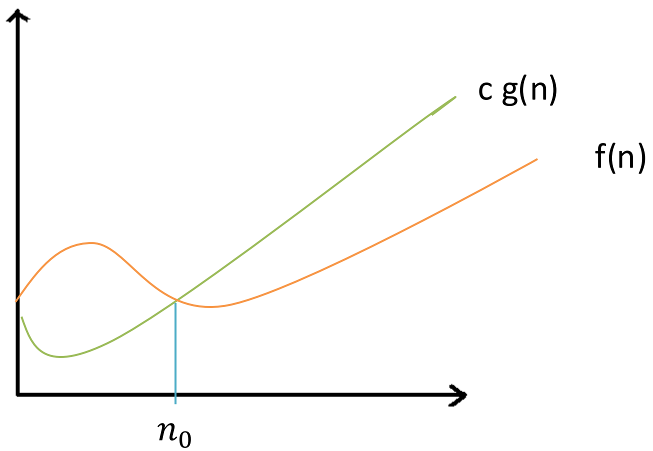
ANALYSIS OF ALGORITHMS

CMSC 142, LEC1

Outline of Today's Lecture

- O notation
- Ω notation
- Θ notation

O notation



$f(n) \leq cg(n)$

for $n \geq n_0$

$c > 0$, $n_0 \geq 1$

$f(n) \in O(g(n))$

Example

Given: f(n) = 3n+2 and g(n) = n

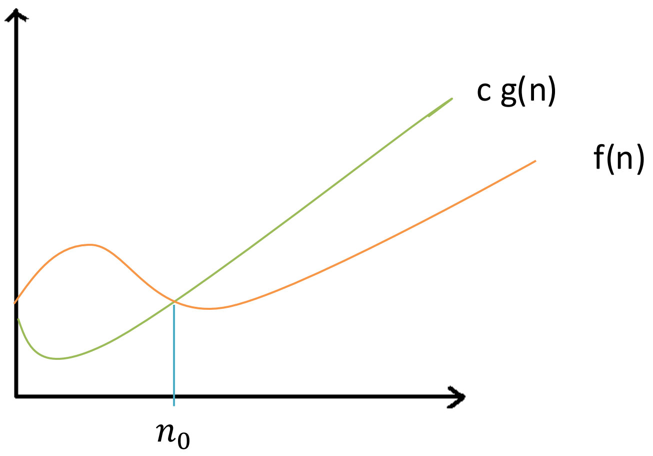
Prove that f(n) ∈ O(g(n))

$$f(n) \leq cg(n)$$

3n + 2 ≤ cg(n),   c = 5

3n+2 ≤ 5n

$n_0 \geq 1$



O notation

- f, g : nonnegative functions of nonnegative arguments

O(g) : $\left\{ f \mid f \text{ is a non negative function s.t. } \exists \text{ constants } c, n_o > 0 \text{ s.t. } f(n) \leq cg(n) \text{ for } n \geq n_0 \right\}$

O notation

- g(n) is an asymptotic upper bound for f(n)
- Using O-notation, we can often describe the RT of algorithm merely by inspecting the algorithm's ?

- Example: double for-loop: n^2

O notation

- g(n) is an asymptotic upper bound for f(n)
- Using O-notation, we can often describe the RT of algorithm merely by inspecting the algorithm's overall structure

- Example: double for-loop: n^2

O notation

- Since O-notation describes upper-bound, when we use it as bound of the __________ RT of an algorithm, we have a bound on the RT of the algorithm on every input

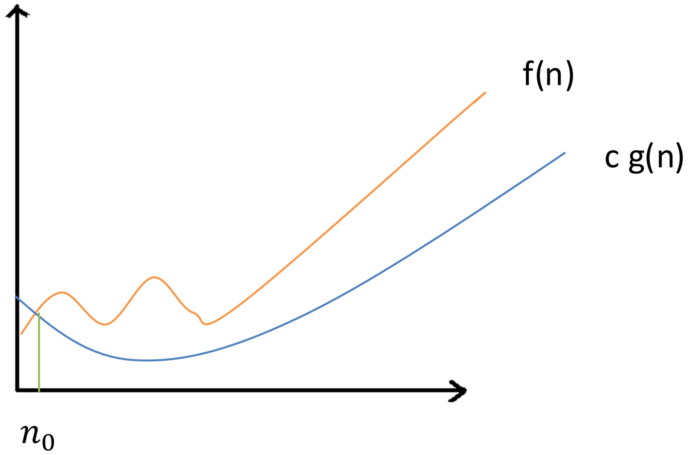
O notation

- Since O-notation describes upper-bound, when we use it as bound of the worst case RT of an algorithm, we have a bound on the RT of the algorithm on every input

