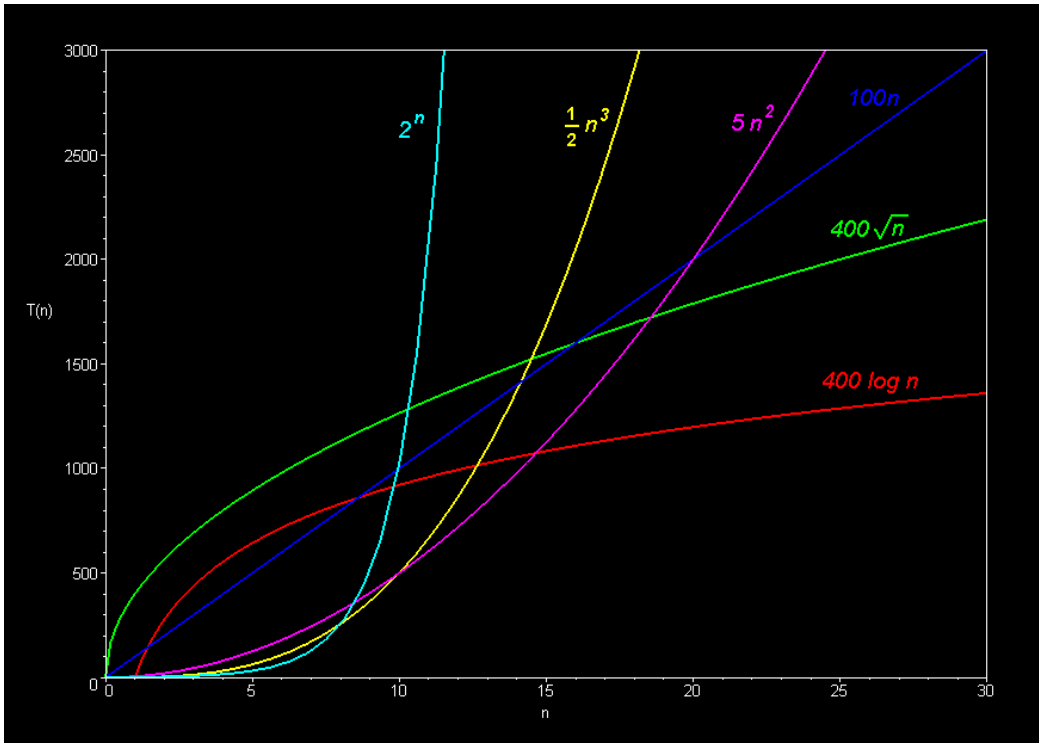
Data Structures and Algorithm Analysis (CSC317)



Week 2: Growth of Functions

Picture from
http://science.slc.edu/~jmarshall/courses/2002/spring/cs50/BigO/

First: Another loop invariant example

y=0
For i=0 to n
  y=y+2^i
Return y

(Try to find a loop invariant that holds at start of each iteration of for loop! We want y not as a function of adding y each iteration, but to know what y would be after 2, 3, 4 , … k iterations)

This will allow us to figure out what y is after the loop termination, and check correctness

First: Another loop invariant example

y=0
For i=0 to n
  y=y+2^i
Return y

(Try to find a loop invariant that holds at start of each iteration of for loop! We want y not as a function of adding y each iteration, but to know what y would be after 2, 3, 4 , … k iterations)

This will allow us to figure out what y is after the loop termination, and check correctness

Try going through a few iterations of the for loop in breakout groups and figuring out the value of y …

First: Another loop invariant example

y=0
For i=0 to n
  y=y+2^i
Return y

Let's try a few iterations

i=0 $y = 0$ (before first iteration for loop)

i=1 $y = 0 + 2^0 = 1$

i=2 $y = 1 + 2^1 = 3$

i=3 $y = 3 + 2^2 = 7$

Is there a pattern??

First: Another loop invariant example

y=0
For i=0 to n
  y=y+2^i
Return y

Let's try a few iterations

i=0 $y = 0$ (before first iteration for loop) $2^0 - 1$

i=1 $y = 0 + 2^0 = 1$ $2^1 - 1$

i=2 $y = 1 + 2^1 = 3$ $2^2 - 1$

i=3 $y = 3 + 2^2 = 7$ $2^3 - 1$

What is y for i=4??

