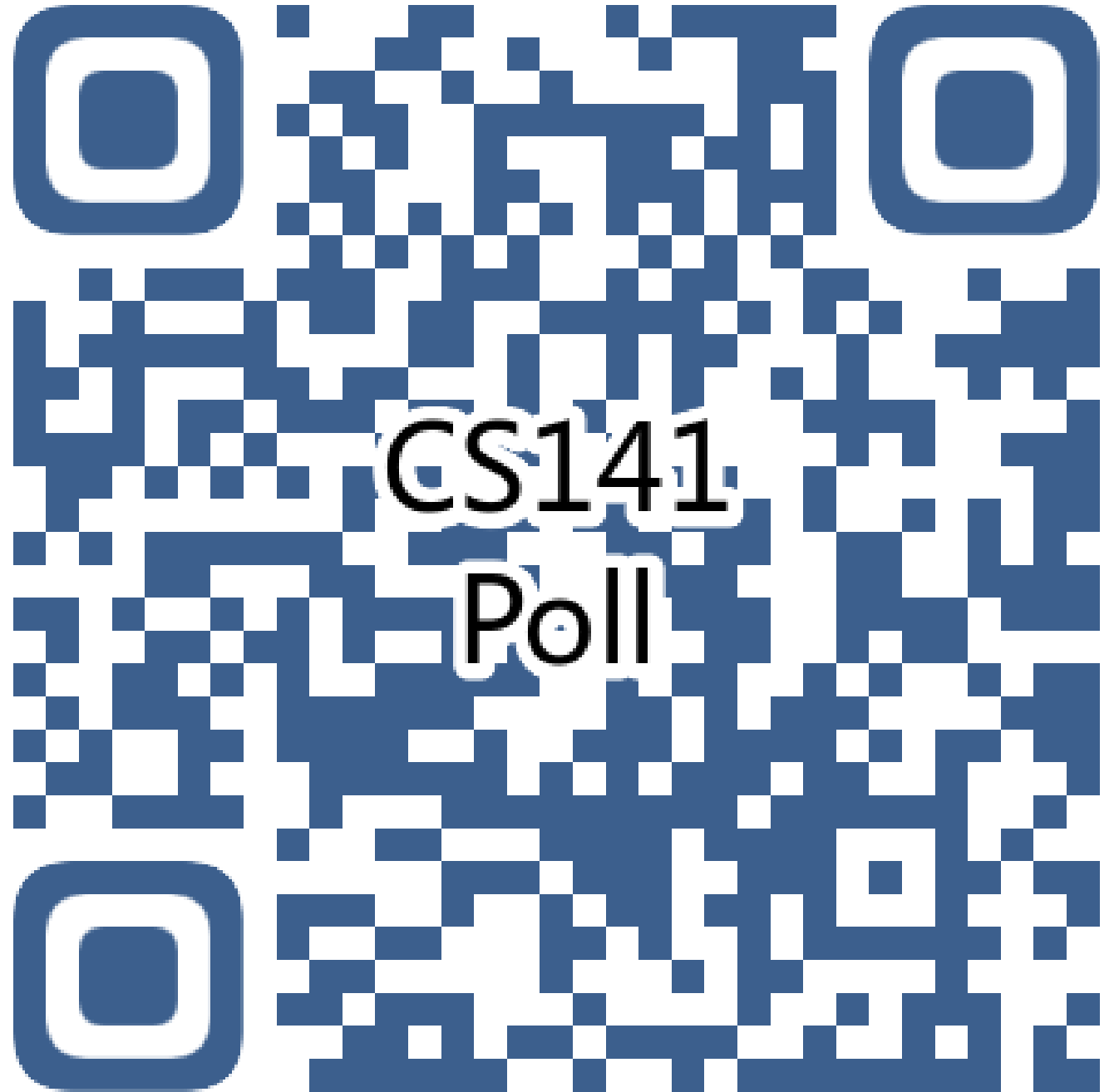


Poll link

- <https://PollEv.com/yihansun724>
- Or
- Text YIHANSUN724 to 37607 to join the session



Analysis of Algorithms

Yihan Sun

Analyzing algorithms

- Predict how your algorithm performs in practice

- **Criteria**

- Running time ← Time complexity
- Space usage
- Cache I/O
- Disk I/O
- Network bandwidth
- Power consumption
- Lines of codes

What are good algorithms?

- **Tale about Gauss**



One day Gauss's teacher asked his class to add together all the numbers from 1 to 100, assuming that this task would occupy them for quite a while. He was shocked when young Gauss, after a few seconds thought, wrote down the answer 5050. (source: <https://nrich.maths.org/2478>)

Other students:

```
sum = 0;  
for (int i = 1; i <= n; i++)  
    sum += i;
```

$O(n)$ time complexity

Why we say the complexity is $O(n)$?

Gauss:

$\text{sum} = (1+n)*n/2;$

$O(1)$ time complexity

$$\begin{array}{cccccccccccc} 1 & + & 2 & + & 3 & + & 4 & + & 5 & + & 6 & + & 7 & + & 8 & + & 9 & + & 10 & = & ? \\ \hline & & & & & & & & & + & & & & & & & & & & & \\ & & & & & & & & & + & & & & & & & & & & & \\ & & & & & & & & & + & & & & & & & & & & & \\ & & & & & & & & & + & & & & & & & & & & & \\ & & & & & & & & & + & & & & & & & & & & & \end{array}$$
$$(1+10) + (2+9) + (3+8) + (4+7) + (5+6) = ?$$

Analyze running time

- Random Access Machine (RAM) model

- Every operation, including memory access, arithmetic operations, etc., takes **unit time**

```
sum = 0;  
for (int i = 1; i <= n; i++)  
    sum += i;
```

$3n+2$ operations?

```
sum = (1+n)*n/2;
```

3 operations?

Analyze running time

- **Random Access Machine (RAM) model**
 - Every operation, including memory access, arithmetic operations, etc., takes **unit time**
- We only care about the **order of the cost**, e.g., we omit
 - Lower-order terms
 - Constants

Asymptotic Analysis

Why we use “asymptotical analysis”?

- Because we only care about how fast a function grows!



The ruler liked the chess very much, and wanted to give the inventor a prize. He could have anything he wished, the ruler said. The inventor requested that

- One grain of wheat be put in the first square
- Two grains of wheat in the second square.
- Four grains of wheat in the third
-

In general, the number of grains of wheat in one square is always the double of the previous square. The ruler laughed it off as an insignificant prize for his achievement and the ruler granted his wish. Later the court treasurers reported that the entire grain reserve of the kingdom was not sufficient to fulfill the request!

Grow exponentially (2^n)!

