemtation
  - Compare Grace Hopper’s implementation to yours
- What machine are you running on?
  - Your old laptop? A lab machine? The newest, shiniest processor?
- What bottlenecks exist? CPU vs IO vs Memory vs Network…

Wall-clock time is not terribly useful…

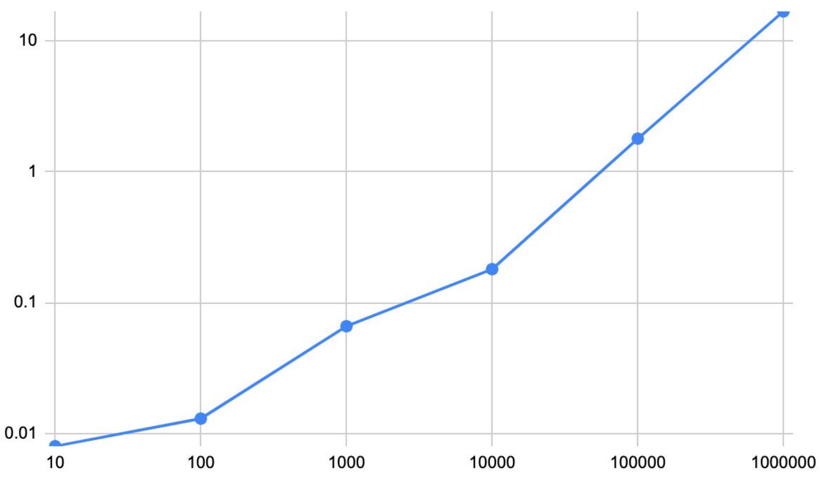
Let’s do a quick demo…

Comparing Random Access for Array vs List

Array

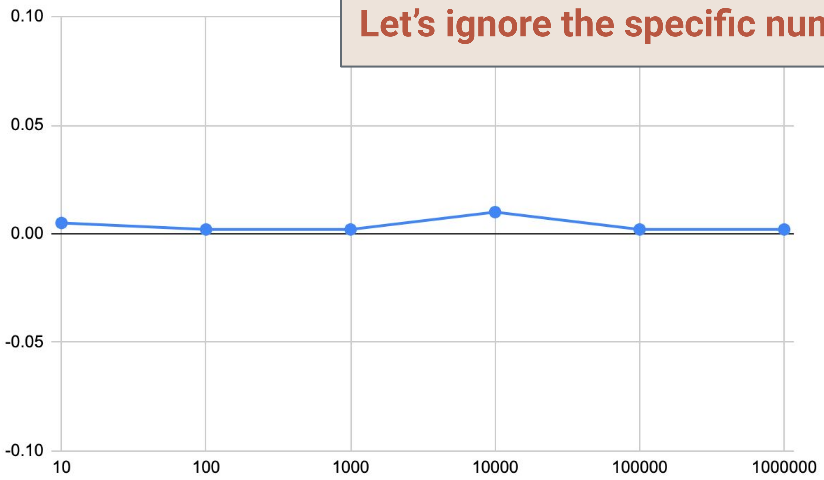


List

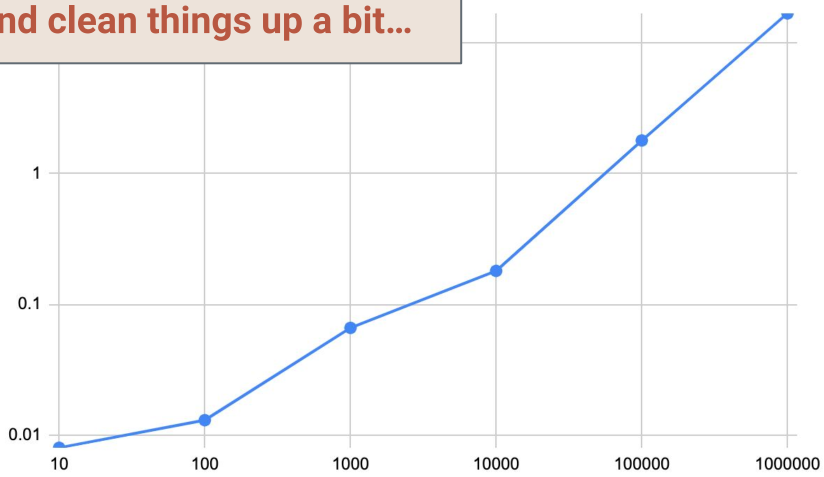


Comparing Random Access for Array vs List

Array



List

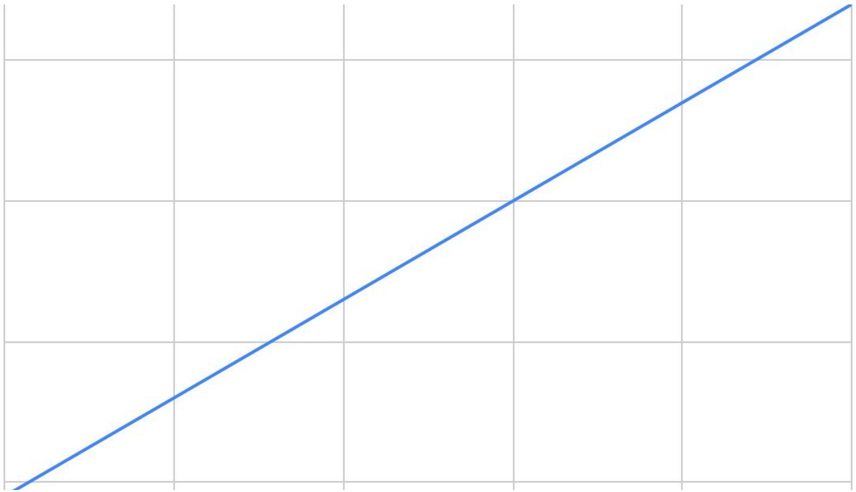


Let's ignore the specific numbers and clean things up a bit…

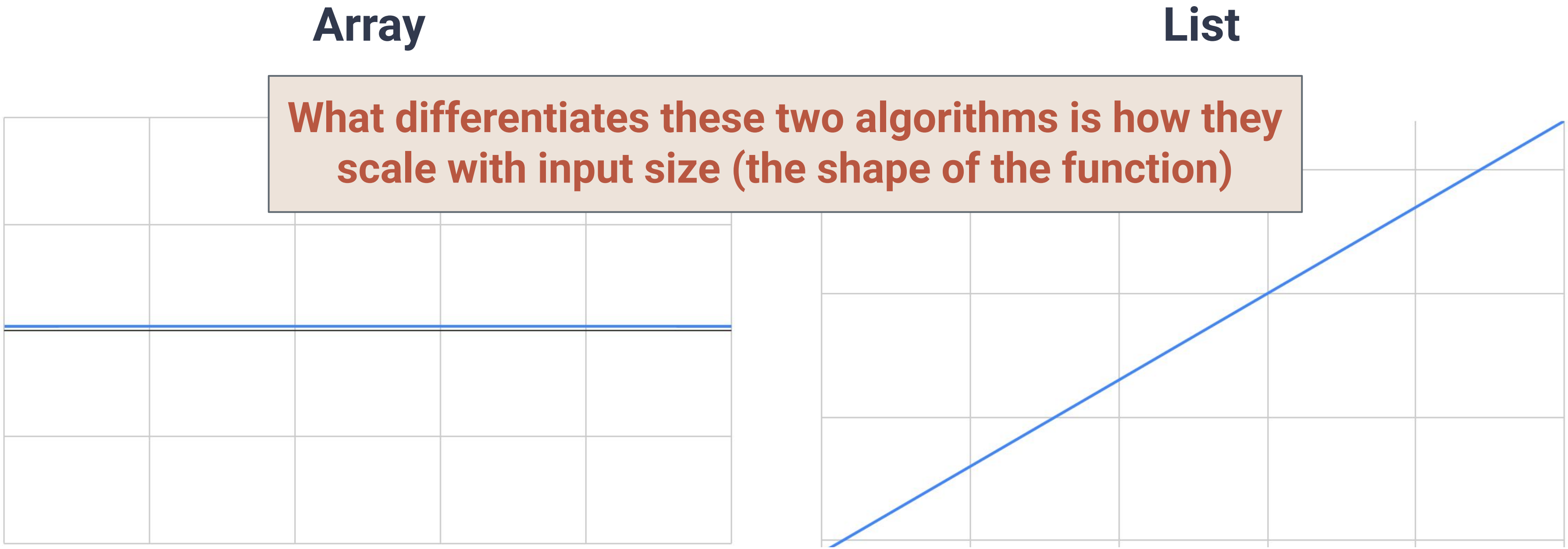
Comparing Random Access for Array vs List

Array

List



Comparing Random Access for Array vs List



What differentiates these two algorithms is how they scale with input size (the shape of the function)

When is an algorithm “fast”?

- To give a useful solution, we should take “scale” into account
  - How does the runtime change as we change the size of the input (number of users, records, pixels, elements, etc)
- Don’t think in terms of wall-time, think in terms of “number of steps”

Scaling Examples

- “Five steps plus Ten steps per user”
- “Ten steps per network connection. Each node has connections to 1% of the other nodes in the system”
- “Seven steps for every possible pair of elements
- “For each user, Ten steps, plus Three steps per post”

Scaling Examples

- “Five steps plus Ten steps per user”
  - **5 + (10 x |Users|)**
- “Ten steps per network connection. Each node has connections to 1% of the other nodes in the system”
- “Seven steps for every possible pair of elements
- “For each user, Ten steps, plus Three steps per post”

Scaling Examples

- “Five steps plus Ten steps per user”
  - **5 + (10 x |Users|)**
- “Ten steps per network connection. Each node has connections to 1% of the other nodes in the system”
  - **10 x (|Nodes| x (0.01 x |Nodes|))**
- “Seven steps for every possible pair of elements

- “For each user, Ten steps, plus Three steps per post”