Outline of Today's Lecture

- O notation
- Ω notation
- Θ notation

Ω notation



$f(n) \geq cg(n)$

for $n \geq n_0$

c >0, $n_0 \geq 1$

Example 1

Given: f(n) = 3n+2 and g(n) = n

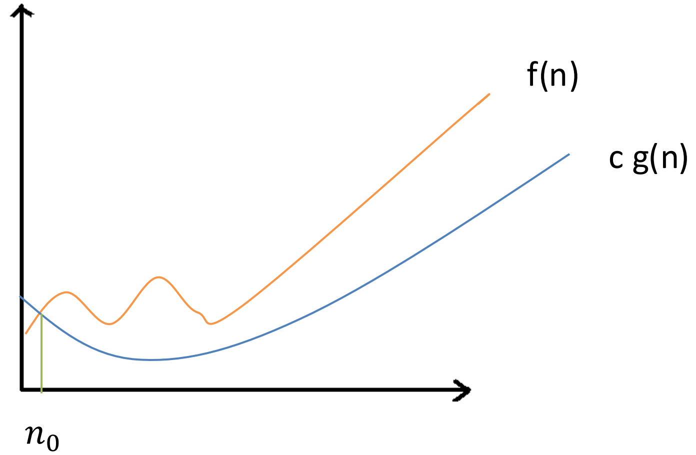
Prove that f(n) ∈ Ω (g(n))

$$f(n) \geq cg(n)$$

3n + 2 ≥ cg(n), c = 1

3n+2 ≥ n

$n_0 \geq 1$



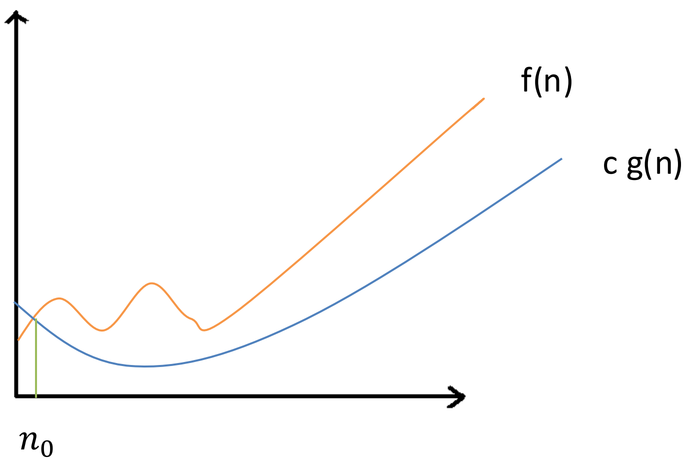
Example 2

Given: f(n) = 3n+2 and g(n) = n^2

Is f(n) ∈ Ω (g(n))?

$f(n) \geq cg(n)$

3n+2 ≥ cn^2 , n_0



It's not possible! f(n) ∈ Ω (n^2)

O(log n)

Ω notation

- f, g : nonnegative functions of nonnegative arguments

Ω(g) : $\left\{ f \mid f \text{ is a non negative function s.t. } \exists \text{ constants } c, n_o > 0 \text{ s.t. } cg(n) \le f(n) \text{ for } n \ge n_0 \right\}$

Ω notation

- If O-notation provides asymptotic upper bound, Ω-notation (Big Omega) provides asymptotic ___________

Ω notation

- If O-notation provides asymptotic upper bound, Ω-notation (Big Omega) provides asymptotic lower bound

Ω notation

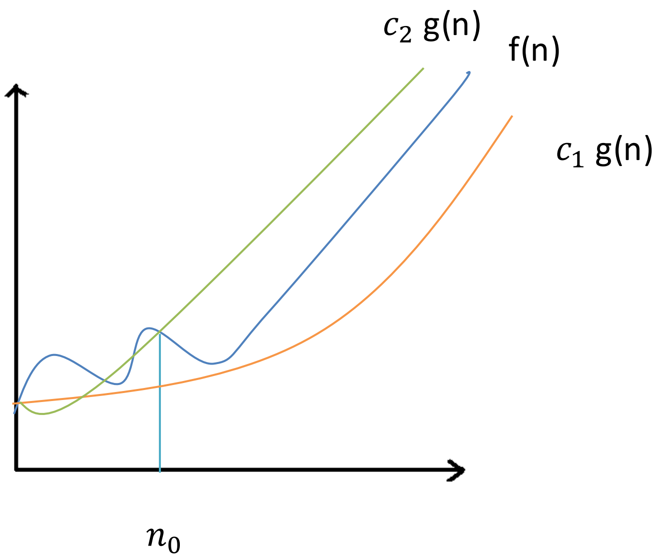
- We will not deal with Ω-notation as often as O-notation. [Why?]