18,446,744,073,709,551,615

This is more wheat than in the entire whole world; in fact, it would fill a building 25 miles long, 25 miles wide, and 1000 feet tall.

Would you rather have a million dollars or one cent on day one, doubled every day for a month?

- **A. I want a million dollars immediately!!**
- **B. I want to have one cent in the first day, doubled every day for a month!**
- **Actually, even in February, for choice B you can get more than 1 million!**
- **Your total reward on day i is**
 - $1 + 2 + 4 + \dots + 2^{i-1}$
 - $= 2^i - 1$ cents $= (2^i - 1)/100$ dollars
- **Is it larger than 1,000,000 dollars?**
 - It will be for large enough i !

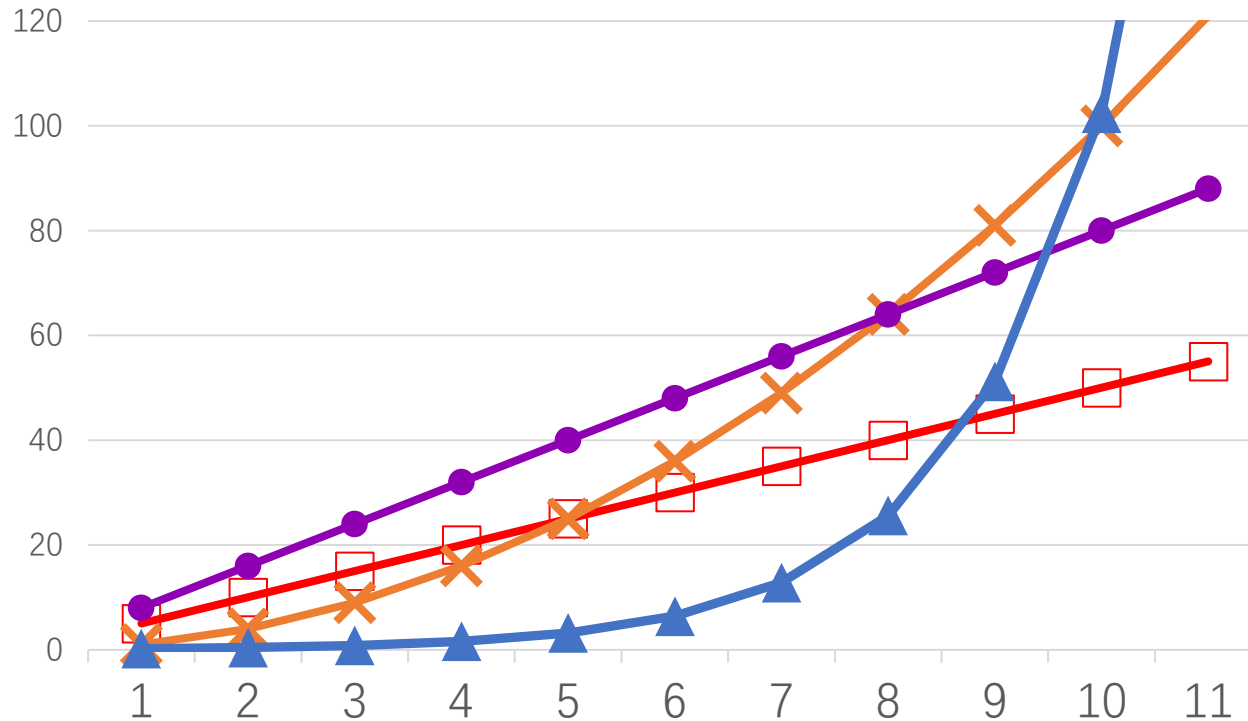
Growth of Functions

- The **growth** of the function is what interests us (**large data**)
 - For the cost function $f(n)$ we care when n is large enough
 - When n is small, $f(n)$ is small anyway. We care the analysis when the running time can be long
 - The **constant factors** and **lower-order terms** doesn't affect the growth of the function

$$f(n) = (2^n - 1)/100 \text{ vs. } g(n) = 1,000,000$$

When n is large enough, $f(n)$ will be much larger than $g(n)$

Growth of Functions



$$g_2(n) = 0.1 \times 2^i$$

$$g(n) = n^2$$

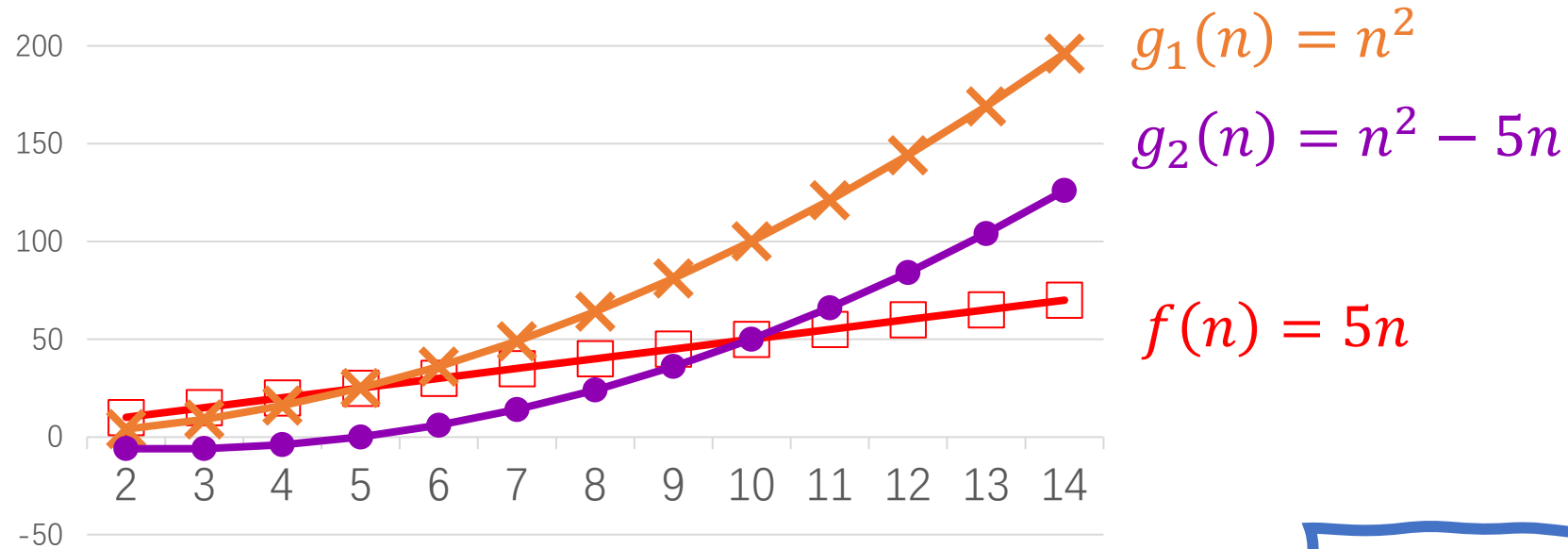
$$f_2(n) = 8n$$

$$f_1(n) = 5n$$

Omit the constant factors.

