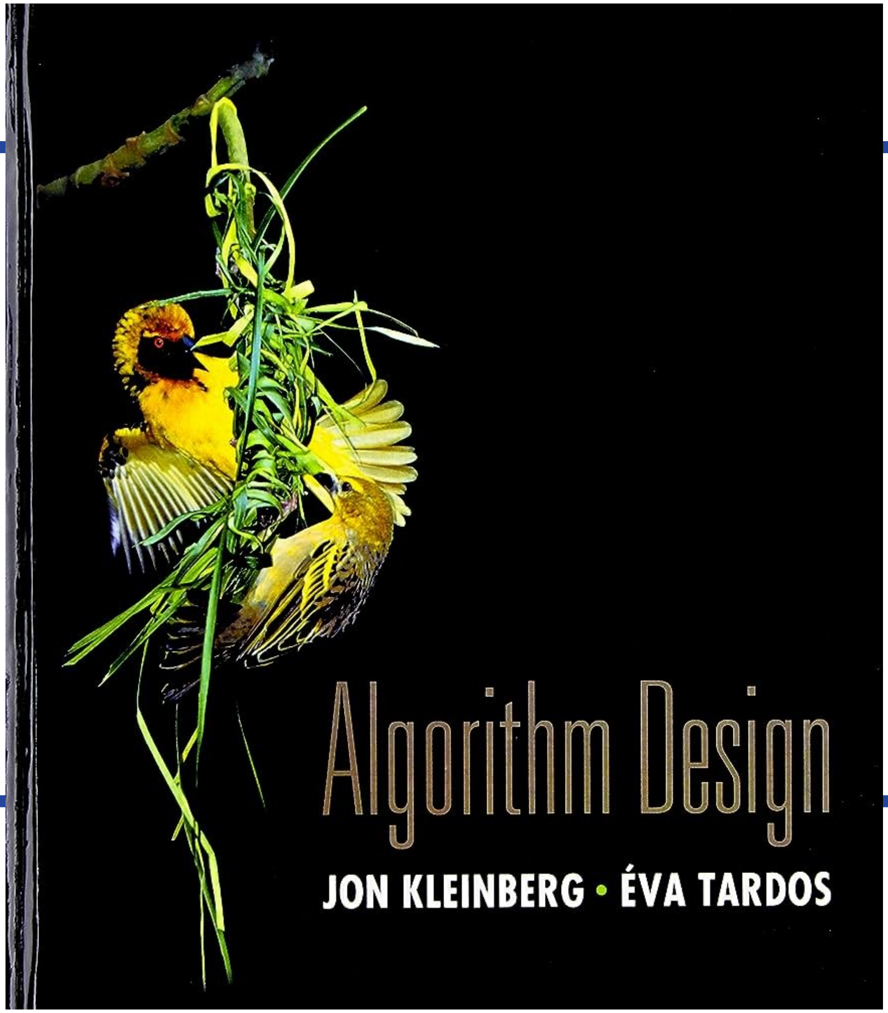
CS 310: Algorithms

Lecture 4

Instructor: Naveed Anwar Bhatti



Chapter 2:
Basics of Algorithm Analysis

Reasons to analyze algorithms

- Predict performance
- Compare algorithms
- Provide guarantees
- Improve performance
- Understand theoretical basis

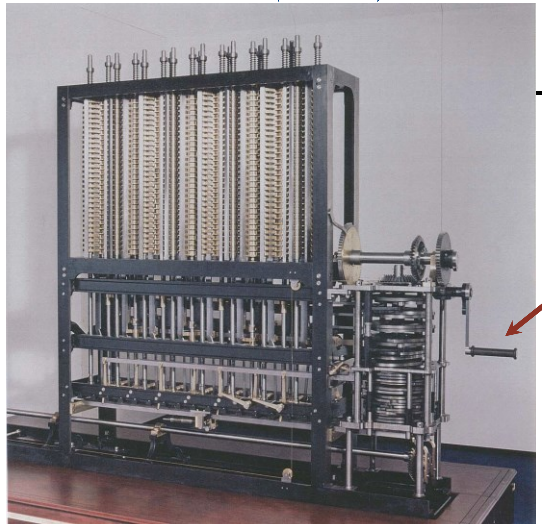
Primary practical reason: **avoid performance bugs.**

client gets poor performance because programmer
did not understand performance characteristics

A strikingly modern thought

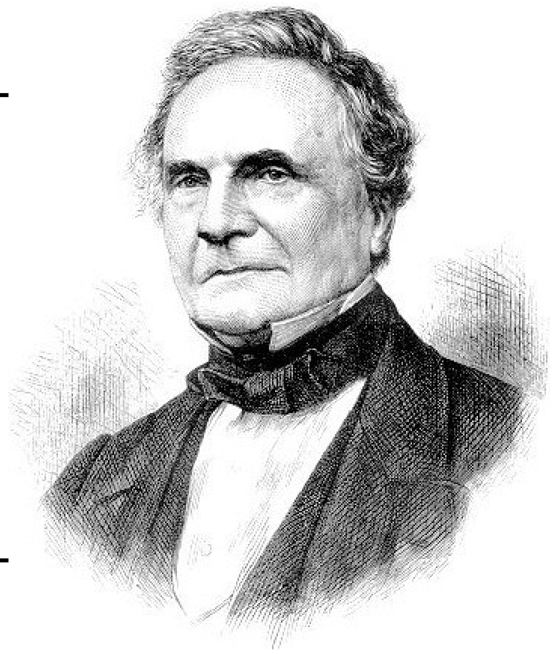
" As soon as an Analytic Engine exists, it will necessarily guide the future course of the science. Whenever any result is sought by its aid, the question will arise—By what course of calculation can these results be arrived at by the machine in the shortest time? " — Charles Babbage (1864)

Efficiency?



how many times do you have to turn the crank?

Analytic Engine



How can we define efficiency?

"An algorithm is efficient if, when implemented, it runs quickly on real input instances."