- Ω-notation describes best case, but we want to always consider the worst-case

Outline of Today's Lecture

- O notation
- Ω notation
- Θ notation

Θ notation



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

$c_1, c_2 > 0$ for n≥ n_0

Example 1

Given: f(n) = 3n+2 and g(n) = n

Prove that f(n) ∈ O(g(n))

$f(n) \le cg(n)$

3n + 2 $\le$ cg(n) , c = 5

3n+2 $\le$ 5n , $n_0 \geq 1$

$f(n) \geq cg(n)$

3n + 2 $\geq$ cg(n) , c = 1

3n+2 $\geq$ n , $n_0 \geq 1$

n $\le$ 3n+2 $\le$ 5n

Example 2

- $f_1(n) = 10n^3 + 5n^2 + 17 \quad \in \quad \Theta(n^3)$
- $f_2(n) = 2n^3 + 3n + 79 \quad \in \quad \Theta(n^3)$

Example 2

$f_1(n) = 10n^3 + 5n^2 + 17 \quad \in \quad \Theta(n^3)$

Proof:

$10n^3 \leq f_1 n \leq (10 + 5 + 17)n^3 = 32n^3$

$c_1 = 10, \quad c_2 = 32$

$c_1 n^3 \leq f_1 n \leq c_2 n^3$

For all n ≥ 1

Θ notation

- f, g : nonnegative functions of nonnegative arguments

Θ(g) : $\left\{ \begin{array}{l} f \mid f \text{ is a non negative function s.t. } \exists \text{ constants } c_1, c_2, n_o > 0 \\ \text{s.t. } c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for } n \geq n_0 \end{array} \right\}$

Θ notation

- Theta Notation = f(n) can be sandwiched between c_1g(n) and c_2g(n), for sufficiently large n

- Θ-notation is usually used to describe average cases.

Note!

Suppose $f \in \Theta(g)$

More common style: $f = \Theta(g)$

Outline of Today's Lecture

- Θ notation
- O notation
- Ω notation

$O(g)$ vs $\Omega(g)$ vs $\Theta(g)$

O(g) vs Ω(g) vs Θ(g)

- f, g : nonnegative functions of nonnegative arguments

O(g) : $\left\{ f \mid f \right.$ is a non negative function s.t. ∃ constants c_2, n_o>0
s.t. $f(n) \le c_2 g(n)$ for n≥ $\left. n_0 \right\}$

Ω(g) : $\left\{ f \mid f \right.$ is a non negative function s.t. ∃ constants c_1, n_o>0
s.t. $c_1 g(n) \le f(n)$ for n≥ $\left. n_0 \right\}$

Θ(g) : $\left\{ f \mid f \right.$ is a non negative function s.t. ∃ constants $c_1, c_{2,} n_o$>0
s.t. $\left. c_1 g(n) \le f(n) \le c_2 g(n) \right\}$

O(g) vs Ω(g) vs Θ(g)

- O(g) : __________ bound
- Ω(g) : __________ bound
- Θ(g) : __________ bound

O(g) vs Ω(g) vs Θ(g)

- O(g) : upper bound, worst case
- Ω(g) : lower bound, best case
- Θ(g) : average average case

$O(g)$ vs $\Omega(g)$ vs $\Theta(g)$

- $f \in O(g)$: f no larger than g
- $f \in \Omega(g)$: f is greater than or equal to g, ignoring constants
- $f \in \Theta(g)$: f nearly similar to g

$O(g)$ vs $\Omega(g)$ vs $\Theta(g)$

- $f \in O(g)$: $\leq$
- $f \in \Omega(g)$: $\geq$
- $f \in \Theta(g)$: $=$

What notation should be used

- **$f \in O(g)$, in practice**
- $f \in \Omega(g)$
- $f \in \Theta(g)$

What notation should be used

- $f \in O(g)$, in practice
- $f \in \Omega(g)$
- **$f \in \Theta(g)$, in general cases**

Comparing growth rates

Algorithm A

Running Time:

f(n) = $3n^2 - n + 10$

If n = 5

A = 80

B = 38

Algorithm B

Running Time:

g(n) = $2^n - 50n + 256$

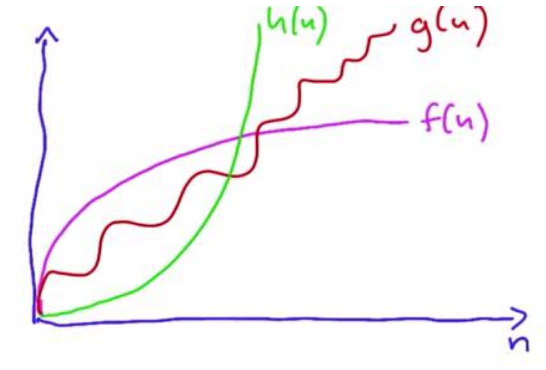
If n = 100

A = 29910

B = 1267650600228229401496703200632

Introduction to Theoretical Computer Science, Udacity

Comparing growth rates



- Which of the following is true?