When n is large enough, $g(n)$ will be much larger than $f_1(n)$ or $f_2(n)$
 $f_1(n)$ and $f_2(n)$ will have similar growth trend

Growth of Functions



Omit the lower-order terms.

When n is large enough, $g_1(n)$ or $g_2(n)$ will still be much larger than $f(n)$
 $g_1(n)$ and $g_2(n)$ will have similar growth trend
because $-5n$ is much smaller compared to n^2

Asymptotic notation

```
sum = 0;  
for (int i = 1; i <= n; i++)  
    sum += i;
```

$3n+2$ operations?

$O(n)$ operations

```
sum = (1+n)*n/2;
```

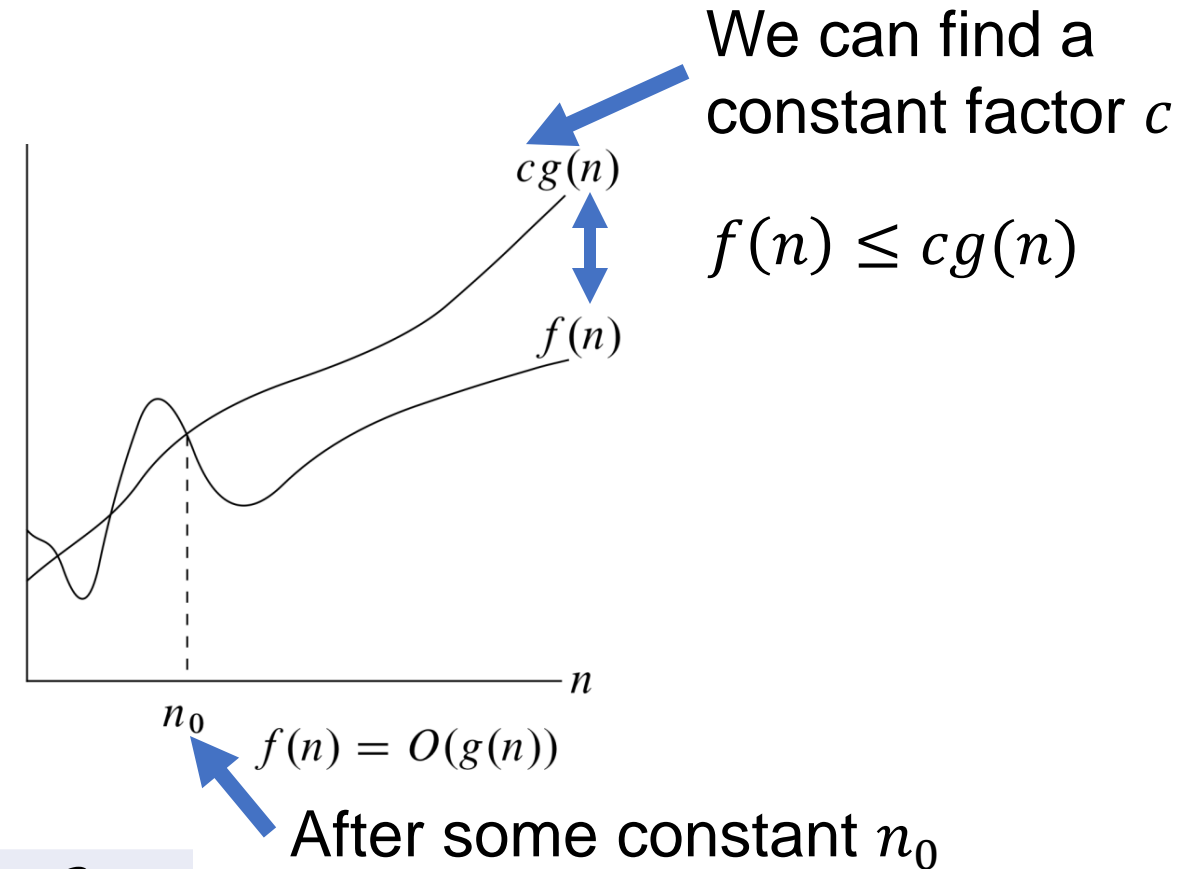
3 operations?

$O(1)$ operations

O (Big- O) $f(n) = O(g(n))$

- Asymptotic **upper bound**
- “ \leq ”: can be of the **same order**, but can also be “**smaller**”

$$\begin{aligned} \exists c > 0, n_0 > 0 \\ 0 \leq f(n) \leq cg(n) \\ \text{for } n \geq n_0 \end{aligned}$$