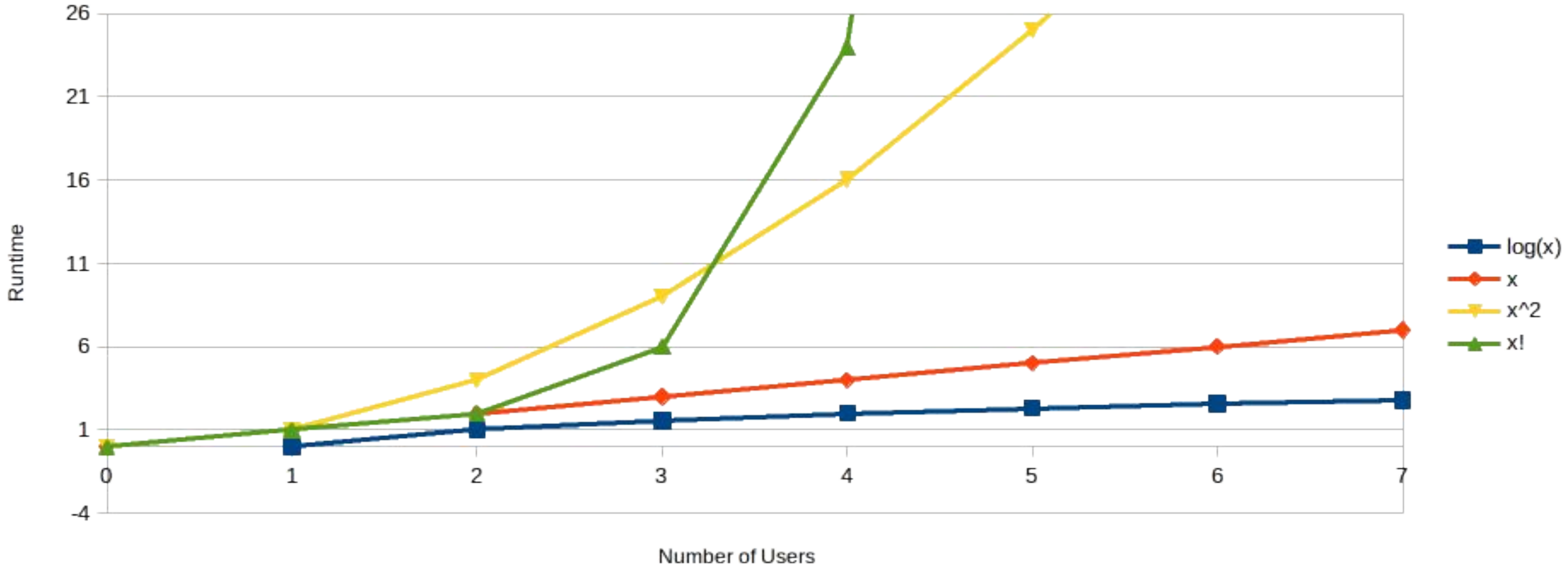
Scaling Examples

- “Five steps plus Ten steps per user”
  - **$5 + (10 \times |Users|)$**
- “Ten steps per network connection. Each node has connections to 1% of the other nodes in the system”
  - **$10 \times (|Nodes| \times (0.01 \times |Nodes|))$**
- “Seven steps for every possible pair of elements
  - **$7 \times 2^{|Elements|}$**
- “For each user, Ten steps, plus Three steps per post”

Scaling Examples

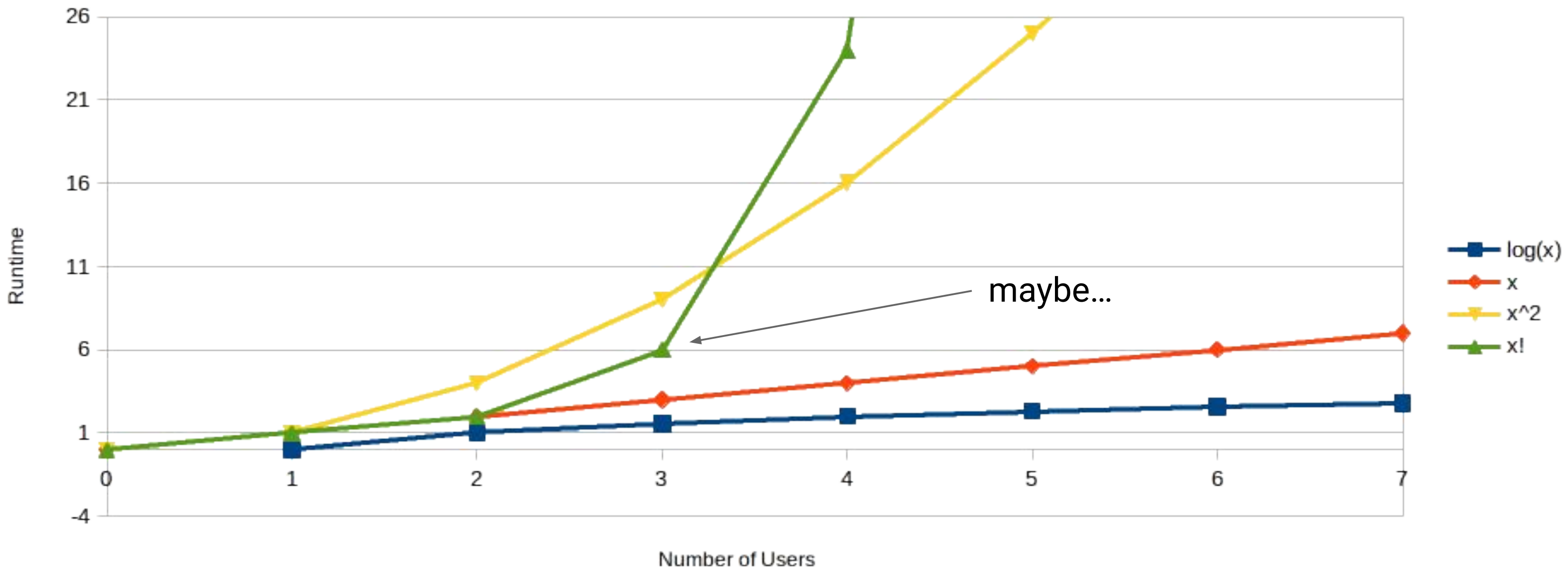
- “Five steps plus Ten steps per user”
  - **5 + (10 x |Users|)**
- “Ten steps per network connection. Each node has connections to 1% of the other nodes in the system”
  - **10 x (|Nodes| x (0.01 x |Nodes|))**
- “Seven steps for every possible pair of elements
  - **$7 \times 2^{|Elements|}$**
- “For each user, Ten steps, plus Three steps per post”
  - **|Users| x (10 + 3 x |Posts|)**