___ $f(n) \in O(g(n))$

___ $f(n) \in O(h(n))$

____ $g(n) \in O(f(n))$

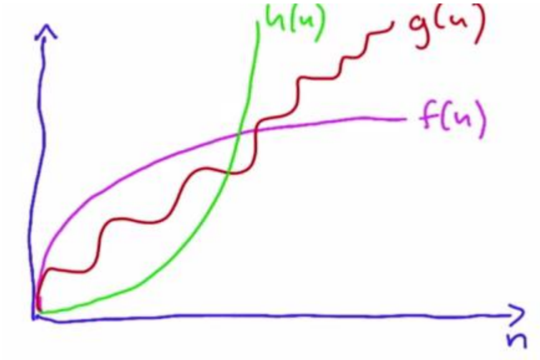
___ $g(n) \in O(h(n))$

____ $h(n) \in O(f(n))$

____ $h(n) \in O(g(n))$

Introduction to Theoretical Computer Science, Udacity

Comparing growth rates



- Which of the following is true?

✓ $f(n) \in O(g(n))$

✓ $f(n) \in O(h(n))$

_____ $g(n) \in O(f(n))$

✓ $g(n) \in O(h(n))$

_____ $h(n) \in O(f(n))$

_____ $h(n) \in O(g(n))$

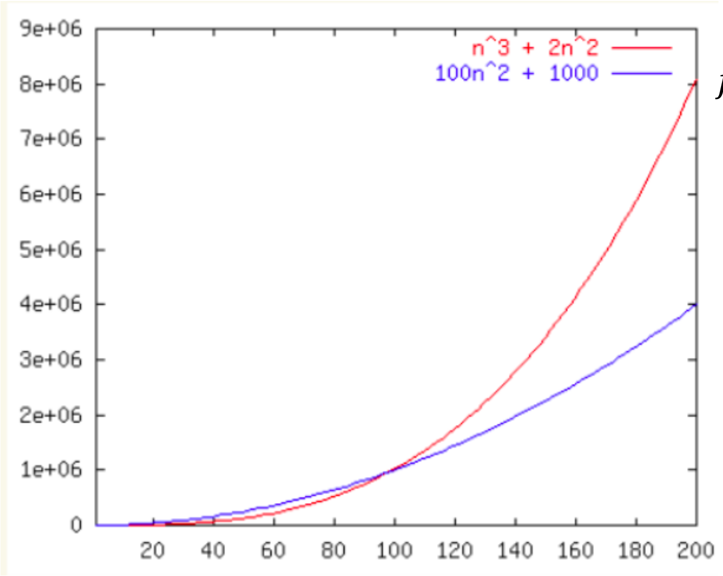
Introduction to Theoretical Computer Science, Udacity

Which one is faster?

Is $f(n) \in O(g(n))$?

$f(n) = n^3 + 2n^2$

$g(n) = 100n^2 + 1000$



$f(n)$

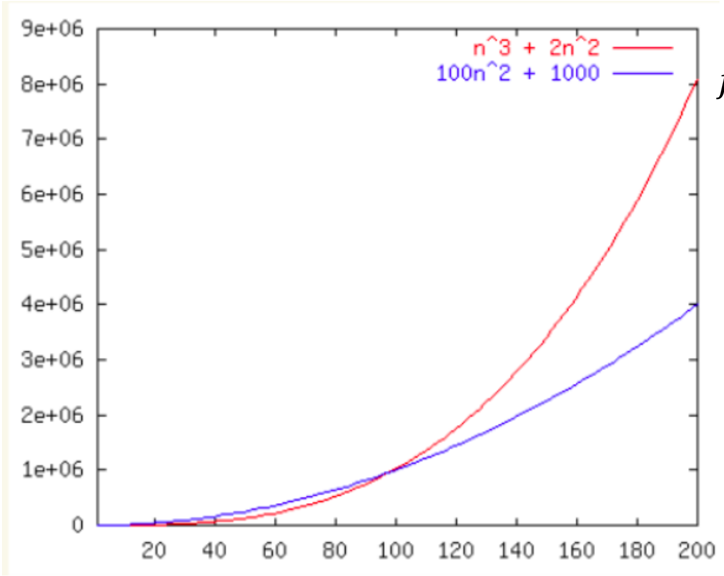
$g(n)$

Which one is faster?

Is $f(n) \in O(g(n))$? **NO!**

$f(n) = n^3 + 2n^2$

$g(n) = 100n^2 + 1000$



$f(n)$

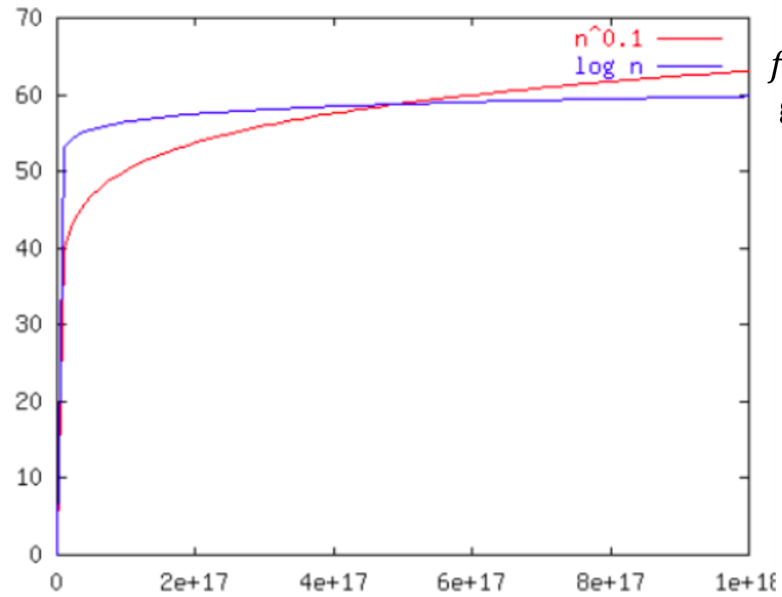
$g(n)$

Which one is faster?