How can we measure efficiency? (Running Time)

Empirical Analysis

Mathematical Models

Asymptotic Models

Asymptotic Order of Growth

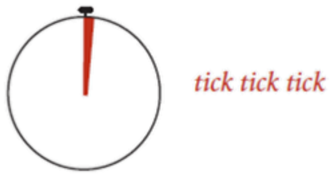
Empirical Analysis

Problem: Find a triplet in an array whose sum is **Zero**

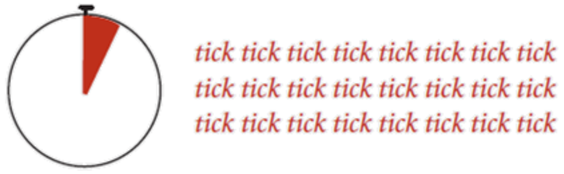
Solution:
```
for (int i = 0; i < N; i++)
    for (int j = i+1; j < N; j++)
        for (int k = j+1; k < N; k++)
            if (a[i] + a[j] + a[k] == 0)
                count++;
```

Experimental Setup:

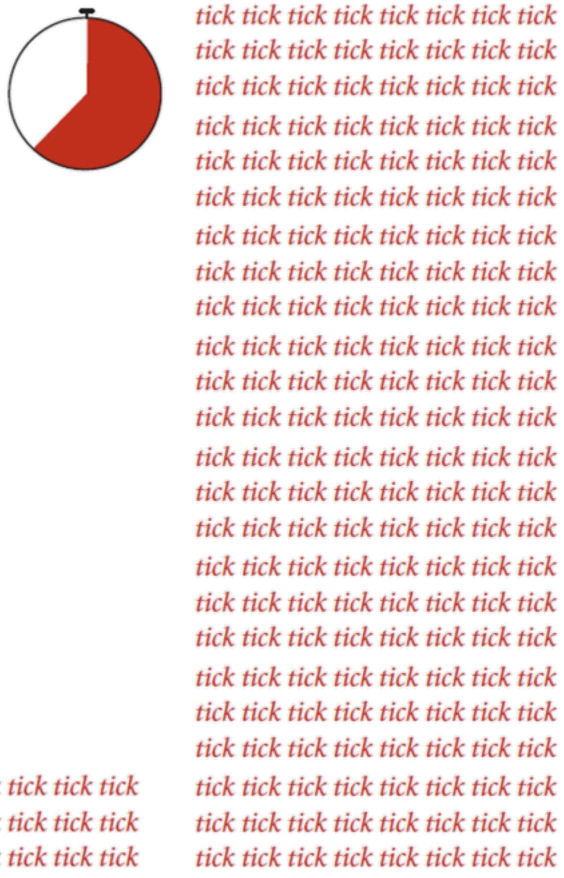
1K array



2K array


tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick

4K array


tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick

Empirical Analysis

System independent effects.

- Algorithm.
- Input data.

We are interested in measuring this

System dependent effects.

- Hardware: CPU, memory, cache, …
- Software: compiler, operating system, garbage collector, …

But we are also measuring this

Bad News: Sometimes difficult to get precise measurements.

Good News: We can generate accurate mathematical models

Mathematical Model: Examples

Problem | **Algorithm** | **Mathematical Model**

Find a index in an array whose value is **Zero**

```
int count = 0;
for (int i = 0; i < N; i++)
      if (a[i] == 0)
          count++;
```

$$n$$

Find a twins in an array whose value is **Zero**

```
int count = 0;
for (int i = 0; i < N; i++)
      for (int j = i+1; j < N; j++)
           if (a[i] + a[j] == 0)
               count++;
```

$$n(n-1)$$
$$= n^2 - n$$

Find a triplet in an array whose sum is **Zero**

```
for (int i = 0; i < N; i++)
   for (int j = i+1; j < N; j++)
      for (int k = j+1; k < N; k++)
          if (a[i] + a[j] + a[k] == 0)
               count++;
```

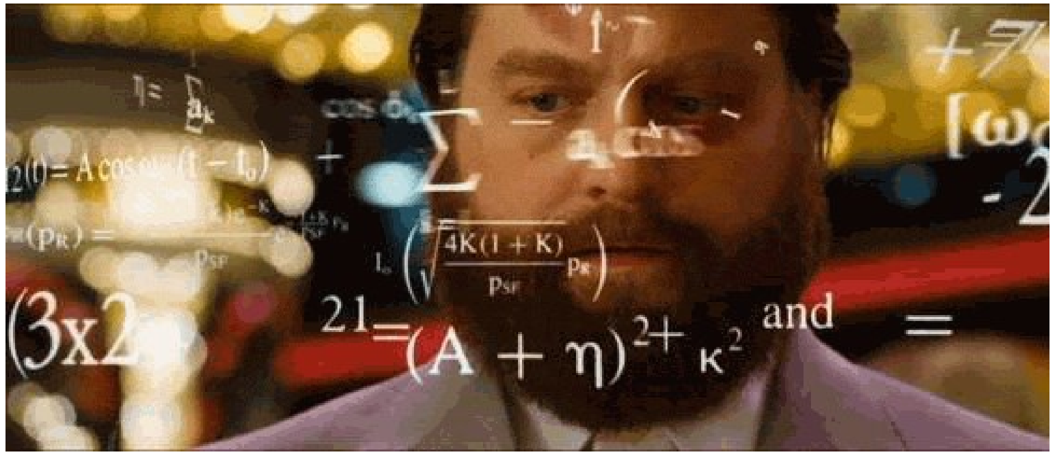
$$n(n-1)\,(n-2)$$
$$= n^3 - 3n^2 + 2n$$

Mathematical Model: We need *Simplification*

In principle, accurate mathematical models are available.

In practice,

- Formulas can be **complicated.**
- Advanced mathematics might be required.
- Exact models best left for experts.



Last option left: Use **Asymptotic** models

Section 2.2:
Asymptotic Order of Growth

Asymptotic Model:

Estimate running time (or memory) as a function of input size ***N***.

- Ignore **lower order terms**.
- Ignore **constants**
- when ***N* is large**, terms are negligible
- when ***N* is small**, we don't care