Runtime as a Function



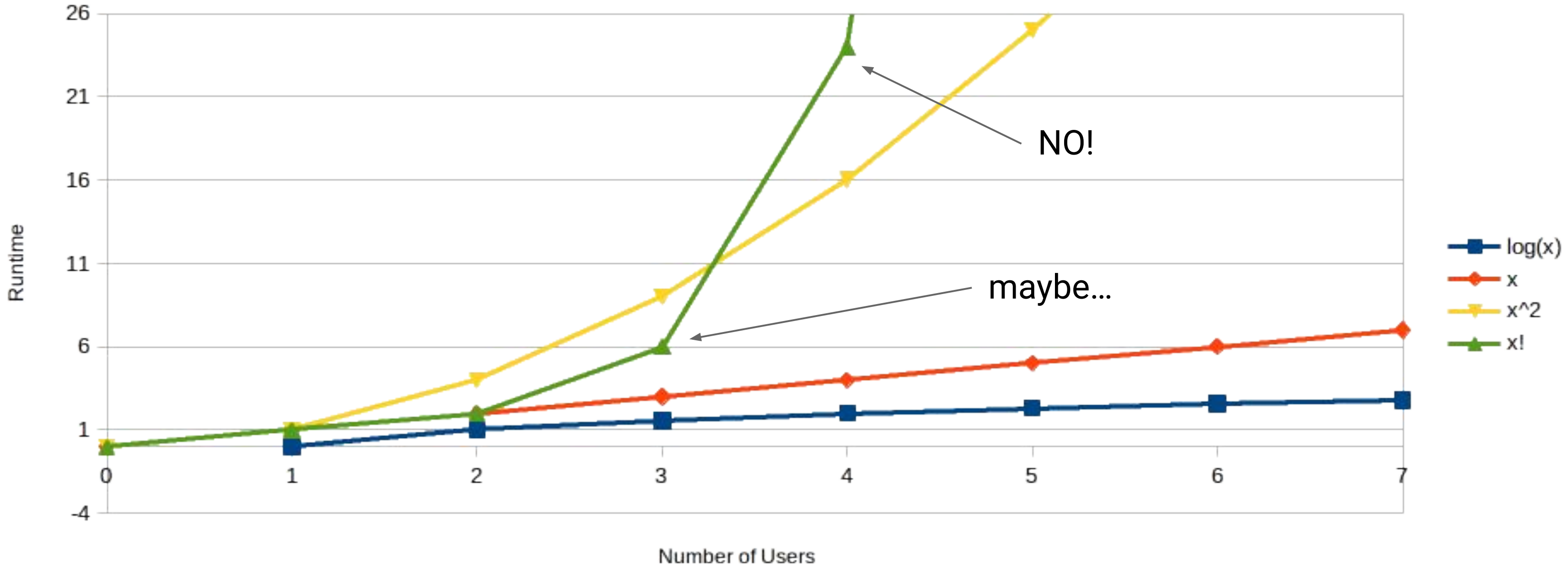
Would you consider an algorithm that takes |Users|! number of steps?

Runtime as a Function



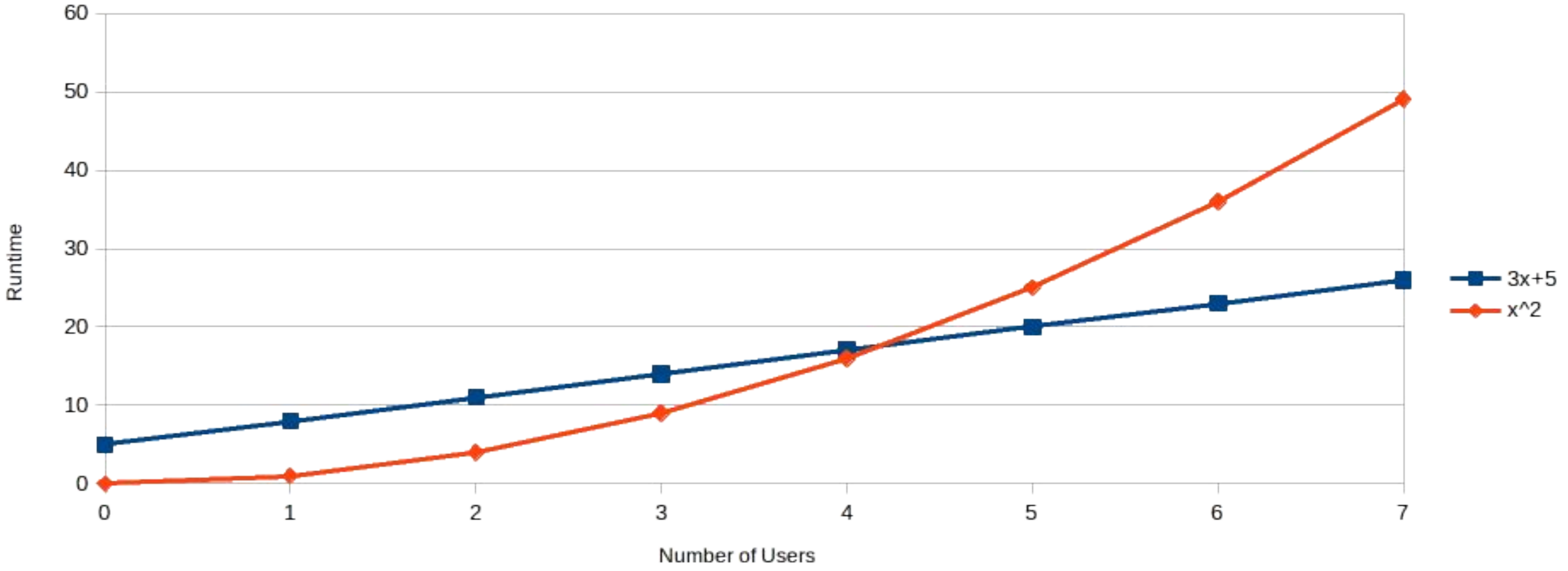
Would you consider an algorithm that takes |Users|! number of steps?