Is $f(n) \in O(g(n))$?

$f(n) = n^{0.1}$

$g(n) = logn$



$f(n)$

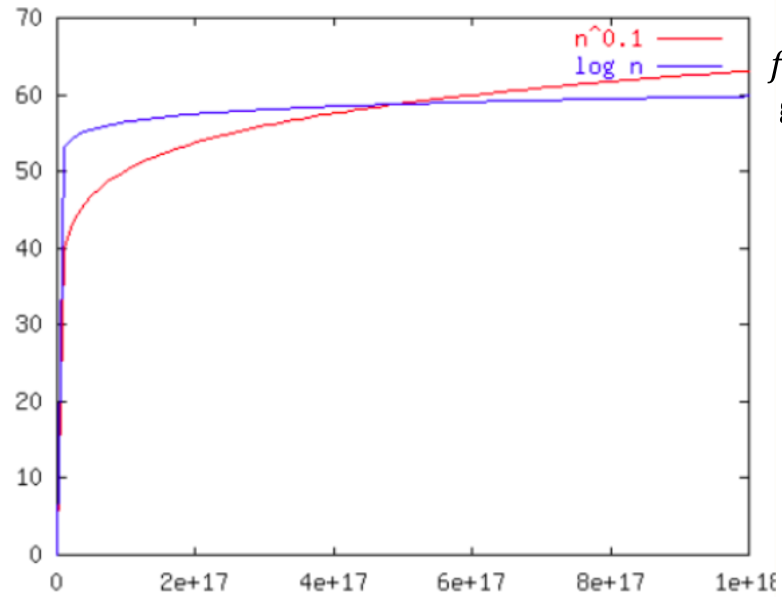
$g(n)$

Which one is faster?

Is $f(n) \in O(g(n))$? **NO!**

$f(n) = n^{0.1}$

$\mathrm{g}(n) = logn$



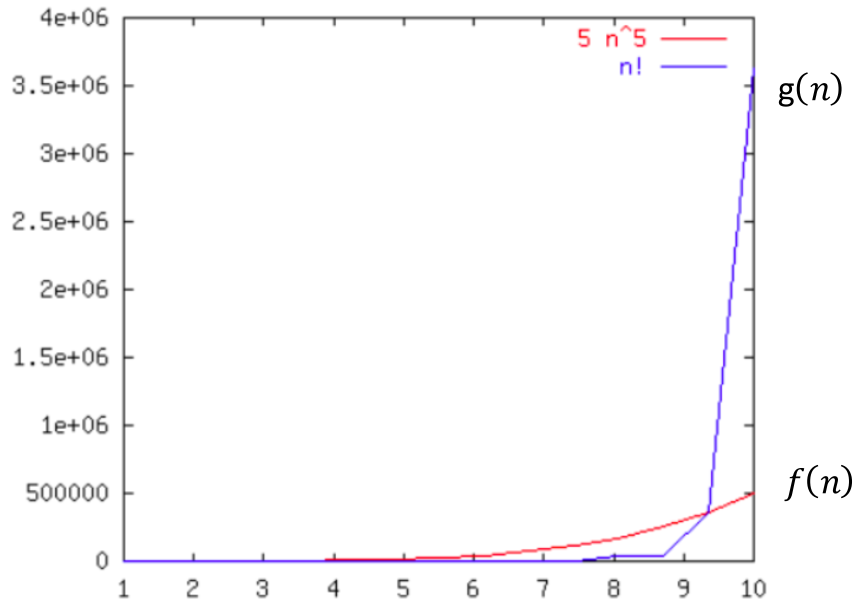
$f(n)$
$\mathrm{g}(n)$

Which one is faster?

Is $f(n) \in O(g(n))$?