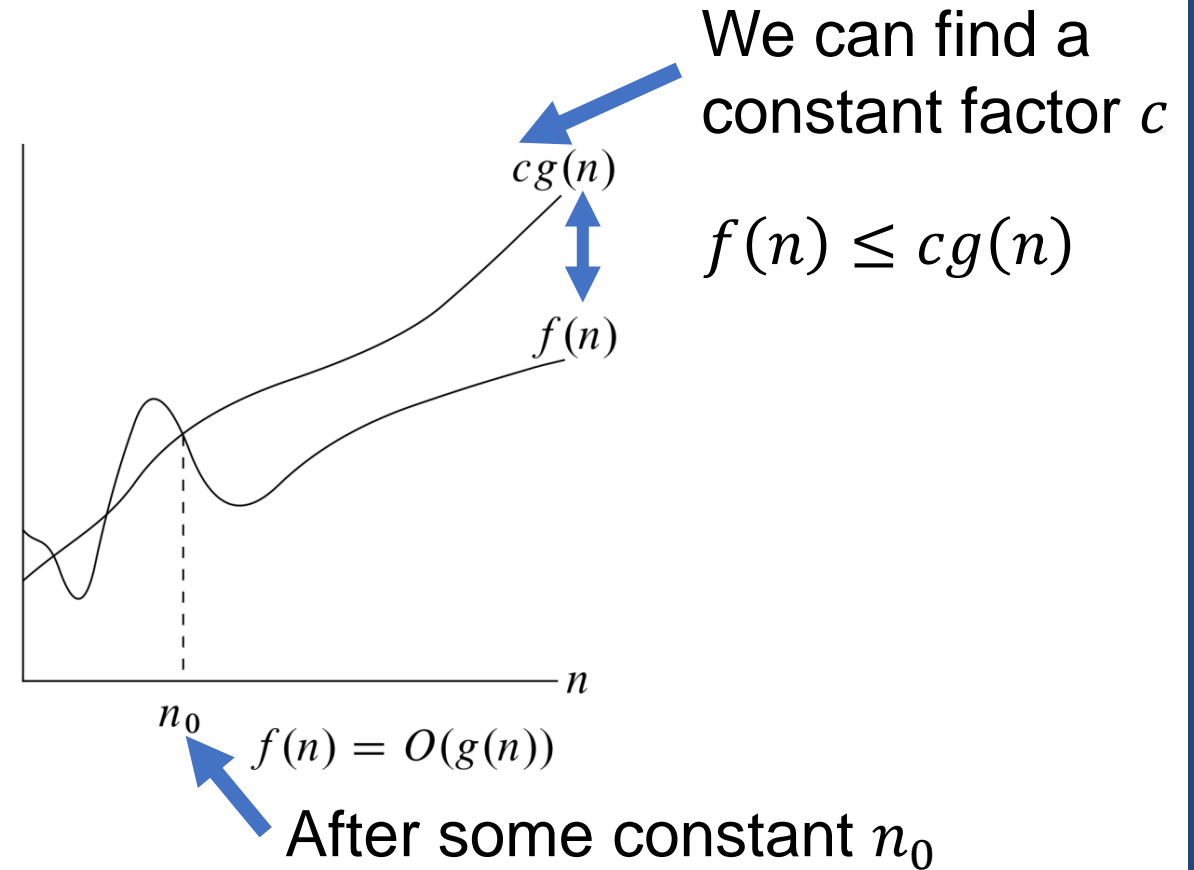


$$n = O(n^2) \quad n^2 = O(n^2), \quad n^3 \neq O(n^2)$$

O (Big-0) – smaller than or equal to

- $f(n) = 5n^2, g(n) = n^2$
- How can we show $f(n) = O(g(n))$?
- Let $n_0 = 10$
- Let $c = 5$
- $f(n) \leq cg(n)$
 - $cg(n) = 5n^2$, which equals to $f(n)$

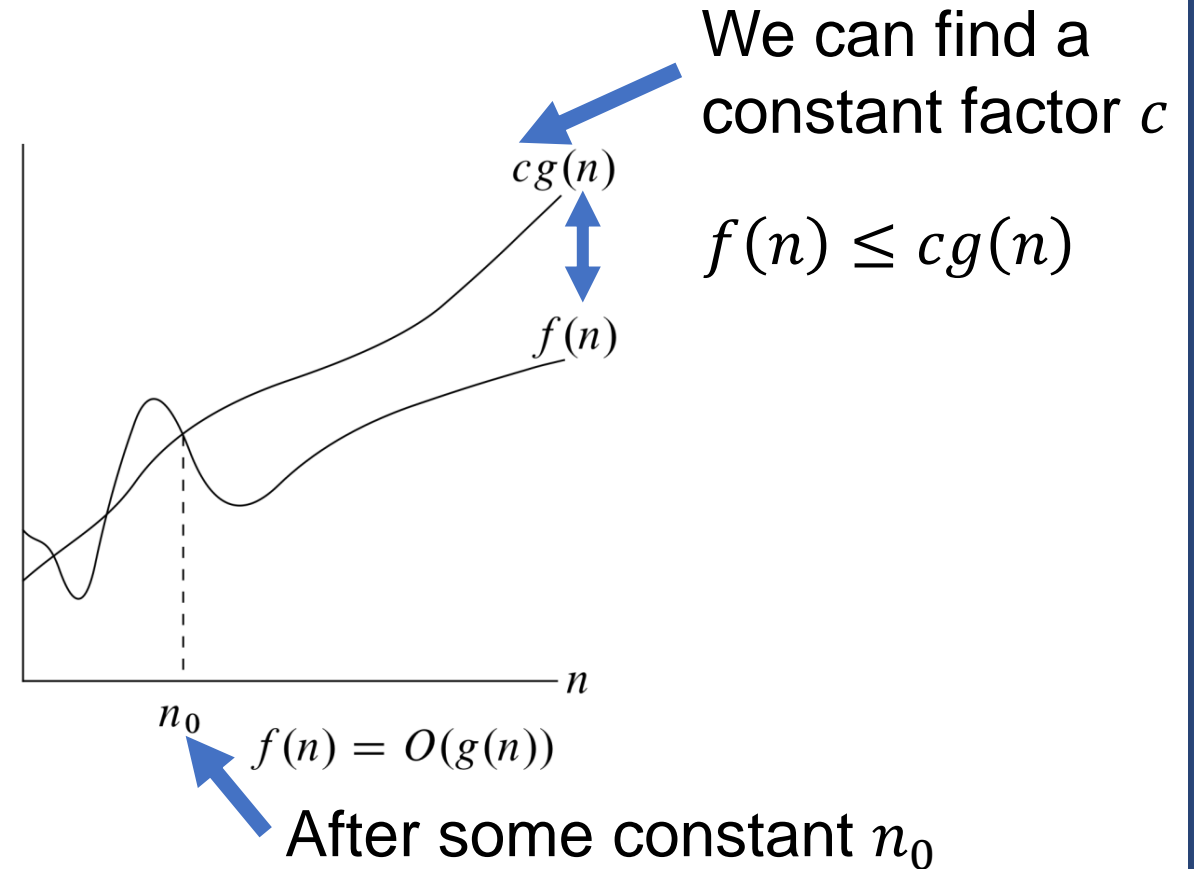
$$\begin{aligned} \exists c > 0, n_0 > 0 \\ 0 \leq f(n) \leq cg(n) \\ \text{for } n \geq n_0 \end{aligned}$$



O (Big-0) – smaller than or equal to

- $f(n) = n^2 + 2n, g(n) = n^3,$
- how can we show $f(n) = O(g(n))$?
- Let $n_0 = 10$
- Let $c = 3$
- $f(n) \leq cg(n)$
 - $cg(n) = 3 \times n^3 = n^3 + 2n^3$, which is always larger than $n^2 + 2n$

$$\begin{aligned} \exists c > 0, n_0 > 0 \\ 0 \leq f(n) \leq cg(n) \\ \text{for } n \geq n_0 \end{aligned}$$