Runtime as a Function



Would you consider an algorithm that takes |Users|! number of steps?

Runtime as a Function



Which is better? 3x|Users|+5 or $|Users|^2$

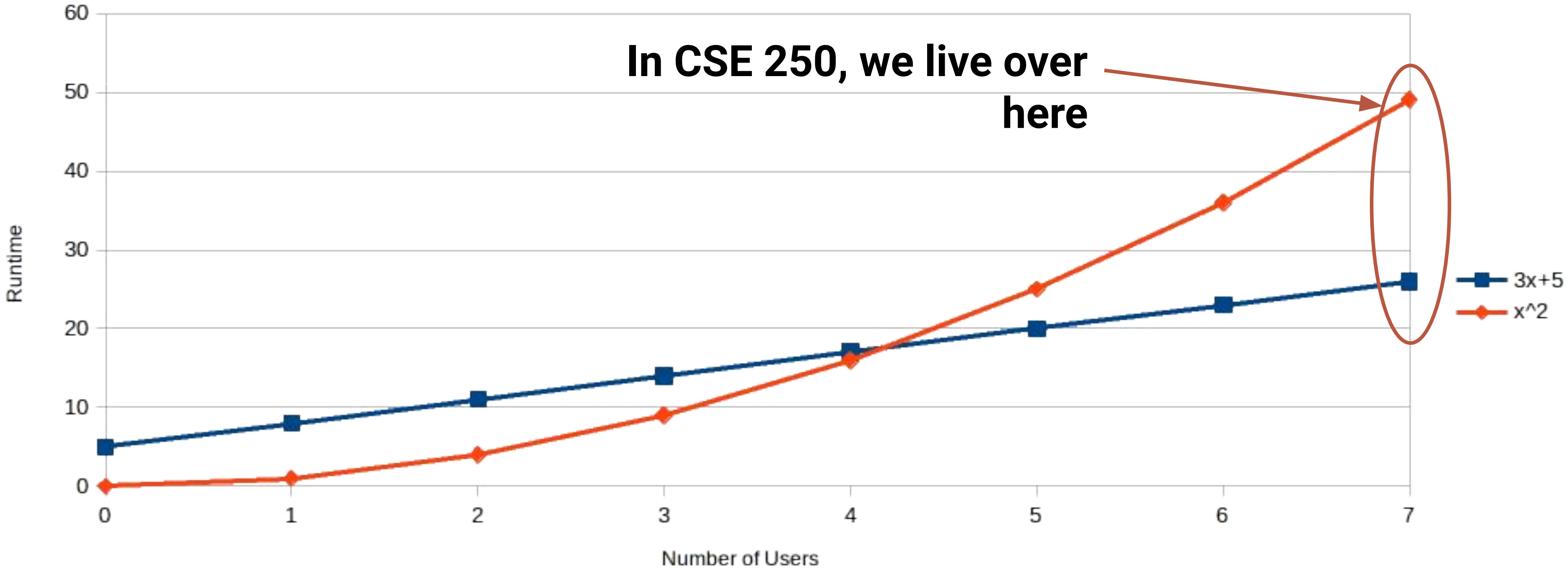
When is an algorithm "fast"?

- To give a useful solution, we should take "scale" into account
  - How does the runtime change as we change the size of the input (number of users, records, pixels, elements, etc)
- Don't think in terms of wall-time, think in terms of "number of steps"

When is an algorithm “fast”?

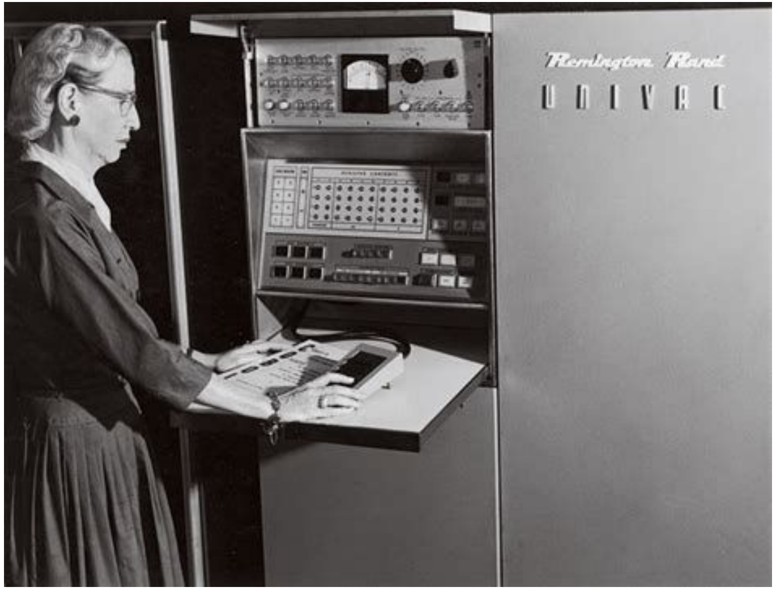
- To give a useful solution, we should take “scale” into account
  - How does the runtime change as we change the size of the input (number of users, records, pixels, elements, etc)
- Don’t think in terms of wall-time, think in terms of “number of steps”
- Focus on “large” inputs
  - Rank functions based on how they behave at large scales