Asymptotic Model: Examples

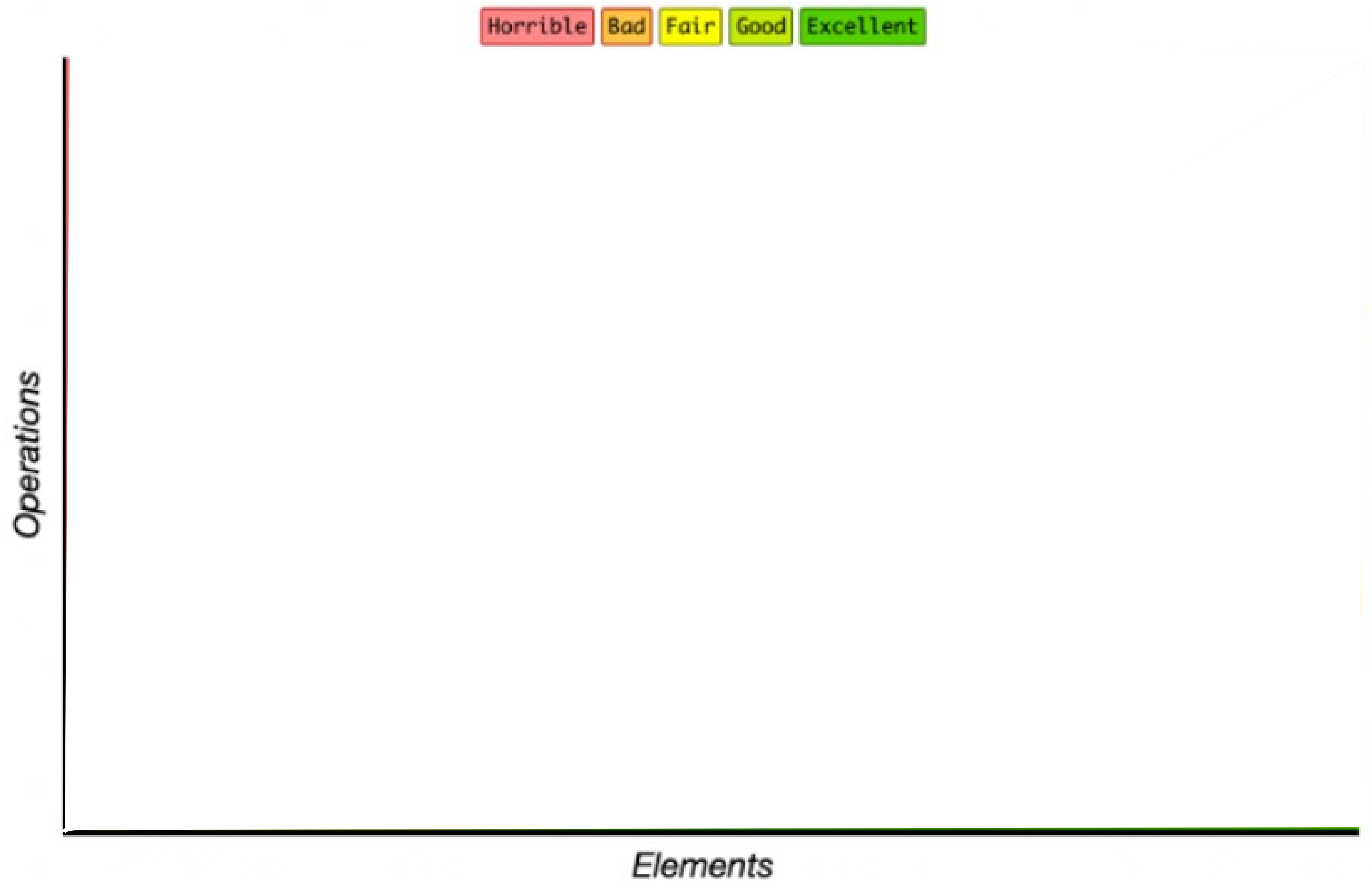
Problem	Algorithm	Mathematical Model	Asymptotic Model
Find a index in an array whose value is **Zero**	`int count = 0; for (int i = 0; i < N; i++) if (a[i] == 0) count++;`	n	$\sim n$
Find a twins in an array whose value is **Zero**	`int count = 0; for (int i = 0; i < N; i++) for (int j = i+1; j < N; j++) if (a[i] + a[j] == 0) count++;`	$n(n-1)$ $= n^2 - n$	$\sim n^2$
Find a triplet in an array whose sum is **Zero**	`for (int i = 0; i < N; i++) for (int j = i+1; j < N; j++) for (int k = j+1; k < N; k++) if (a[i] + a[j] + a[k] == 0) count++;`	$n(n-1)(n-2)$ $= n^3 - 3n^2 + 2n$	$\sim n^3$

Common Asymptotic order-of-growth classifications

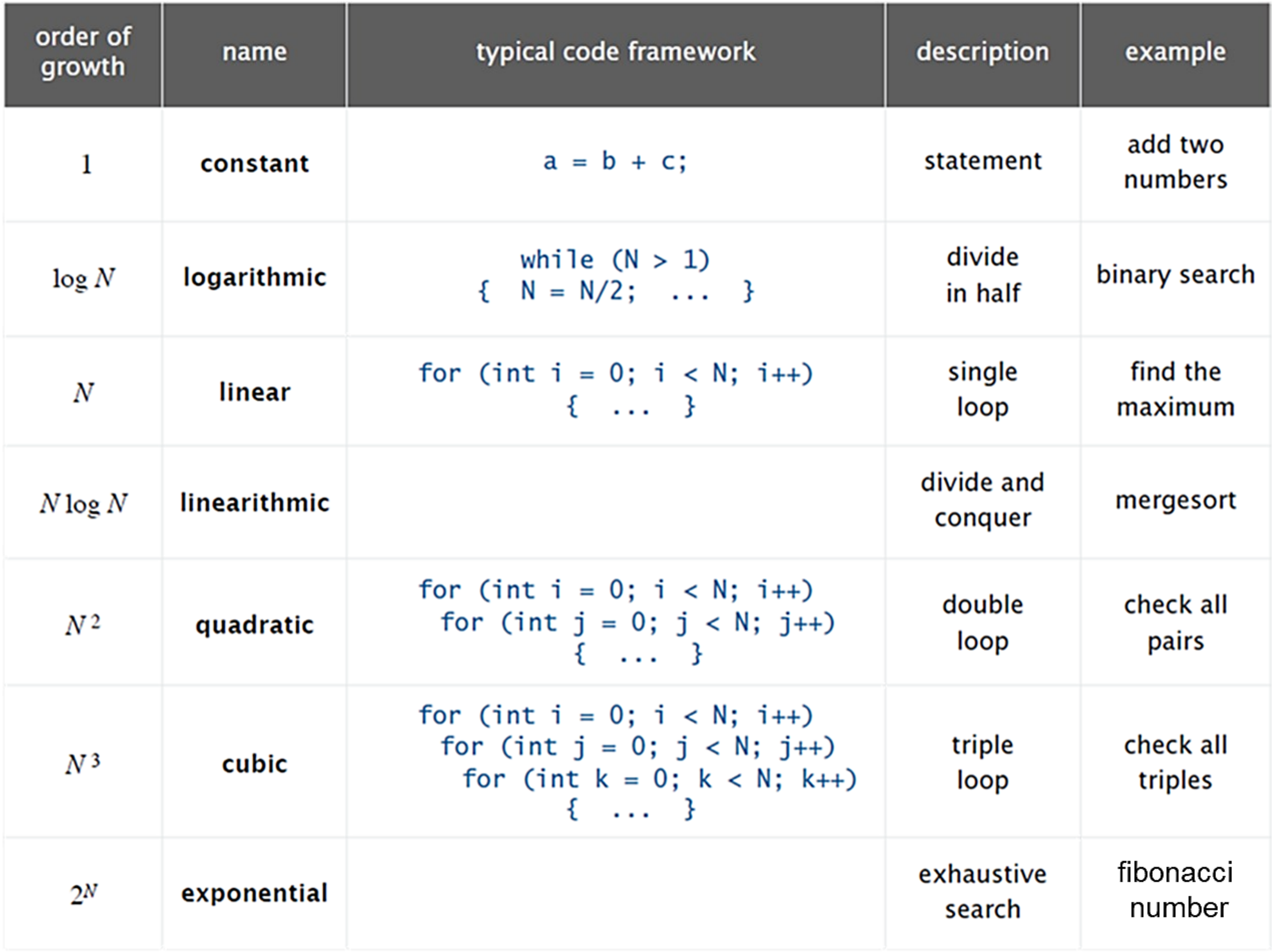
Good News: The set of functions

$\mathbf{1}$, $\mathbf{\log N}$, $\boldsymbol{N}$, $\boldsymbol{N \log N}$, $\boldsymbol{N^2}$, $\boldsymbol{N^3}$, $\mathbf{2}^{\boldsymbol{N}}$ and $\boldsymbol{N}!$

suffice to describe the order of growth of **most common algorithms**



Common Asymptotic order-of-growth classifications



order of growth	name	typical code framework	description	example
1	**constant**	`a = b + c;`	statement	add two numbers