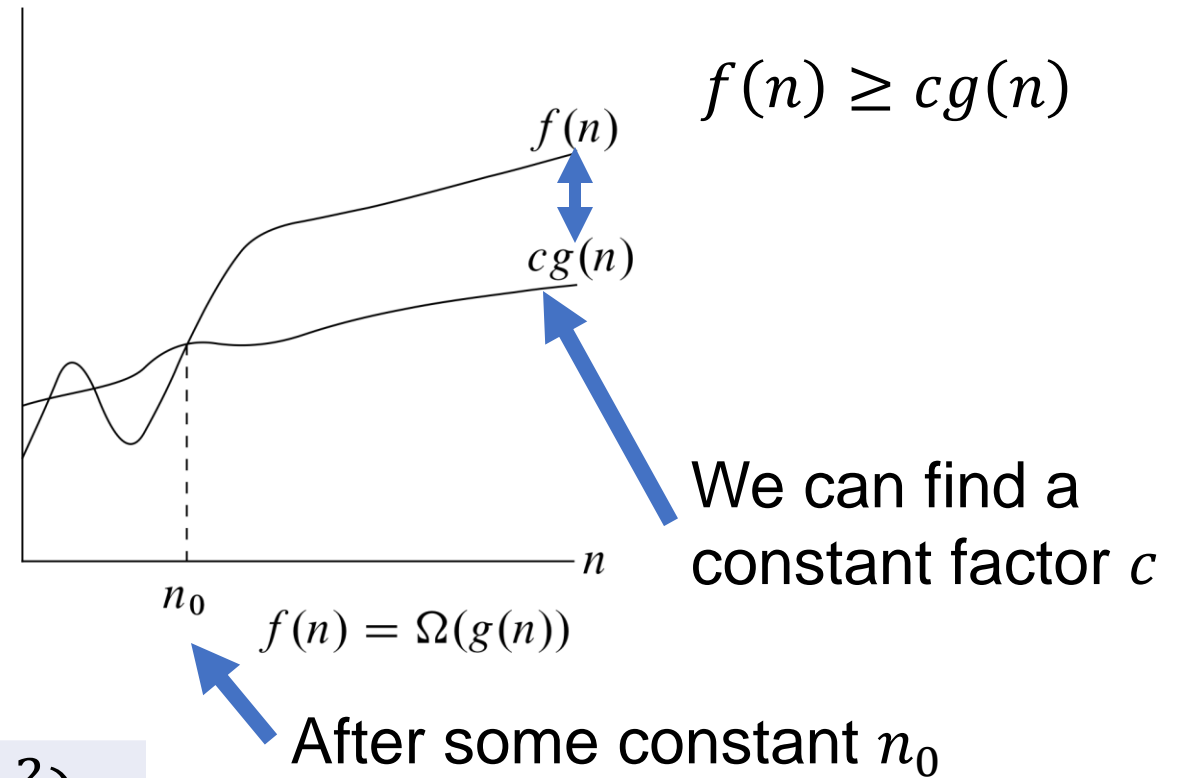


Ω (big-omega)

$$f(n) = \Omega(g(n))$$

- Asymptotic **lower bound**
- “ \geq ”: can be of the **same order**, but can also be “**larger**”

$$\begin{aligned} \exists c > 0, n_0 > 0 \\ 0 \leq cg(n) \leq f(n) \\ \text{for } n \geq n_0 \end{aligned}$$



$$n \neq \Omega(n^2) \quad n^2 = \Omega(n^2), \quad n^3 = \Omega(n^2)$$

Θ (theta) $f(n) = \Theta(g(n))$

- Asymptotic **tight bound**
- “=”: must be of the same order

$$\exists c_1, c_2 > 0, n_0 > 0$$

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$$

$$n \geq n_0$$

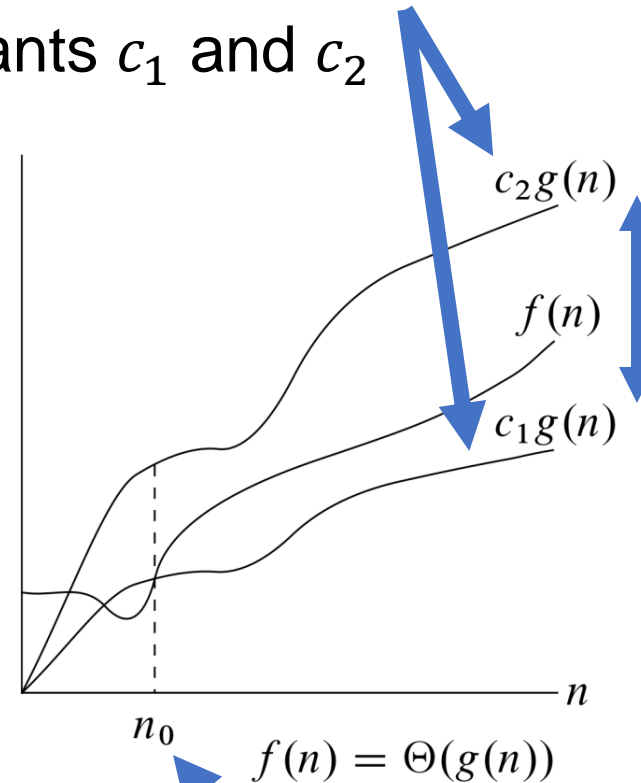
$f(n) = \Theta(g(n))$ means

$$f(n) = O(g(n)) (\leq)$$

$$f(n) = \Omega(g(n)) (\geq)$$

Must be =

We can find a constants c_1 and c_2



$f(n)$ must be within this range

$$n \neq \Theta(n^2) \quad n^2 = \Theta(n^2), \quad n^3 \neq \Theta(n^2)$$

Θ (theta) – equal to

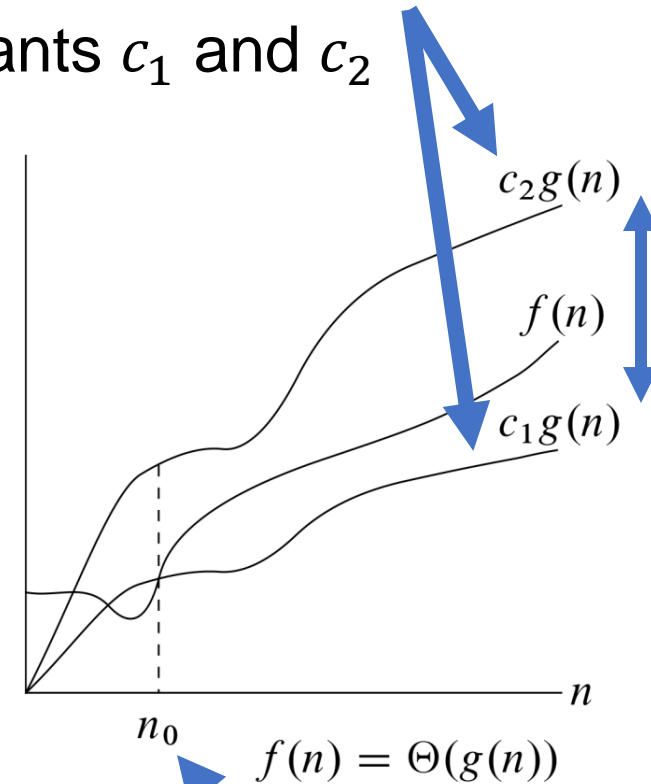
- $f(n) = 2n^2 + 3n, g(n) = n^2$
- How can we show $f(n) = \Theta(g(n))$?
- Let $n_0 = 10$
- Let $c_1 = 1, c_2 = 5$
- $c_1 g(n) \leq f(n) \leq c_2 g(n)$
 - $n^2 \leq 2n^2 + 3n \leq 5n^2$