Runtime as a Function



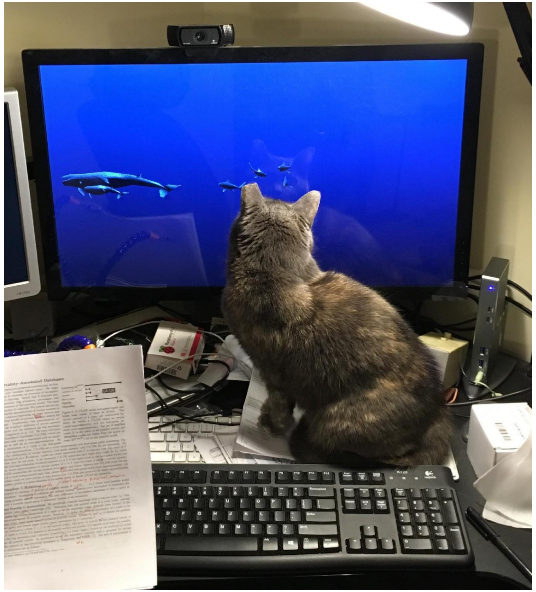
Which is better? 3x|Users|+5 or $|Users|^2$

Goal: Ignore implementation details



Seasoned Pro Implementation

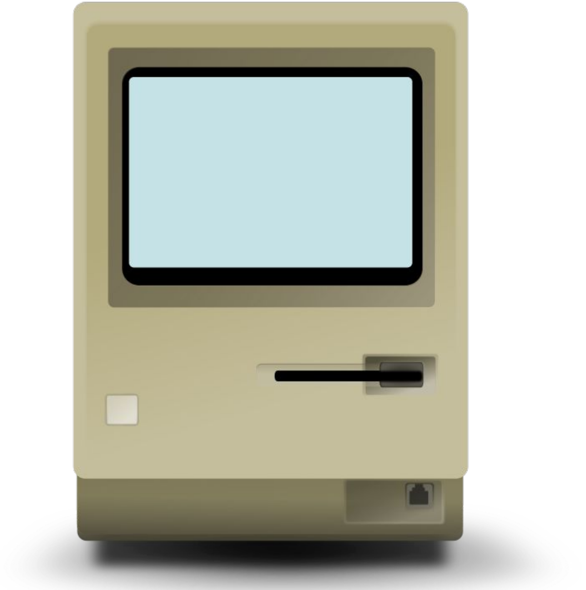
vs



Error 23: Cat on Keyboard

Goal: Ignore execution environment

vs



Intel i9

Motorola 68000

Images from openclipart.org, used with permission

Goal: Judge the Algorithm Itself

- How fast is a step? Don’t care
  - Only count number of steps
- Can this be done in two steps instead of one?
  - “3 steps per user” vs “some number of steps per user”
  - Sometimes we don’t care…sometimes we do

When is an algorithm "fast"?

- To give a useful solution, we should take "scale" into account
  - How does the runtime change as we change the size of the input (number of users, records, pixels, elements, etc)
- Don't think in terms of wall-time, think in terms of "number of steps"
- Focus on "large" inputs
  - Rank functions based on how they behave at large scales

When is an algorithm “fast”?

- To give a useful solution, we should take “scale” into account
  - How does the runtime change as we change the size of the input (number of users, records, pixels, elements, etc)
- Don’t think in terms of wall-time, think in terms of “number of steps”
- Focus on “large” inputs
  - Rank functions based on how they behave at large scales
- Decouple algorithm from infrastructure/implementation
  - Asymptotic notation…?

And now a brief interlude…

Logarithms (refresher)

Let $a, b, c, n > 0$

Exponent Rule:	$\log(n^a) = a \log(n)$
Product Rule:	$\log(an) = \log(a) + \log(n)$
Division Rule:	$\log(n/a) = \log(n) - \log(a)$
Change of Base:	$\log_b(n) = \log_c(n) / \log_b(n)$
Log/Exponent are Inverses:	$b^{\log(n)} = \log_b(b^n) = n$

Logarithms (refresher)

Let $a, b, c, n > 0$

Exponent Rule: $\log(n^a) = a \log(n)$

Product Rule: $\log(an) = \log(a) + \log(n)$

Division Rule: $\log(n/a) = \log(n) - \log(a)$

Change of Base: $\log_b(n) = \log_c(n) / \log_b(n)$

Log/Exponent are Inverses: $b^{\log(n)} = \log_b(b^n) = n$

In this class, always assume log base 2 unless specified otherwise

Now back to “fast”...

Attempt #2: Growth Functions

Not a function in code…but a mathematical function:

$$f(n)$$

n: The “size” of the input

ie: number of users,rows, pixels, etc

f(n): The number of “steps” taken for input of size n

ie: 20 steps per user, where n = |Users|, is 20 x n

Some Basic Assumptions:

Problem sizes are non-negative integers

$$n \in \mathbb{Z}^+ \cup \{0\} = \{0, 1, 2, 3, ...\}$$

We can’t reverse time…(obviously)

$$f(n) > 0$$

Smaller problems aren’t harder than bigger problems

$$\forall n_1 < n_2, f(n_1) \leq f(n_2)$$

Some Basic Assumptions:

Problem sizes are non-negative integers

$$n \in \mathbb{Z}^+ \cup \{0\} = \{0, 1, 2, 3, ...\}$$

$$f : \mathbb{Z}^+ \cup \{0\} \to \mathbb{R}^+$$

We can’t reverse time…(obviously)

$$f(n) > 0$$

Smaller problems aren’t harder than bigger problems

$$\forall n_1 < n_2, f(n_1) \leq f(n_2)$$

First Problem...

We are still implementation dependent

$$f_1(n) = 20n$$
$$f_2(n) = 19n$$

First Problem...

We are still implementation dependent

$$f_1(n) = 20n$$
$$f_2(n) = 19n$$

Does 1 extra step per element really matter...?

First Problem...