$\log N$	**logarithmic**	`while (N > 1)`
		`{ N = N/2; ... }`
			divide in half	binary search

N	**linear**	`for (int i = 0; i < N; i++)`
		`{ ... }`
			single loop	find the maximum

$N \log N$	**linearithmic**		divide and conquer	mergesort

N^2	**quadratic**	`for (int i = 0; i < N; i++)`
		`for (int j = 0; j < N; j++)`
		`{ ... }`
			double loop	check all pairs

N^3	**cubic**	`for (int i = 0; i < N; i++)`
		`for (int j = 0; j < N; j++)`
		`for (int k = 0; k < N; k++)`
		`{ ... }`
			triple loop	check all triples

2^N	**exponential**		exhaustive search	fibonacci number

Algorithm Analysis: Live Poll 1

Suppose you have algorithms with the running time of **n^2** *(Assume these are the exact number of operations performed as a function of the input size **n**)* and you have a computer that can perform **10^{10} operations per second**. You need to compute a result in at most **an hour of computation**.

What is the largest input size **n** for which you would be able to get the result within an hour?

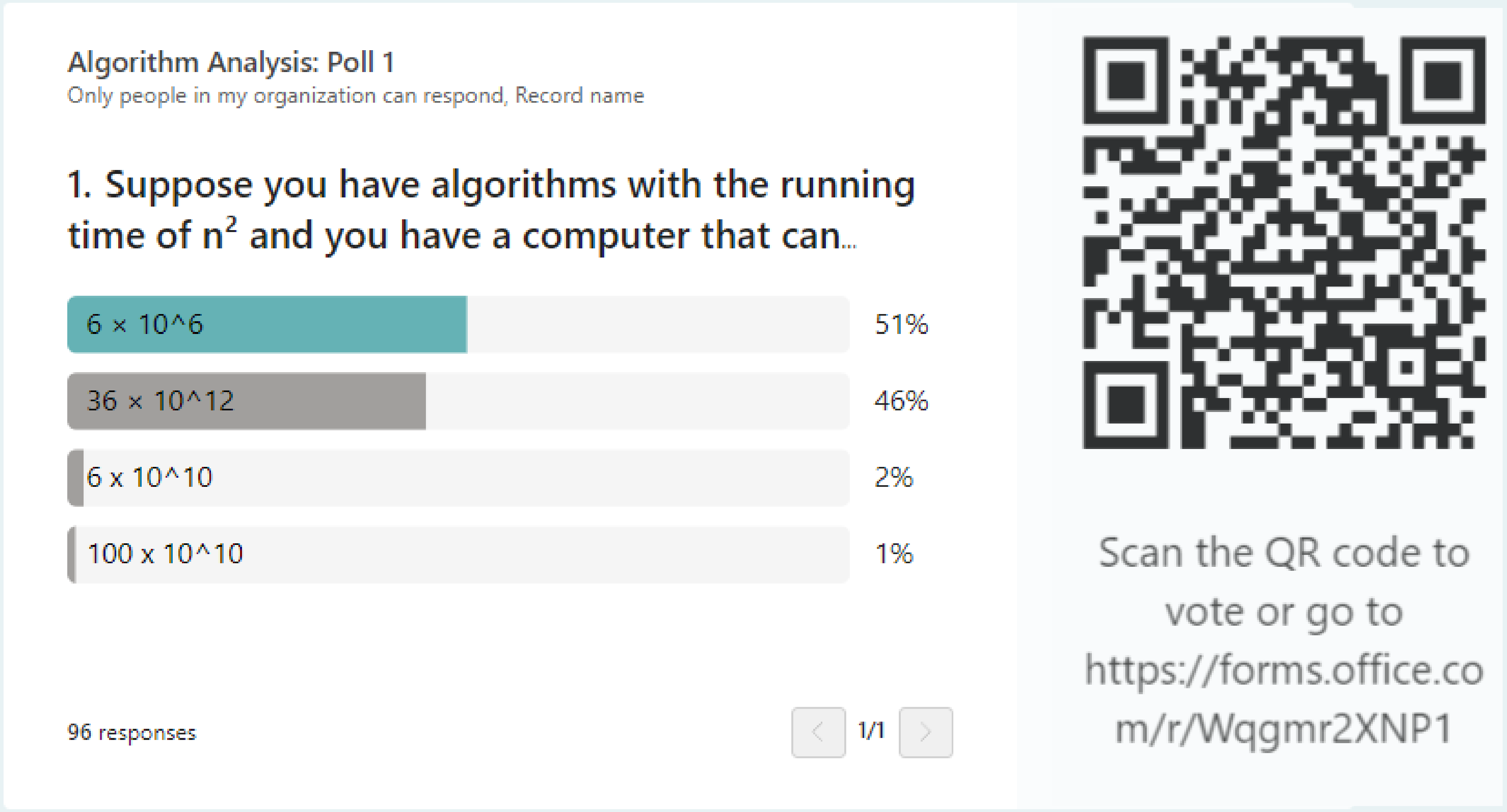
A. 6×10^6

B. 36×10^{12}

C. 6×10^{10}

D. 100×10^{10}

Scan the QR code to vote or go to https://forms.office.com/r/Wqgmr2XNP1



Algorithm Analysis: Poll 1

Only people in my organization can respond, Record name

1. Suppose you have algorithms with the running time of n^2 and you have a computer that can...

6 × 10^6	51%
36 × 10^12	46%
6 x 10^10	2%
100 x 10^10	1%

96 responses

1/1

Scan the QR code to vote or go to https://forms.office.com/r/Wqgmr2XNP1

Asymptotic Order of Growth : Live Poll 1

First, let's find out the total number of operations the computer can perform in an hour:

1 hour = 60 minutes = 3600 seconds

Operations per second= $\mathbf{10^{10}}$

Total operations in an hour = **3600 x** $\mathbf{10^{10}}$

We need $\boldsymbol{n^2} \leq$ **3600 x** $\mathbf{10^{10}}$

Taking the square root of both sides,

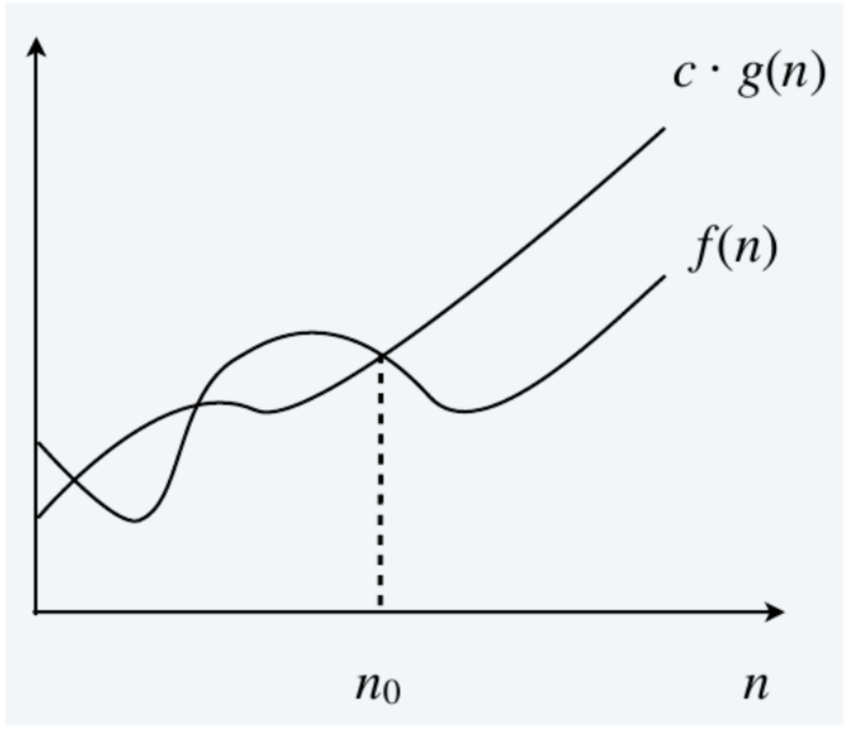
$\boldsymbol{n} \quad \leq$ **60 x** $\mathbf{10^{5}}$

Scan the QR code to vote or go to https://forms.office.com/r/Wqgmr2XNP1

O, Ω, and Θ

Asymptotic Order of Growth – Big O notation

- ***Upper bounds:*** ***f*****(n)** is **O(g(n))** if there exist constants c **>** 0 and $n_0 \geq 0$ such that for all $n \geq n_0$ we have $f(n) \leq c \cdot g(n)$.



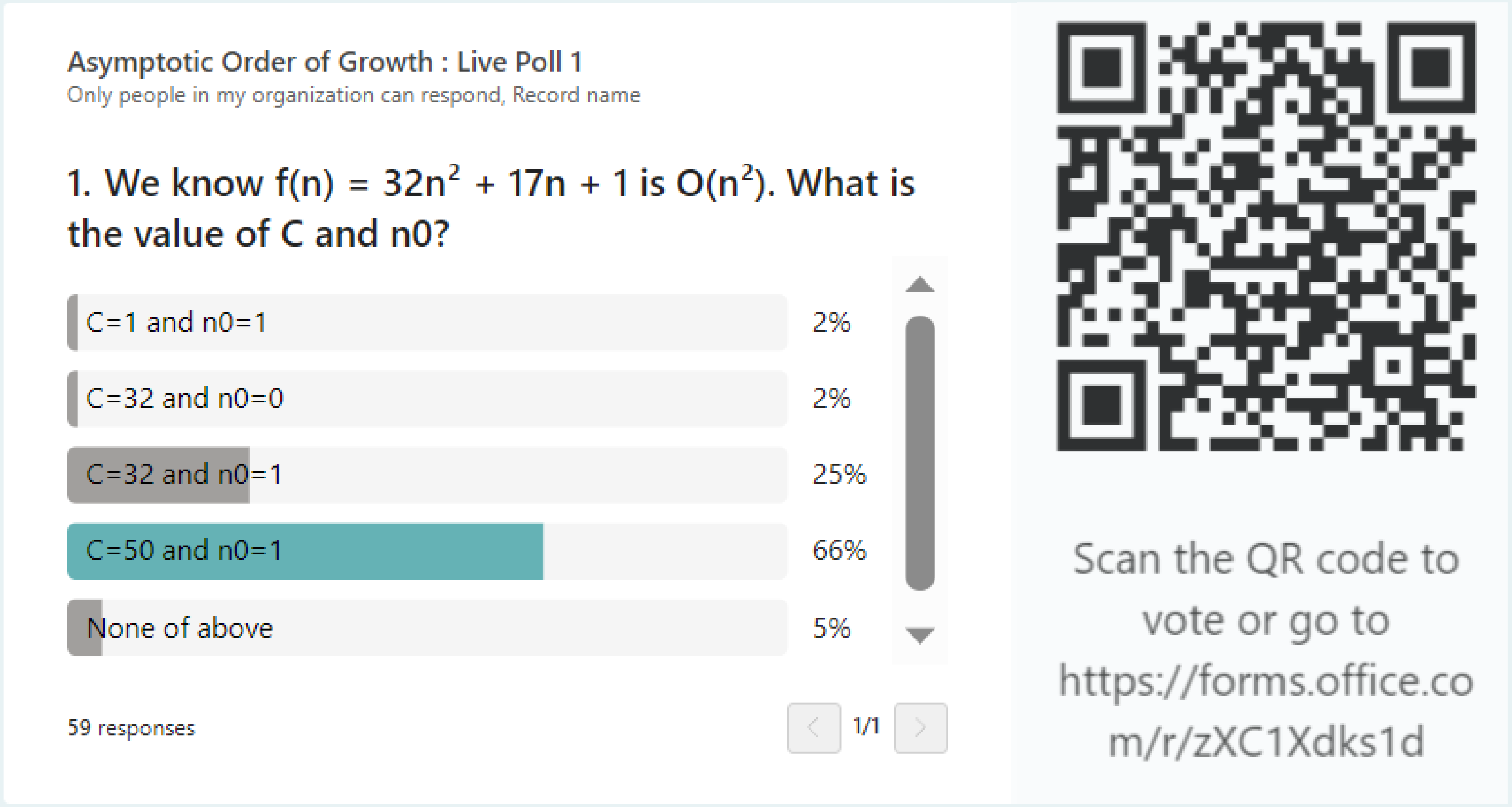
- Ex: $f(n) = 32n^2 + 17n + 1$
  - f(n) is O(n^2)
  - **Can we say** f(n) is O(n^3) ?