$$\exists c_1, c_2 > 0, n_0 > 0$$

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$$

$$\text{for } n \geq n_0$$

We can find a constants c_1 and c_2



$f(n)$ must be within this range

After some constant n_0

Analogy to real numbers

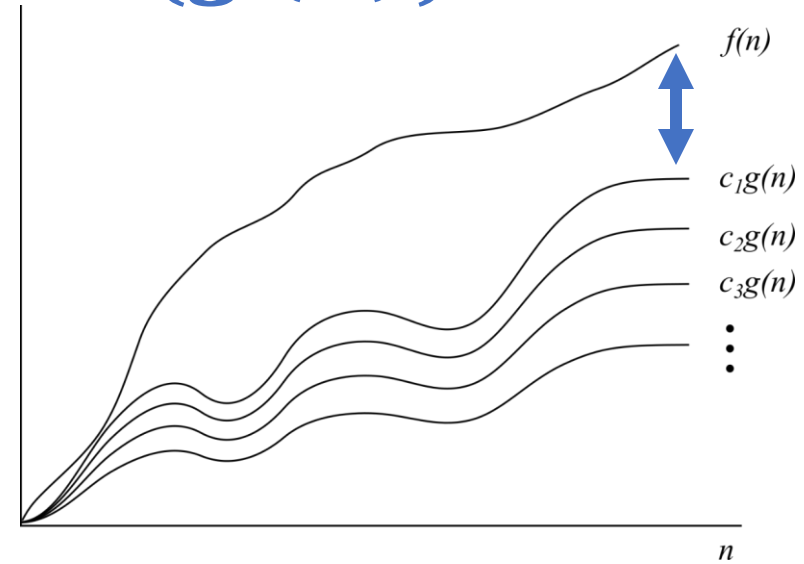
Functions	Real numbers
$f(n) = O(g(n))$	$a \leq b$
$f(n) = \Omega(g(n))$	$a \geq b$
$f(n) = \Theta(g(n))$	$a = b$

ω (small-omega)

- Asymptotic **non-tight lower bound**
- “ $>$ ”: must be “larger”
- You **cannot find a constant c** that makes $cg(n) > f(n)$ for sufficiently large n

$$\begin{aligned} \forall c > 0, \exists n_0 > 0 \\ 0 \leq cg(n) < f(n) \\ n \geq n_0 \end{aligned}$$

$$f(n) = \omega(g(n))$$



Whatever
constant c
you use for
 $g(n)$

We can find
some constant n_0

$f(n) = \omega(g(n))$ means:

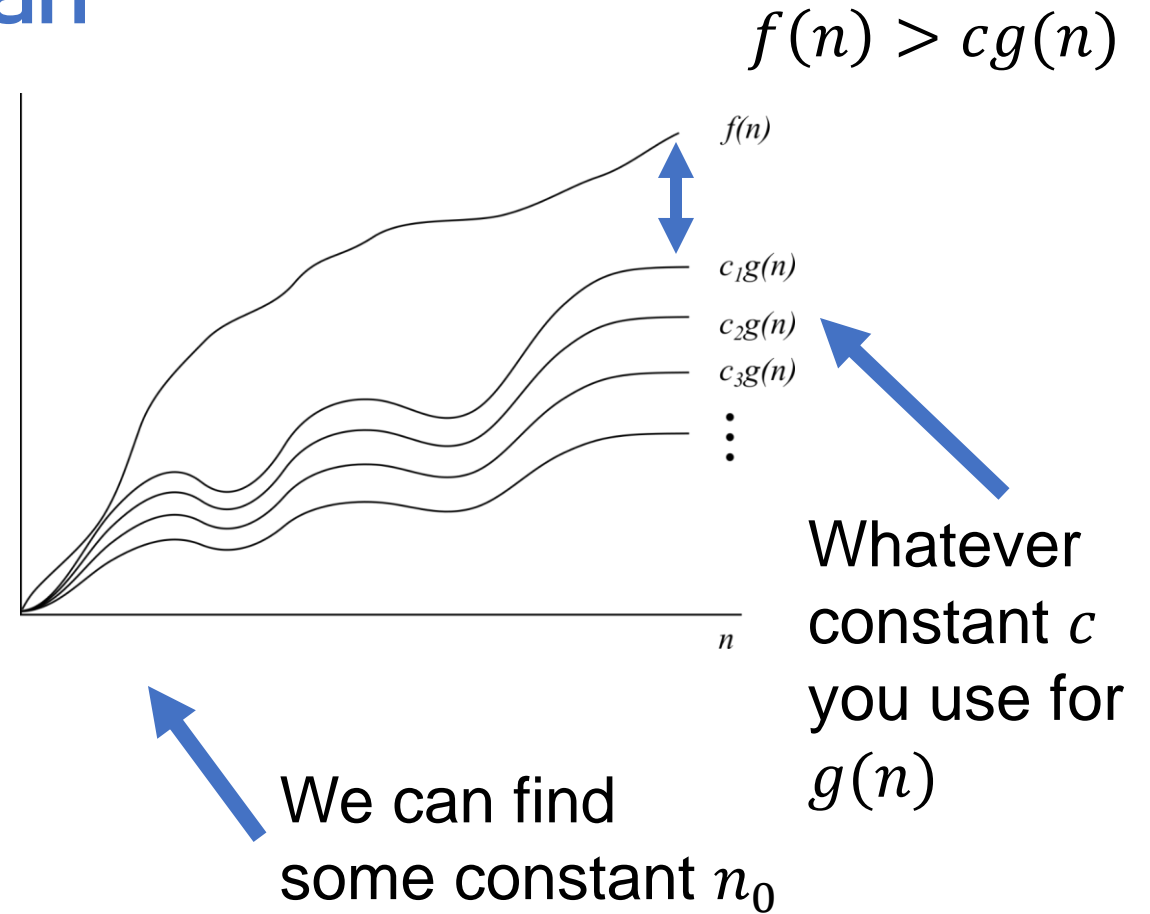
$$f(n) = \Omega(g(n))$$

But $f(n) \neq \Theta(g(n))$

ω (small-omega) – larger than

- $f(n) = n^2, g(n) = n$
- How can we show $f(n) = \omega(g(n))$?
- Whichever c we use, $n^2 > cn$ for all $n > c$
- f is always larger!