$f(n) = 5n^5$

$g(n) = n!$

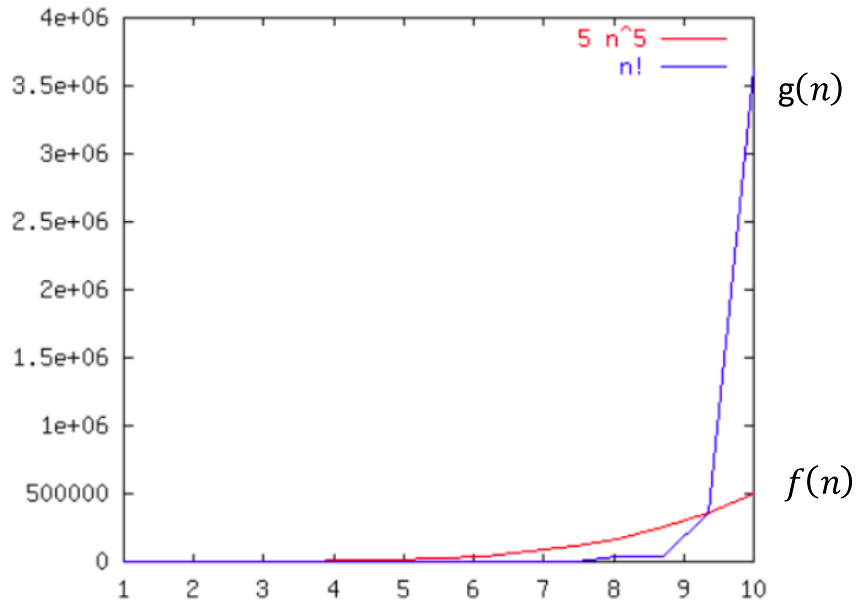


Which one is faster?

Is $f(n) \in O(g(n))$? **YES!**

$f(n) = 5n^5$

$g(n) = n!$



Comparing growth rates

- $3n + 1 \quad \in \quad O(n)$
- $18n^2 - 50 \quad \in \quad O(n^2)$
- $2^n + 30n^6 + 123 \quad \in \quad O(2^n)$

- $3n + 1 \quad \in \quad O(n^2)$, correct but not tightly bound
- $18n^2 - 50 \quad \in \quad O(n^3)$, correct but not tightly bound

Introduction to Theoretical Computer Science, Udacity

Question

			Correct?	Tightly bound?
$4n^2 - 300n + 12$	$\in$	$O(n^2)$	____	____
$4n^2 - 30n + 12$	$\in$	$O(n^3)$	____	____
$3^n + 5n^2 - 3n$	$\in$	$O(n^2)$	____	____
$3^n + 5n^2 - 3n$	$\in$	$O(3^n)$	____	____
$3^n + 5n^2 - 3n$	$\in$	$O(4^n)$	____	____
$50 \cdot 2^n \cdot n^2 + 5n - \log(n)$	$\in$	$O(2^n)$	____	____

Question

			Correct?	Tightly bound?
$4n^2 - 300n + 12$	$\in$	$O(n^2)$	✓	✓
$4n^2 - 30n + 12$	$\in$	$O(n^3)$	✓	___
$3^n + 5n^2 - 3n$	$\in$	$O(n^2)$	___	___
$3^n + 5n^2 - 3n$	$\in$	$O(3^n)$	✓	✓
$3^n + 5n^2 - 3n$	$\in$	$O(4^n)$	✓	___
$50 \cdot 2^n \cdot n^2 + 5n - \log(n)$	$\in$	$O(2^n)$	___	___

Expression	Dominant term(s)	$O(\ldots)$
$5 + 0.001n^3 + 0.025n$		
$500n + 100n^{1.5} + 50n \log_{10} n$		
$0.3n + 5n^{1.5} + 2.5 \cdot n^{1.75}$		
$n^2 \log_2 n + n(\log_2 n)^2$		
$n \log_3 n + n \log_2 n$		
$3 \log_8 n + \log_2 \log_2 \log_2 n$		
$100n + 0.01n^2$		
$0.01n + 100n^2$		
$2n + n^{0.5} + 0.5n^{1.25}$		
$0.01n \log_2 n + n(\log_2 n)^2$		
$100n \log_3 n + n^3 + 100n$		
$0.003 \log_4 n + \log_2 \log_2 n$		

Expression	Dominant term(s)	$O(\ldots)$
$5 + 0.001n^3 + 0.025n$	$0.001n^3$	$O(n^3)$
$500n + 100n^{1.5} + 50n\log_{10} n$	$100n^{1.5}$	$O(n^{1.5})$
$0.3n + 5n^{1.5} + 2.5 \cdot n^{1.75}$	$2.5n^{1.75}$	$O(n^{1.75})$
$n^2\log_2 n + n(\log_2 n)^2$	$n^2\log_2 n$	$O(n^2\log n)$
$n\log_3 n + n\log_2 n$	$n\log_3 n,\ n\log_2 n$	$O(n\log n)$
$3\log_8 n + \log_2\log_2\log_2 n$	$3\log_8 n$	$O(\log n)$
$100n + 0.01n^2$	$0.01n^2$	$O(n^2)$
$0.01n + 100n^2$	$100n^2$	$O(n^2)$
$2n + n^{0.5} + 0.5n^{1.25}$	$0.5n^{1.25}$	$O(n^{1.25})$
$0.01n\log_2 n + n(\log_2 n)^2$	$n(\log_2 n)^2$	$O(n(\log n)^2)$
$100n\log_3 n + n^3 + 100n$	n^3	$O(n^3)$
$0.003\log_4 n + \log_2\log_2 n$	$0.003\log_4 n$	$O(\log n)$

	Cost
count = count + 1;	c1
sum = sum + count;	c2

- Each operation in an algorithm (or a program) has a cost.