We are still implementation dependent

$$f_1(n) = 20n$$

$$f_2(n) = 19n$$

$$f_3(n) = \frac{n^2}{2}$$

f_1 and f_2 are much more "similar" to each other than they are to f_3

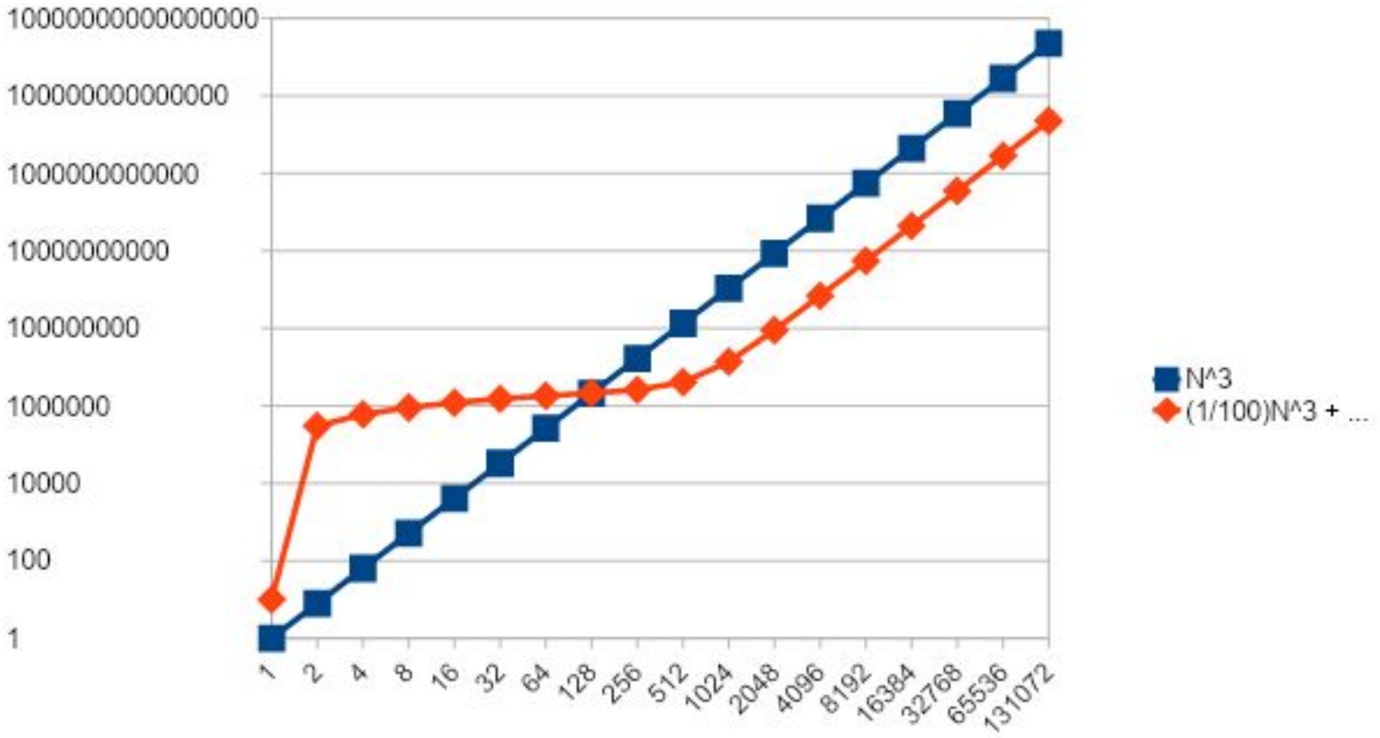
How Do We Capture Behavior at Scale?

Consider the following two functions:

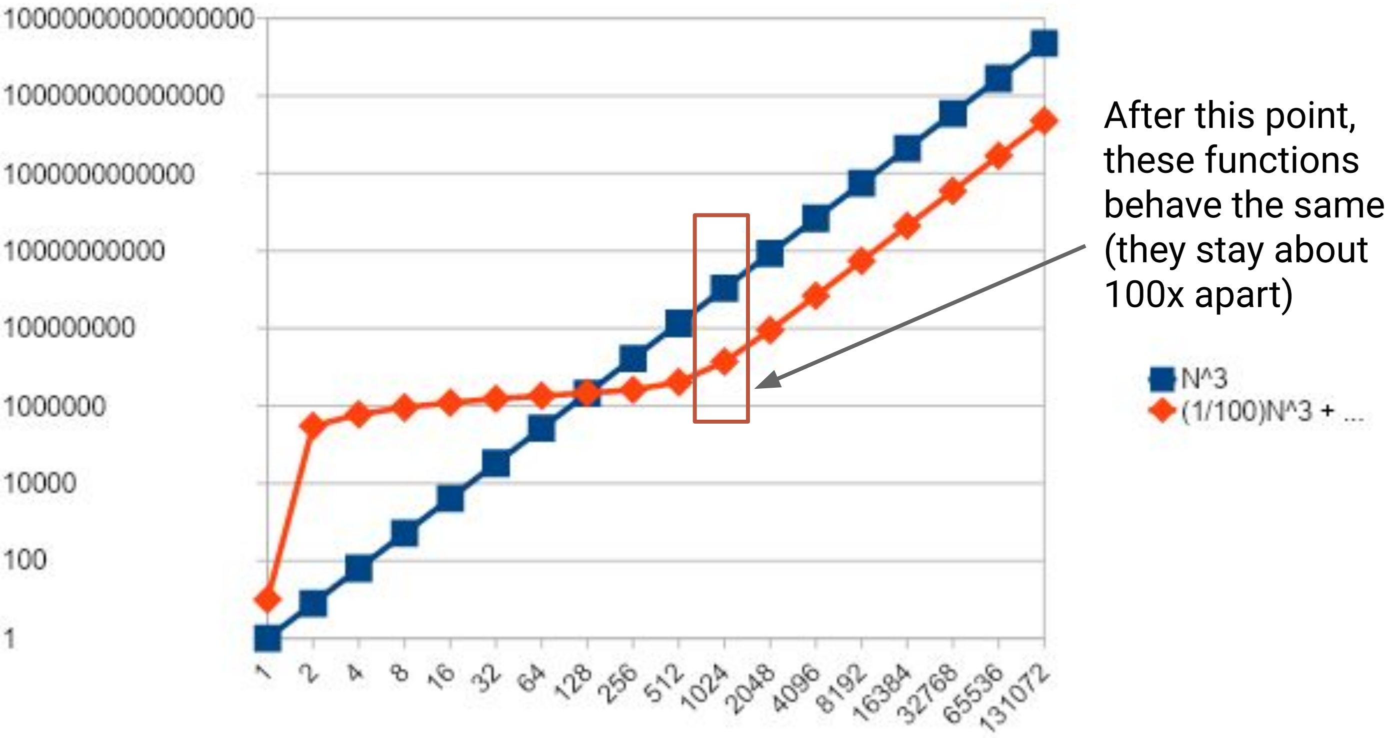
$$\frac{1}{100}n^3 + 10n + 1000000\log(n)$$