$$\begin{aligned} \forall c > 0, \exists n_0 > 0 \\ 0 \leq cg(n) < f(n) \\ n \geq n_0 \end{aligned}$$



o (small- o) $f(n) = o(g(n))$

- Asymptotic **non-tight upper bound**
- “ $<$ ”: must be “smaller”
- You **cannot find a constant c** that makes $cg(n) < f(n)$ for sufficiently large n

$$\forall c > 0, \exists n_0 > 0$$
$$0 \leq f(n) < cg(n)$$

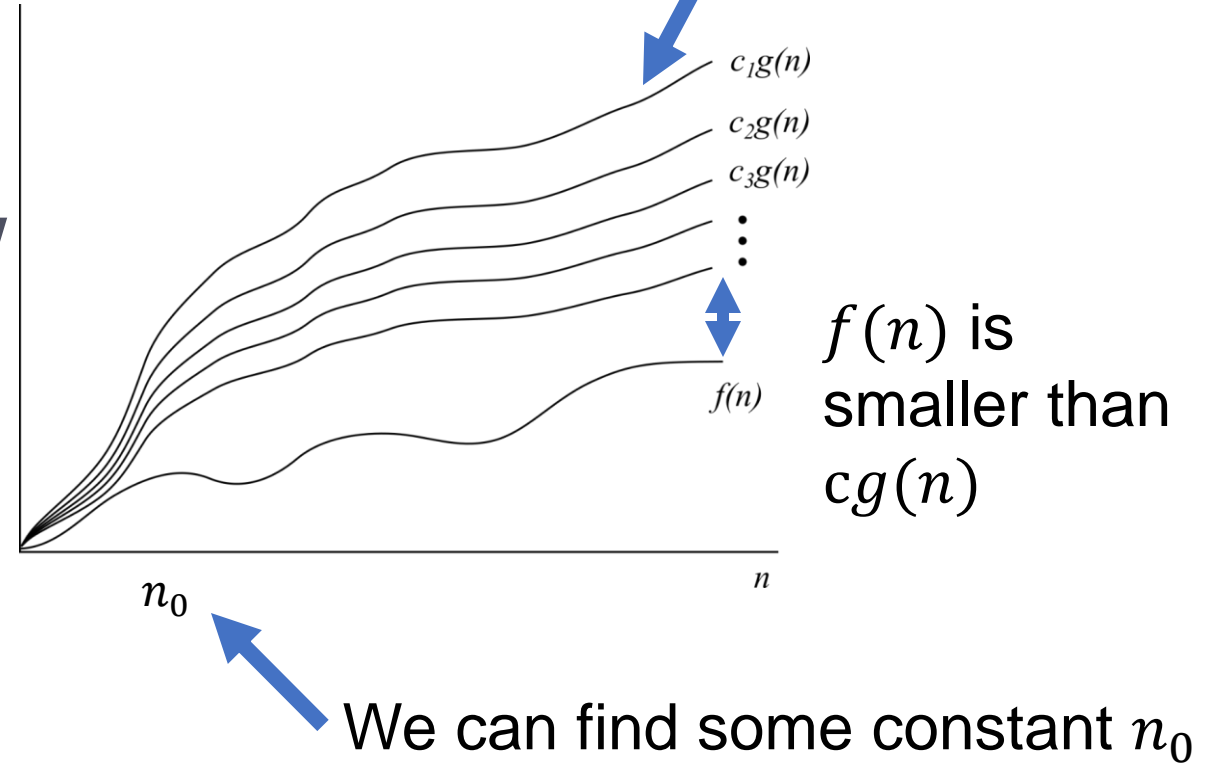
for $n \geq n_0$

$f(n) = o(g(n))$ means:

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

$$\text{But } f(n) \neq \Theta(g(n))$$

Whatever
constant c you
use for $g(n)$



Some notes

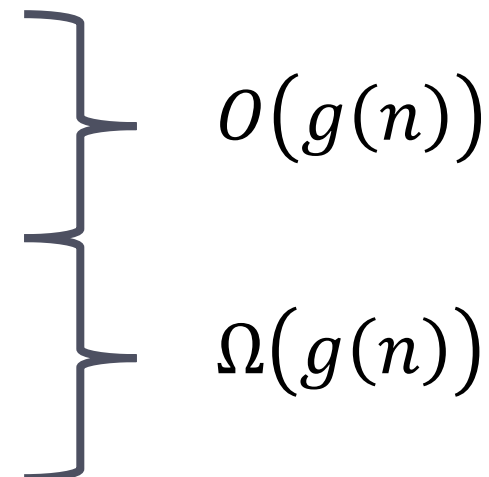
- In an asymptotic bound, we usually **omit constant factors** and **lower-order terms**
 - Eg., we **DONT** use $O(1000)$ (should be $O(1)$), $O(3n^2 + 4n)$ (should be $O(n^2)$), or $O(\log n + \log \log n)$ (should be $O(\log n)$)
- Example:

$n = O(n^2)$	$n^2 = O(n^2),$	$n^3 \neq O(n^2)$
$n \neq \Theta(n^2)$	$n^2 = \Theta(n^2),$	$n^3 \neq \Theta(n^2)$
$n \neq \Omega(n^2)$	$n^2 = \Omega(n^2),$	$n^3 = \Omega(n^2)$
$n \text{ ? } o(n^2)$	$n^2 \text{ ? } o(n^2),$	$n^3 \text{ ? } o(n^2)$
$n \text{ ? } \omega(n^2)$	$n^2 \text{ ? } \omega(n^2)$	$n^3 \text{ ? } \omega(n^2)$

Another easy way to compare to functions

- $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)}$

- 0: $f(n) = o(g(n))$
- **Constant** $c > 0$: $f(n) = \Theta(g(n))$
- ∞ : $f(n) = \omega(g(n))$



$O(g(n))$

$\Omega(g(n))$

Example

- $n = \underline{\quad}(n^2)$

- $\lim_{n \rightarrow \infty} \frac{n}{n^2} = \lim_{n \rightarrow \infty} \frac{1}{n} = 0$

- $n = o(n^2)$

- $3n + \log n = \underline{\quad}(n)$

- $\lim_{n \rightarrow \infty} \frac{3n + \log n}{n} = \lim_{n \rightarrow \infty} 3 + \frac{\log n}{n} = 3$