Asymptotic Order of Growth : Live Poll 1

We know **f(n)** = $32n^2 + 17n + 1$ is **O(n^2).** What is the value of **C** and **n0?**

A. C=1 and n0=1
B. C=32 and n0=0
C. C=32 and n0=1
D. C=50 and n0=1
E. None of above

Scan the QR code to vote or go to https://forms.office.com/r/zXC1Xdks1d



Asymptotic Order of Growth : Live Poll 1

Only people in my organization can respond, Record name

1. We know $f(n) = 32n^2 + 17n + 1$ is $O(n^2)$. What is the value of C and n0?

C=1 and n0=1	2%
C=32 and n0=0	2%
C=32 and n0=1	25%
C=50 and n0=1	66%
None of above	5%

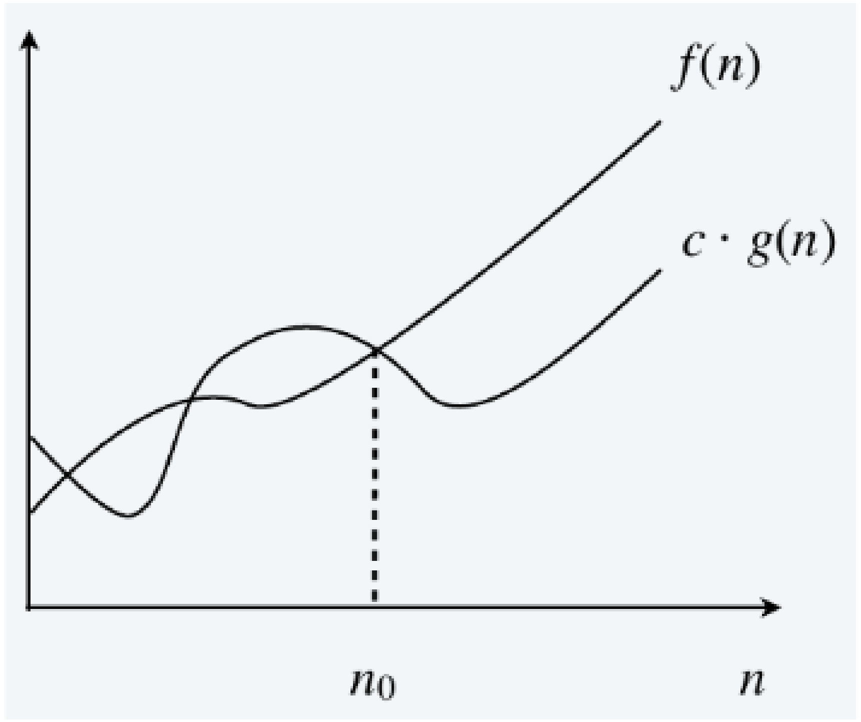
59 responses

1/1

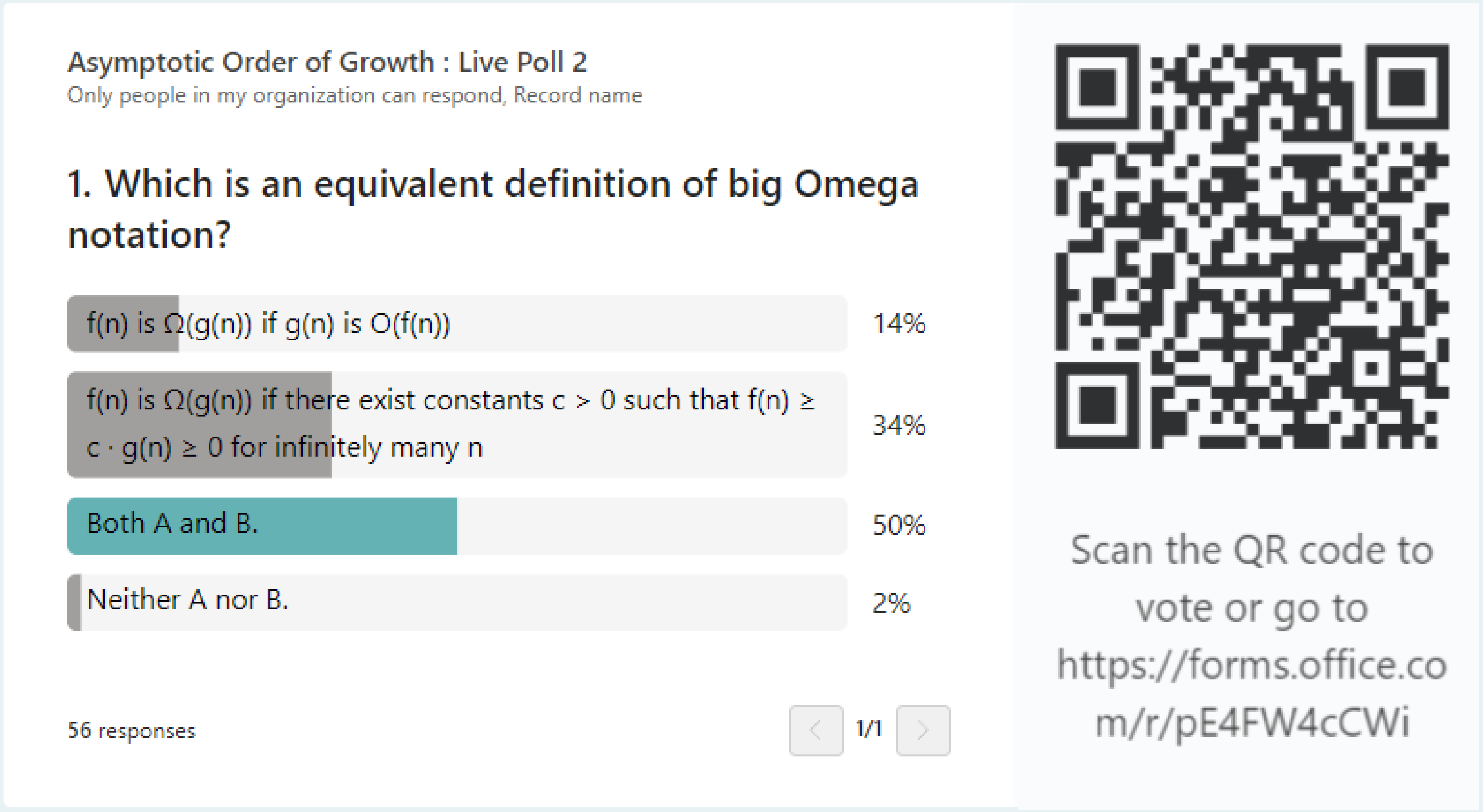
Scan the QR code to vote or go to https://forms.office.com/r/zXC1Xdks1d

Asymptotic Order of Growth – Big Omega notation

- Lower bounds. ***f*(n)** is **Ω(g(n))** if there exist constants $c > 0$ and $n_0 \geq 0$ such that for all $n \geq n_0$ we have $f(n) \geq c \cdot g(n)$.