    ➔ Each operation takes a certain of time.

`count = count + 1;` ➔ take a certain amount of time, but it is constant

A sequence of operations:

```
count = count + 1;
sum = sum + count;
```

Cost: c_1

Cost: c_2

➔ Total Cost = $c_1 + c_2$

Example: Simple If-Statement

	Cost	**Times**
`if (n < 0)`	c1	1
`absval = -n`	c2	1
`else`		
`absval = n;`	c3	1

Total Cost <= c1 + max(c2,c3)

Example: Simple Loop

Code	Cost	Times
`i = 1;`	c1	1
`sum = 0;`	c2	1
`while (i <= n) {`	c3	n+1
`    i = i + 1;`	c4	n
`    sum = sum + i;`	c5	n
`}`		

Total Cost = c1 + c2 + (n+1)*c3 + n*c4 + n*c5

= (c3+c4+c5)*n + (c1+c2+c3)

= a*n + b

➔ So, the growth-rate function for this algorithm is **O(n)**

Example: Nested Loop

```
                                        Cost      Times
 i=1;                                    c1        1
 sum = 0;                                c2        1
 while (i <= n) {                        c3        n+1
      j=1;                               c4        n
      while (j <= n) {                   c5        n*(n+1)
          sum = sum + i;                 c6        n*n
          j = j + 1;                     c7        n*n
    }
    i = i +1;                            c8        n
 }
```

T(n) = c1 + c2 + (n+1)*c3 + n*c4 + n*(n+1)*c5+n*n*c6+n*n*c7+n*c8

= (c5+c6+c7)*n^2 + (c3+c4+c5+c8)*n + (c1+c2+c3)

= a*n^2 + b*n + c

➔ So, the growth-rate function for this algorithm is **O(n^2)**

```
i = 0;
while (i<N) {
  X=X+Y;                    // O(1)
  result = mystery(X);      // O(N), just an example...
  i++;                      // O(1)
}
```

The body of the while loop: O(N)

Loop is executed: N times

$N \times O(N) = O(N^2)$

```
if (i<j)
 for ( i=0; i<N; i++ )
      X = X+i;
else
 X=0;
```

$\left.\begin{array}{l}\texttt{if (i<j)}\\ \texttt{for (i=0; i<N; i++)}\\ \texttt{X = X+i;}\end{array}\right\}$ O(N)

$\left.\begin{array}{l}\texttt{else}\\ \texttt{X=0;}\end{array}\right\}$ O(1)

```
Max ( O(N), O(1) ) = O (N)
```

More Examples

```
for (i = 10; i < n + 5; i += 2)
      op();
```

$$O(n) = (n + 5 - 10)/2$$

```
for (i = 1; i < n; i *= 2)
      op();
```

```
for (i = n; i > 1; i /= 2)
      op();
```

O(log n)

```
for (i = 1; i < n; i *= 2)
      op();
```

O(log n)

	2	2*2		2*2..*2
i_0	i_1	i_2		i_k
2^0	2^1	2^2	...	2^k

Loop stops when

$$2^k \geq n$$

$$\log_2 2^k \geq \log_2 n$$

$$k \leq \log_2 n$$

$$O(logn)$$

```
for (i = n; i > 1; i /= 2)
      op();
```

O(log n)

$$\begin{array}{ccccc} i_0 & i_1 & i_2 & & i_k \\ n * \frac{1}{2^0} & n * \frac{1}{2^1} & n * \frac{1}{2^2} & \cdots & n * \frac{1}{2^k} \end{array}$$

$$n * \frac{1}{2^k} \leq 1$$

$$n \leq 2^k$$

$$\log_2 n \leq \log_2 2^k$$

$$\log_2 n \leq k$$

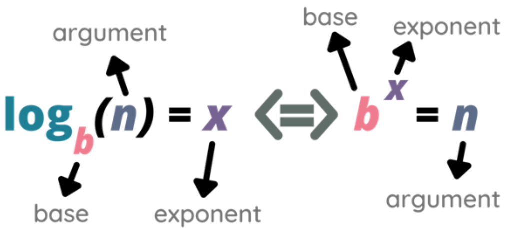
$$O(logn)$$

```
for (i = 10; i < n + 5; i *= 3)
        op();
```

10*3 = 30
30*3 = 90
90*3 = 270
…

The loop will multiply $i = 10$ by 3 until $10 \times 3^i \geq n + 5$. Solving for i, we get $i = \log_3 \dfrac{n+5}{10}$.