$$n^3$$

How Do We Capture Behavior at Scale?



How Do We Capture Behavior at Scale?



After this point, these functions behave the same (they stay about 100x apart)

How Do We Capture Behavior at Scale?

$$\lim_{n \to \infty} \frac{\frac{1}{100}n^3 + 10n + 1000000\log(n)}{n^3}$$

How Do We Capture Behavior at Scale?

$$\lim_{n \to \infty} \frac{\frac{1}{100}n^3 + 10n + 1000000\log(n)}{n^3}$$

$$= \lim_{n \to \infty} \frac{\frac{1}{100}n^3}{n^3} + \frac{10n}{n^3} + \frac{1000000\log(n)}{n^3}$$

How Do We Capture Behavior at Scale?

$$\lim_{n\to\infty} \frac{\frac{1}{100}n^3 + 10n + 1000000\log(n)}{n^3}$$

$$= \lim_{n\to\infty} \frac{\frac{1}{100}n^3}{n^3} + \frac{10n}{n^3} + \frac{1000000\log(n)}{n^3}$$

$$= \lim_{n\to\infty} \frac{1}{100} + \boxed{\lim_{n\to\infty} \frac{10}{n^2}} + \boxed{\lim_{n\to\infty} \frac{1000000\log(n)}{n^3}}$$

These terms go to 0

How Do We Capture Behavior at Scale?

$$\lim_{n\to\infty} \frac{\frac{1}{100}n^3 + 10n + 1000000\log(n)}{n^3}$$

$$= \lim_{n\to\infty} \frac{\frac{1}{100}n^3}{n^3} + \frac{10n}{n^3} + \frac{1000000\log(n)}{n^3}$$

$$= \lim_{n\to\infty} \frac{1}{100} + \lim_{n\to\infty} \frac{10}{n^2} + \lim_{n\to\infty} \frac{1000000\log(n)}{n^3}$$

$$= \frac{1}{100}$$

Attempt #3: Asymptotic Analysis

Consider two functions, f(n) and g(n)

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$$

In this particular case, f grows w.r.t. n faster than g

So…if f(n) and g(n) are the number of steps two different algorithms take on a problem of size n, which is better?

Attempt #3: Asymptotic Analysis

Case 1: $\lim_{n\to\infty}\frac{f(n)}{g(n)} = \infty$ *(f grows faster; g is better)*

Case 2: $\lim_{n\to\infty}\frac{f(n)}{g(n)} = 0$ *(g grows faster; f is better)*

Case 3: $\lim_{n\to\infty}\frac{f(n)}{g(n)} = some\ \ constant$ *(f and g "behave" the same)*

Goal of “Asymptotic Analysis”

We want to organize runtimes (growth functions) into different ***Complexity Classes***

Within the same complexity class, runtimes “behave the same”

Goal of "Asymptotic Analysis"

"Strategic Optimization" focuses on improving the complexity class of your code!

Back to Our Previous Example...

$$\frac{1}{100}n^3 + 10n + 1000000\log(n)$$

The 10n and 1000000 log(n) “don’t matter”

The 1/100 “does not matter”

Back to Our Previous Example...

$$\frac{1}{100}n^3 + 10n + 1000000\log(n)$$

The 10n and 1000000 log(n) “don’t matter”

The 1/100 “does not matter”

n^3 is the dominant term, and that determines the “behavior”

Why Focus on Dominating Terms?

$f(n)$	**10**	**20**	**50**	**100**	**1000**
$log(log(n))$	0.43 ns	0.52 ns	0.62 ns	0.68 ns	0.82 ns
$log(n)$	0.83 ns	1.01 ns	1.41 ns	1.66 ns	2.49 ns
n	2.5 ns	5 ns	12.5 ns	25 ns	0.25 µs
$nlog(n)$	8.3 ns	22 ns	71 ns	0.17 µs	2.49 µs
n^2	25 ns	0.1 µs	0.63 µs	2.5 µs	0.25 ms
n^5	25 µs	0.8 ms	78 ms	2.5 s	**2.9 days**
2^n	0.25 µs	0.26 ms	**3.26 days**	**10^{13} years**	**10^{284} years**
$n!$	0.91 ms	**19 years**	**10^{47} years**	**10^{141} years**	🤯

Why Focus on Dominating Terms?

$$2^n \gg n^c \gg n \gg log(n) \gg c$$