- Ex: $f(n) = 32n^2 + 17n + 1$
  - $f(n)$ is $\Omega(n^2)$, $\Omega(n)$



Asymptotic Order of Growth : Live Poll 2

Only people in my organization can respond, Record name

1. Which is an equivalent definition of big Omega notation?

f(n) is Ω(g(n)) if g(n) is O(f(n))	14%
f(n) is Ω(g(n)) if there exist constants c > 0 such that f(n) ≥ c · g(n) ≥ 0 for infinitely many n	34%
Both A and B.	50%
Neither A nor B.	2%

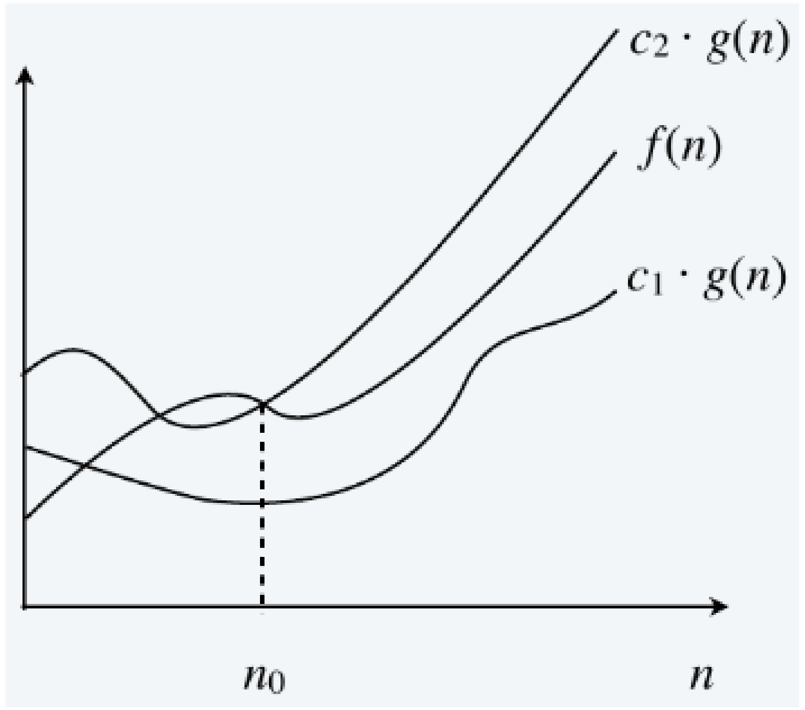
56 responses

1/1

Scan the QR code to vote or go to https://forms.office.com/r/pE4FW4cCWi

Asymptotic Order of Growth – Big Theta notation

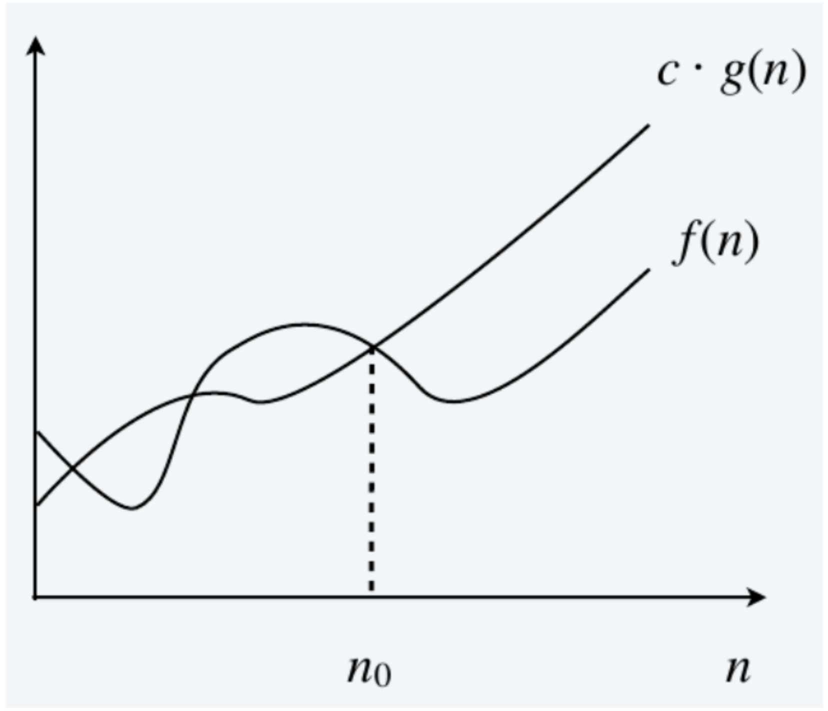
- Tight bound. ***f*(n)** is **Θ(g(n))** if there exist constants $c_1 > 0$, $c_2 > 0$ and $n_0 \geq 0$ such that for all $n \geq n_0$ we have $\mathbf{c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)}$.



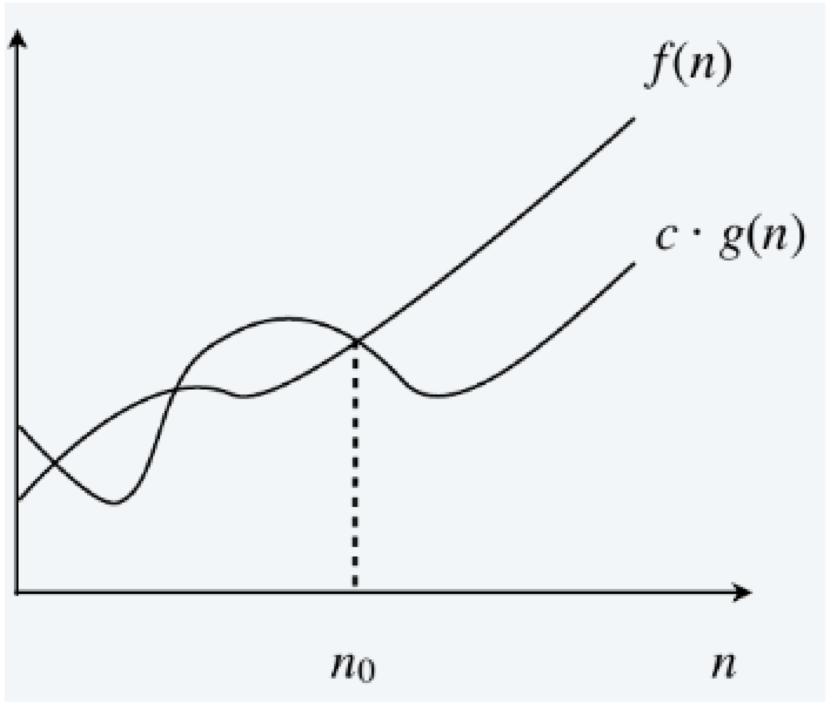
- Ex: $f(n) = 32n^2 + 17n + 1$
  - $f(n)$ is $\Theta(n^2)$