- $3n + \log n = \Theta(n)$

- $n^2 = \underline{\quad}(n \log n)$

- $\lim_{n \rightarrow \infty} \frac{n^2}{n \log n} = \lim_{n \rightarrow \infty} \frac{n}{\log n} = +\infty$

- $n^2 = \omega(n \log n)$

A. o (small-o)

B. Θ (theta)

C. ω (small-omega)

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

0: $f(n) = o(g(n))$

Constant $c > 0$: $f(n) = \Theta(g(n))$

∞ : $f(n) = \omega(g(n))$

Commonly-used functions

For large enough n , we have (asymptotically):

- $c > 0$
• $c < \log \log n < \log n < \log^{c_1} n < n^{c_2} < n < n \log n < n^{c_3} < c_4^n < n^n$
 $c_1 > 1$ $0 < c_2 < 1$ $c_3 > 1$ $c_4 > 1$
- $f(n) = n \log n$, $g(n) = n^{1.1}$, which one is asymptotically larger? [[demo](#)]
 - A. $f(n)$
 - B. $g(n)$

Solution: $n^{1.1}$ is larger. We have $n \log n < n^{c_3}$ for any constant $c_3 > 1$!

Commonly-used functions

For large enough n , we have (asymptotically):

- $c > 0$
• $c < \log \log n < \log n < \log^{c_1} n < n^{c_2} < n < n \log n < n^{c_3} < c_4^n < n^n$
 $c_1 > 1$ $0 < c_2 < 1$ $c_3 > 1$ $c_4 > 1$
- $f(n) = n^{10}$, $g(n) = 2^n$, which one is asymptotically larger? [[demo](#)]
 - A. $f(n)$
 - B. $g(n)$

Solution: 2^n is larger. We have $n^{c_3} < c_4^n$ for any constant c_3 and $c_4 > 1$!

Commonly-used functions

For large enough n , we have (asymptotically):

- $c > 0$
• $c < \log \log n < \log n < \log^{c_1} n < n^{c_2} < n < n \log n < n^{c_3} < c_4^n < n^n$
 $c_1 > 1$ $0 < c_2 < 1$ $c_3 > 1$ $c_4 > 1$
- $f(n) = \log^2 n$, $g(n) = \sqrt{n}$, which one is asymptotically larger? [[demo](#)]
 - A. $f(n)$
 - B. $g(n)$

Solution: $\sqrt{n} = n^{0.5}$ is larger. We have $\log^{c_1} n < n^{c_3}$ for any constant c_1 and $c_3 > 1$!

(Let's use $\log n$ as $\log_2 n$, and c is a constant)

Simple Rules

- We can **omit constants**
- We can **omit lower order terms**
- $\Theta(an^2 + bn + c)$ **becomes** $\Theta(n^2)$
- $\Theta(c)$ **becomes** $\Theta(1)$
- $\Theta(\log_c n)$ **becomes** $\Theta(\log n)$ (assuming $c > 1$)
 - $\log_c n = \log n / \log c$
- $\Theta(\log(n^c)) = \Theta(c \log(n))$ **becomes** $\Theta(\log n)$
 - But $\Theta(\log^2 n) \neq \Theta(\log n)$

Simple Rules

- $f(n) = \Theta(f(n))$
- $\Theta(f(n))\Theta(g(n)) = \Theta(f(n)g(n))$
- $\Theta(f(n)) + \Theta(g(n)) = \Theta(f(n) + g(n))$
- $O(f(n)) - O(g(n)) \neq O(f(n) - g(n))$

$f(n) = \Theta(n^2), g(n) = \Theta(n^2) \Rightarrow f(n) - g(n) = \Theta(n^2 - n^2) = 0?$
(No. Consider $f(n) = n^2 + n, g(n) = n^2, f_1(n) - f_2(n) = \Theta(n)$)

Analogy to real numbers

Functions	Real numbers
$f(n) = O(g(n))$	$a \leq b$
$f(n) = \Omega(g(n))$	$a \geq b$
$f(n) = \Theta(g(n))$	$a = b$
$f(n) = o(g(n))$	$a < b$
$f(n) = \omega(g(n))$	$a > b$

Popular Classes of Functions

		Example
• Constant:	$f(n) = \Theta(1)$	1, 10, 100000000
• Logarithmic:	$f(n) = \Theta(\log(n))$	$\log n$, $\log n + \log \log n$
• Poly-logarithmic:	$f(n) = O(\log^k n)$	$\log^2 n$, $\log^9 n + 8$
• Sublinear:	$f(n) = o(n)$	$\log n$, \sqrt{n} , $n^{1/5}$
• Linear:	$f(n) = \Theta(n)$	n , $5n + \log n$
• Super-linear:	$f(n) = \omega(n)$	n^3 , $n \log n$
• Quadratic:	$f(n) = \Theta(n^2)$	n^2 , $3n^2 + n$
• Polynomial:	$f(n) = O(n^k)$	$n^3 + 2n^2 + 4$, $4n^5$
• Exponential:	$f(n) = \Theta(k^n)$	2^n

($k \geq 0$ is a constant)

Case study: selection sort

- Find the smallest element in the array
- Move it to the front
- For the rest of elements, find the smallest element and repeat

```
for (int i = 0; i < n; i++) {  
    for (int j = i+1; j < n; j++) {  
        if (a[j] < a[i]) swap(a[i], a[j]);  
    }  
}
```

$$(n-1) + (n-2) + \dots + 1 = n \times \frac{n-1}{2} = \Theta(n^2)$$

Why we don't care about constants & lower-order terms?

- Why we only care about the order (main term)?

Quicksort: $O(n \log n)$ time in expectation

```
void qsort(data_type* A, int l, int r)
{
    int i = l, j = r-1;
    data_type x = A[random_number(l, r)];
    do {
        while (A[i] < x) i++;
        while (A[j] > x) j--;
        if (i <= j) {
            swap(A[i], A[j]);
            i++;
            j--;
        }
    } while (i <= j);

    if (i < r - 1) qsort(A, i, r);
    if (l < j) qsort(A, l, j+1);
}
```

Running time in ms:

	Quicksort	Selection sort
10	0.0098	0.0067
100	0.011	0.027
1,000	0.14	1.683
10,000	1.695	286.425
100,000	19.46	13799.5
1,000,000	226.443	1291300
10,000,000	1706.54	

Within 1s

About 20 min

Selection sort: $O(n^2)$ time:

```
for (int i = 0; i < n; i++)
    for (int j = i+1; j < n; j++)
        if (a[i]>a[j]) { swap(A[i], A[j]); }
```