First: Another loop invariant example

y=0
For i=0 to n
  y=y+2^i
Return y

Let's try a few iterations

i=0 $y = 0$ (before first iteration for loop) $2^0 - 1$

i=1 $y = 0 + 2^0 = 1$ $2^1 - 1$

i=2 $y = 1 + 2^1 = 3$ $2^2 - 1$ Pattern: $2^i - 1$

i=3 $y = 3 + 2^2 = 7$ $2^3 - 1$

First: Another loop invariant example

y=0
For i=0 to n
  y=y+2^i
Return y

Loop invariant: at the start of each iteration of the for loop,
$y=2^i-1$

i=0
y=0
Assert $y=2^i-1$
For i=0 to n
  y=y+2^i

- This can be proved inductively
- We manually did an assert
- Assert is common in many programming languages

First: Another loop invariant example

Python assert: AssertLoopInvariant.ipynb

In [7]:
```
i=0
y=0
assert y == (2**i - 1) # assert initial
n = 20
for i in range (0,n):
    assert(y == 2**(i) - 1) # true so far for i; assert for i
    print(y)
    print(2**(i) - 1)
    y = y + 2**i
    assert(y == 2**(i+1) - 1) # now assert for i+1
print(2**(n) - 1)
assert(y == 2**(n) - 1) # assert termination
```

First: Another loop invariant example

Python assert: AssertLoopInvariant.ipynb

```
0
0
1
1
3
3
7
7
15
15
31
31
63
63
127
127
255
255
511
511
1023
1023
2047
2047
4095
4095
8191
8191
16383
16383
32767
32767
65535
65535
131071
131071
262143
262143
524287
524287
1048575
```

First: Another loop invariant example

Python assert: AssertLoopInvariant.ipynb

In [3]:
```
i=0
y=0
assert y == (2**i - 1)
n = 20
for i in range (0,n):
    assert(y == 2**(i+1) - 1) # cannot assert yet for i+1
    print(y)
    print(2**(i) - 1)
    y = y + 2**i
    assert(y == 2**(i+1) - 1)
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-3-f947adf92b02> in <module>()
      4 n = 20
      5 for i in range (0,n):
----> 6     assert(y == 2**(i+1) - 1) # cannot assert yet
      7     print(y)
      8     print(2**(i) - 1)

AssertionError:
```

First: Another loop invariant example

Python assert: AssertLoopInvariant.ipynb

In [8]:
```
# if we had wrong loop invariant
i=0
y=0
n = 20
for i in range (0,n):
    assert(y == 2**(i)) # true so far for i?
    print(y)
    print(2**(i) - 1)
    y = y + 2**i
print(2**(n) - 1)
assert(y == 2**(n) - 1) # assert termination
```

```
---------------------------------------------------------------------------
-----
AssertionError                            Traceback (most recent call
last)
<ipython-input-8-d7ee22c282cd> in <module>()
      4 n = 20
      5 for i in range (0,n):
----> 6     assert(y == 2**(i)) # true so far for i?
      7     print(y)
      8     print(2**(i) - 1)

AssertionError:
```

Growth of functions

We've already been talking about "Grows as" for the sort examples, but what does this really mean?

We already know that:

- We ignore constants and low order terms; why?
- Asymptotic analysis: we focus on large input size; growth of function for large input; why do we care?

Complexity petting zoo

This is a petting zoo, because there are many more complexity classes, and we are only exploring the surface…

Complexity petting zoo (see notes of prof Burt Rosenberg: http://blog.cs.miami.edu/burt/2014/09/01/a-complexity-petting-zoo/)

Complexity classes

Constant time $T(1)$

Example? First number in an array
Also second number…

Complexity classes

$T(\log n)$

Example?

Complexity classes

$T(\log n)$

Example?

Binary search: Sorted array A; find value v between range low and high

A = [1 3 4 10 15 23 35 40 45]
Find v=4

Solution: Search in middle of array:
value found, or recursion left side, or recursion right half

Growth of functions

$T(n)$

Example?