Asymptotic Order of Growth

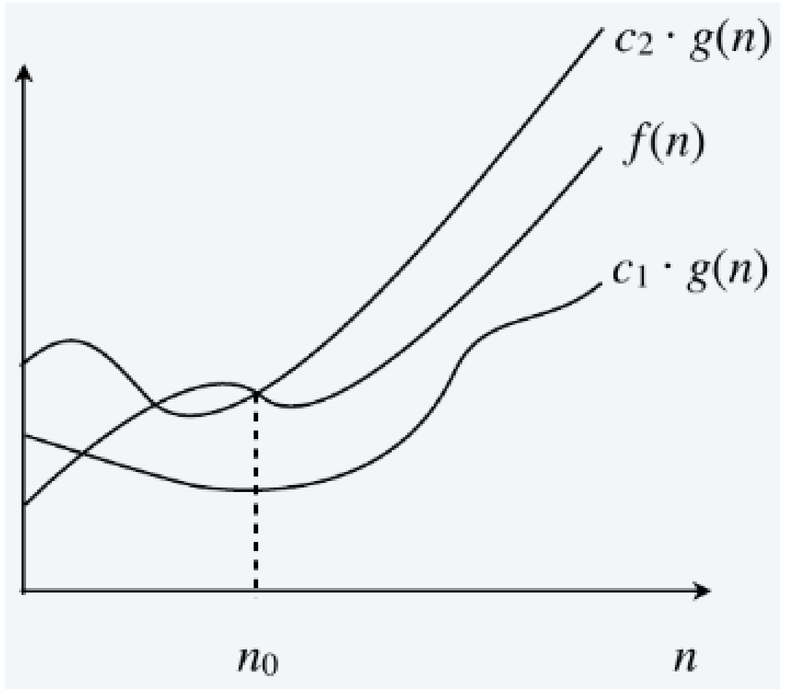
O



Ω



Θ



Notation

- Slight abuse of notation. $T(n) = O(f(n))$.
  - Asymmetric:
    - $f(n) = 5n^3$; $g(n) = 3n^2$
    - $f(n) = O(n^3) = g(n)$
    - but $f(n) \neq g(n)$.
  - Better notation: $T(n) \in O(f(n))$.

- Meaningless statement. Any comparison-based sorting algorithm requires at least $O(n \log n)$ comparisons.
  - Statement doesn't "type-check."
  - Use Ω for lower bounds.

Properties

- Transitivity.
  - If f = O(g) and g = O(h) then f = O(h).
  - If f = Ω(g) and g = Ω(h) then f = Ω(h).
  - If f = Θ(g) and g = Θ(h) then f = Θ(h).

- Additivity.
  - If f = O(h) and g = O(h) then f + g = O(h).
  - If f = Ω(h) and g = Ω(h) then f + g = Ω(h).
  - If f = Θ(h) and g = Θ(h) then f + g = Θ(h).

Asymptotic Bounds for Some Common Functions

- Polynomials. $a_0 + a_1 n + \ldots + a_d n^d$ is $\Theta(n^d)$ if $a_d > 0$.

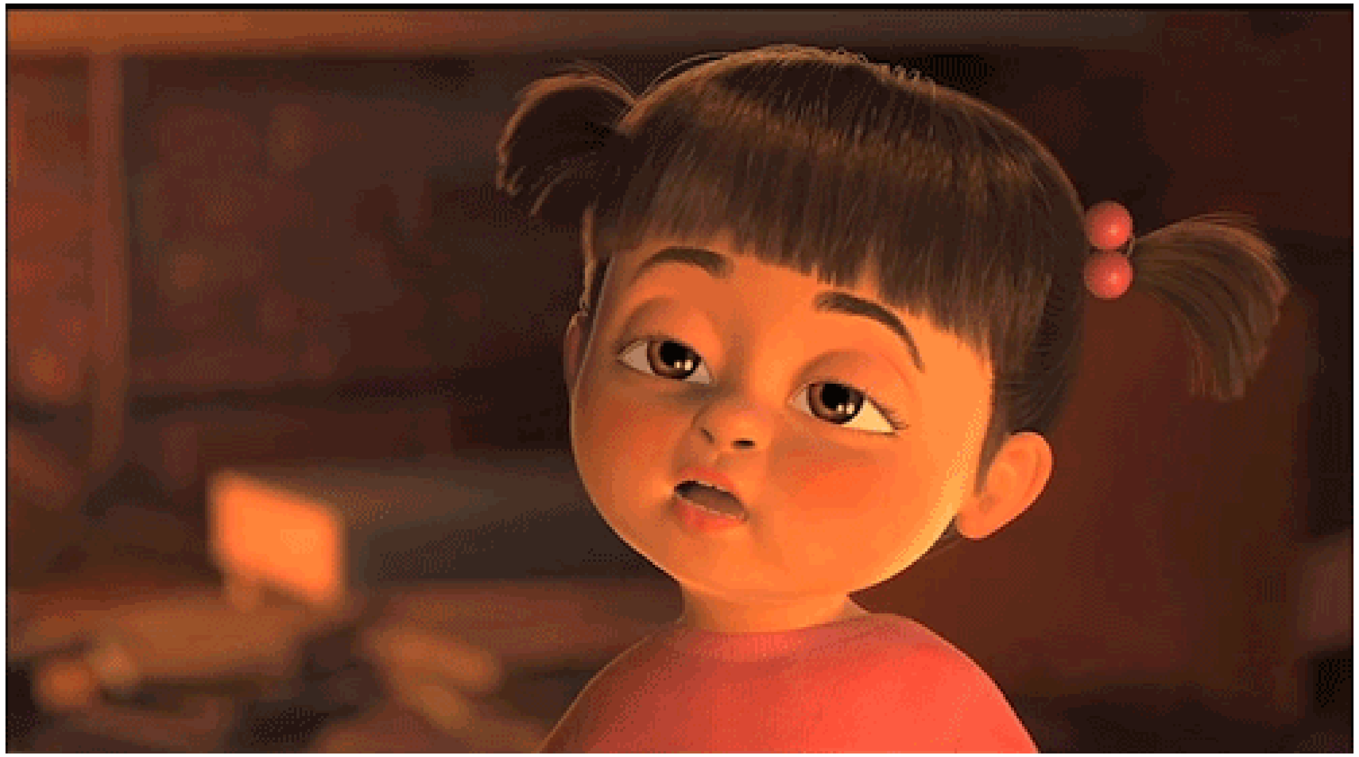
- Polynomial time. Running time is $O(n^d)$ for some constant d independent of the input size n.

- Logarithms. $O(\log_a n) = O(\underset{\substack{\uparrow \\ \text{can avoid specifying the base}}}{\log_b} n)$ for any constants a, b > 0.

- Logarithms. For every x > 0, $\underset{\substack{\uparrow \\ \text{log grows slower than every polynomial}}}{\log n} = O(n^x)$.

- Exponentials. For every r > 1 and every d > 0, $\underset{\substack{\uparrow \\ \text{every exponential grows faster than every polynomial}}}{n^d} = O(r^n)$.

Thanks a lot



If you are taking a Nap, **wake up**........Lecture Over