$10*3^0 = 10$
$10*3^1 = 30$
$10*3^2 = 90$
$10*3^3 = 270$
….



```
for (i = 10; i < n + 5; i *= 3)
        op();
```

i_0 $10 * 3^0$

i_1 $10 * 3^1$

i_2 $10 * 3^2$

i_k $10 * 3^k$

Loop stops when $\boldsymbol{i_k \geq \mathbf{n} + 5}$

$$10 * 3^k \geq n + 5$$

$$3^k \geq \frac{(n+5)}{10}$$

$$\log_3 3^k \geq \log_3 \frac{n+5}{10}$$

$$\mathrm{k} \geq \log_3 \frac{n+5}{10}$$

```
for (i = 1; i < n * n * n; i *= 2)
      op();
```

i_0 i_1 i_2 i_k

2^0 2^1 2^2 … 2^k

O(log n)

$$\log_2 n^3 = 3\log_2 n.$$

$$2^k \geq n^3$$

$$\log_2 2^k \geq \log_2 n^3$$

$$k \geq 3\log_2 n$$

$$O(logn)$$

$$\log m^n = n\log m$$

```
for (i = 10; i < n; i++)
        for (j = 0; j < n; j += 2)
                op();
```

$O(n^2)$

`op()` is called exactly $(n - 10) \times \dfrac{n}{2} = \frac{1}{2}n^2 - 5n$ times.

```
for (i = 0; i < n; i++)
     for (j = 0; j < 100; j++)
          op();
```

$O(n)$

`op()` is called $100n$ times, but we ignore the coefficient.

```
for (i = 0; i < n; i++) {
      for (j = 0; j < n; j++)
            op();

      for (j = 1; j < n; j *= 2);
            op();
}
```

$O(n^2)$

At each iteration of the outer loop, the first inner loop runs and then the second inner loop runs. Therefore, **op()** is called $n \times (n + \log n)$ $= n^2 + n \log n$ times, which is in the order of n^2.

```
for (i = 1; i <= n; i++)
    for (j = 1; j <= i; j ++)
        op();
```

$O(n^2)$

The inner loop performs 1 iteration when $i = 1$, and 2 iterations when $i = 2$, etc.

Therefore, `op()` is called $1 + 2 + \ldots + n$ times. This can be represented as a summation:

$$\sum_{i=0}^{n} i = \frac{n(n+1)}{2}$$

Eq. 1

Quiz

Find the running time complexity of the following snippet of codes. Provide an explanation or solution.

1

```
function FUN(int n)
    int m = n*n;
    for (int i = n/2; i > 1; i/=3) do
        for (int j = 0; j < m; j++) do
            Console.print(i+j)
        end for
    end for
    return 0
end function
```

2

```
function FUN(int n)
    while n > 0 do
        for (int i = n; i > 0; i/=3) do
            Console.print(n)
        end for
        n = n / 2
    end while
    return 0
end function
```

```
function FUN(int n)
    int m = n*n;
    for (int i = n/2; i > 1; i/=3) do
        for (int j = 0; j < m; j++) do
            Console.print(i+j)
        end for
    end for
    return 0
end function
```

$$\begin{array}{ccccc} i_0 & i_1 & i_2 & & i_k \\ \frac{n}{2}x\frac{1}{3^0} & \frac{n}{2}x\frac{1}{3^1} & \frac{n}{2}x\frac{1}{3^2} & \dots & \frac{n}{2}x\frac{1}{3^k} \end{array}$$

$O(n^2)$

$O(n^2 logn)$

Loop stops when i_k<1

$$\frac{n}{2}x\frac{1}{3^k} \leq 1$$

$$\frac{n}{2} \leq 3^k$$

$$\log_3 \frac{n}{2} \leq \log_3 3^k$$

$$\log_3 \frac{n}{2} \leq k\log_3 3$$

$$\log_3 \frac{n}{2} \leq k$$

$$O(logn)$$

```
function FUN(int n)
    while n > 0 do
        for (int i = n; i > 0; i/=3) do
            Console.print(n)
        end for
        n = n / 2
    end while
    return 0
end function
```

$$i_0 \quad i_1 \quad i_2 \quad \cdots \quad i_k$$

$$n * \frac{1}{3^0} \quad n * \frac{1}{3^1} \quad n * \frac{1}{3^2} \quad \cdots \quad n * \frac{1}{3^k}$$

Loop stops when i_k<=0

$$n * \frac{1}{3^k} \leq 1$$

$$n \leq 3^k$$

$$\log_3 n \leq \log_3 3^k$$

$$\log_3 n \leq k\log_3 3$$

$$\log_3 n \leq k$$

$$O(logn)$$

$$i_0 \quad i_1 \quad i_2 \quad \cdots \quad i_k$$

$$n * \frac{1}{2^0} \quad n * \frac{1}{2^1} \quad n * \frac{1}{2^2} \quad \cdots \quad n * \frac{1}{2^k}$$

$$n * \frac{1}{2^k} \leq 1$$

$$n \leq 2^k$$

$$\log_2 n \leq \log_2 2^k$$

$$\log_2 n \leq k$$

$$O(logn)$$