Growth of functions

$T(n)$

Example?
Largest number in sequence
Sum of fixed sequence
Whenever you step through entire sequence or array
Even if you have to do this 20 times

Also … Merge function we did last class

Complexity classes

$T(n \log n)$

Example?
We've seen; merge sort…

Growth of functions

$T(n^2)$

Example?
We've seen; insertion sort…

Complexity classes

$T(n^3)$

Example?
Naive matrix multiplication (for an n by n matrix) is classical example; we shall see more later…

Complexity classes

All of these are polynomial time (class P)

$$T(n); T(n\log n); T(n^2); T(n^3)$$

$T(n^k)$ K nonnegative

Complexity classes

More than polynomial time??

Complexity classes

More than polynomial time? Exponential

$$T(2^n)$$

Complexity classes

What about this problem: subset sum problem?
How long to find a solution??

Input: set of integers size n
Output: is there a subset that sums to 0?

A={1; 4; -3; 2; 9; 7}
Is there a subset that sums to 0?

(Any subset; need not be continuous. Is there a subset here??)

Complexity classes

What about this problem: subset sum problem?
How long to find a solution??

Input: set of integers size n
Output: is there a subset that sums to 0?

A={1; 4; -3; 2; 9; 7}
Is there a subset that sums to 0?

Might take exponential time if we have to go through every possible subset (brute force)

Complexity classes

What about this problem: subset sum problem?

Input: set of integers size n
Output: is there a subset that sums to 0?

A={1; 4; -3; 2; 9; 7}
Is there a subset that sums to 0?

What about if I hand you a subset:
{1; -3; 2}
How long to verify if this sums to 0?

Complexity classes

What about this problem: subset sum problem?

Input: set of integers size n
Output: is there a subset that sums to 0?

A={1; 4; -3; 2; 9; 7}
Is there a subset that sums to 0?

What about if I hand you a subset:
{1; -3; 2}
How long to verify if this sums to 0? Polynomial, linear, time.

Complexity classes

Algorithms that are *verifiable* in polynomial time (good) are called NP class

But might take exponential number to go through every possible input (possibly bad)

Example: Subset sum problem

A={1; 4; -3; 2; 9; 7}
Is there a subset that sums to 0?

{1; -3; 2} is verifiable to sum to 0 quickly

Complexity classes

Class NP = Nondeterministic Polynomial

Algorithms that are verifiable in polynomial time (good)

But might take exponential number to go through every possible input! (possibly bad)

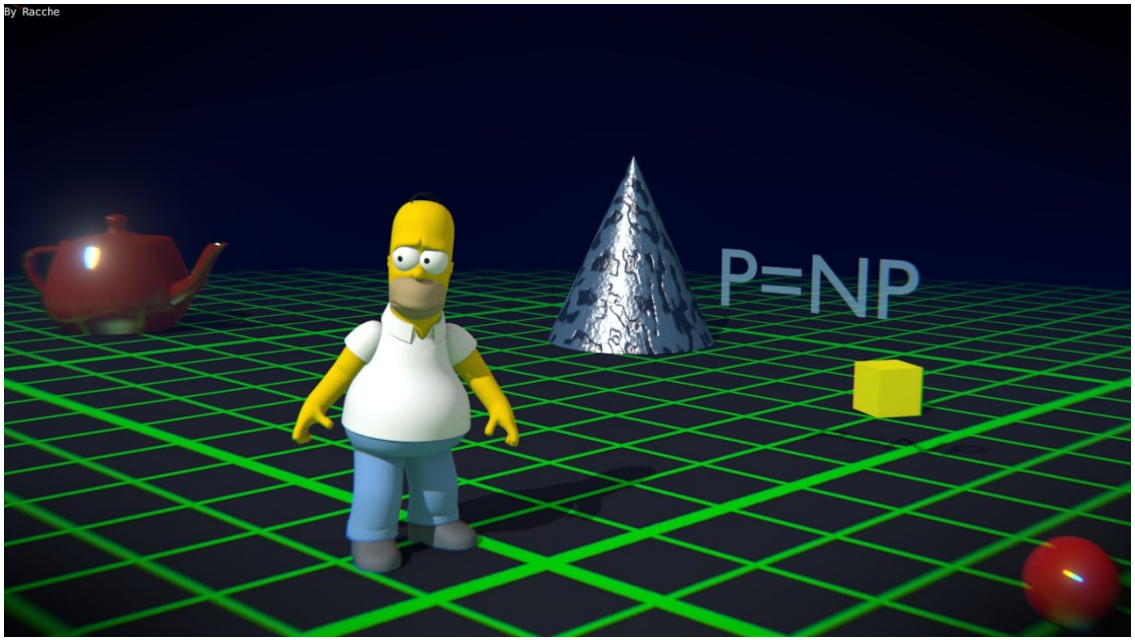
Nondeterministic = random = if I was magically handed solution.
Originally from nondeterministic Turing machine

Complexity classes

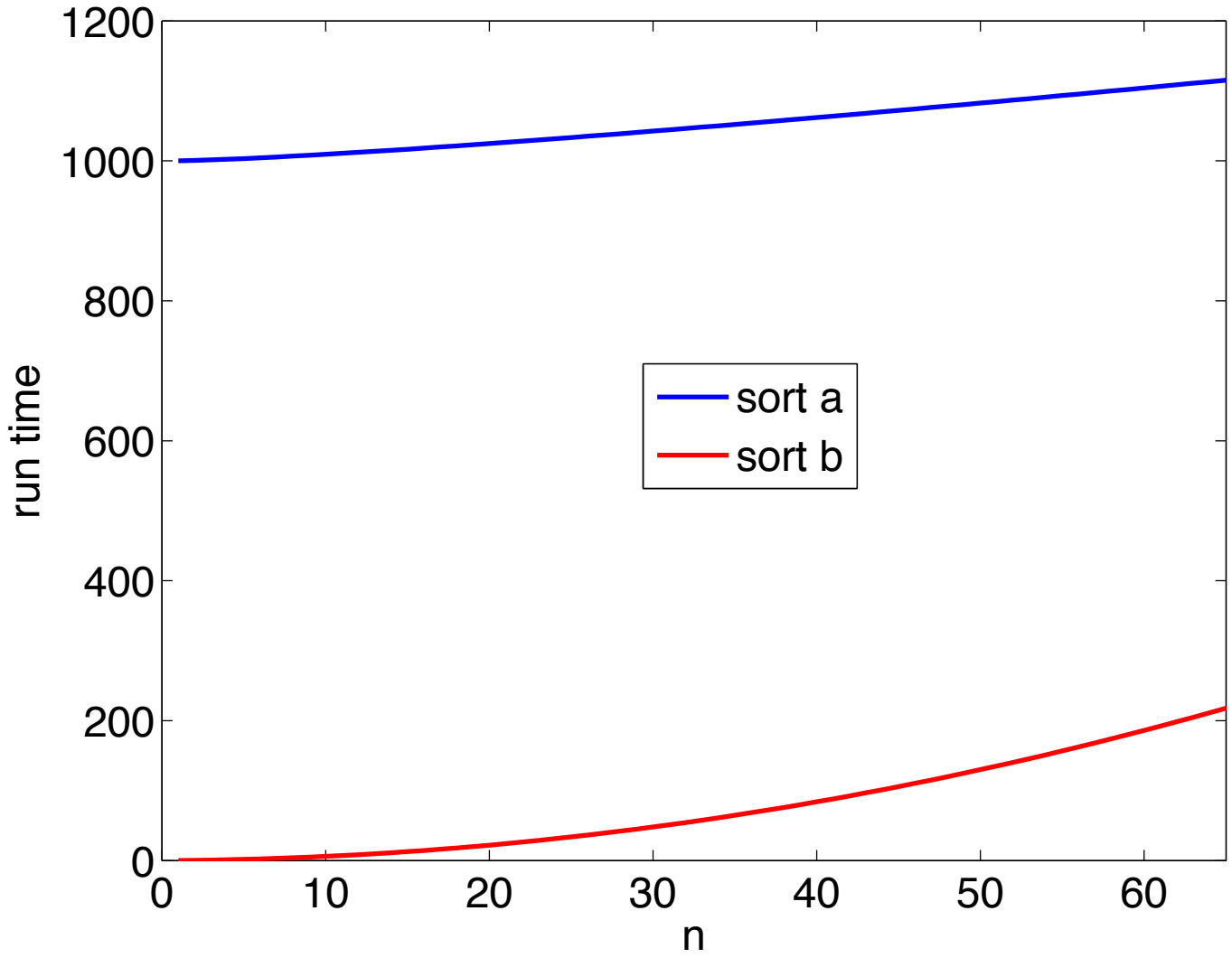
P = NP ???

Can problem that is quickly verifiable (ie, polynomial time) be quickly solved (ie, polynomial time)?

Unknown; Millenium prize problem

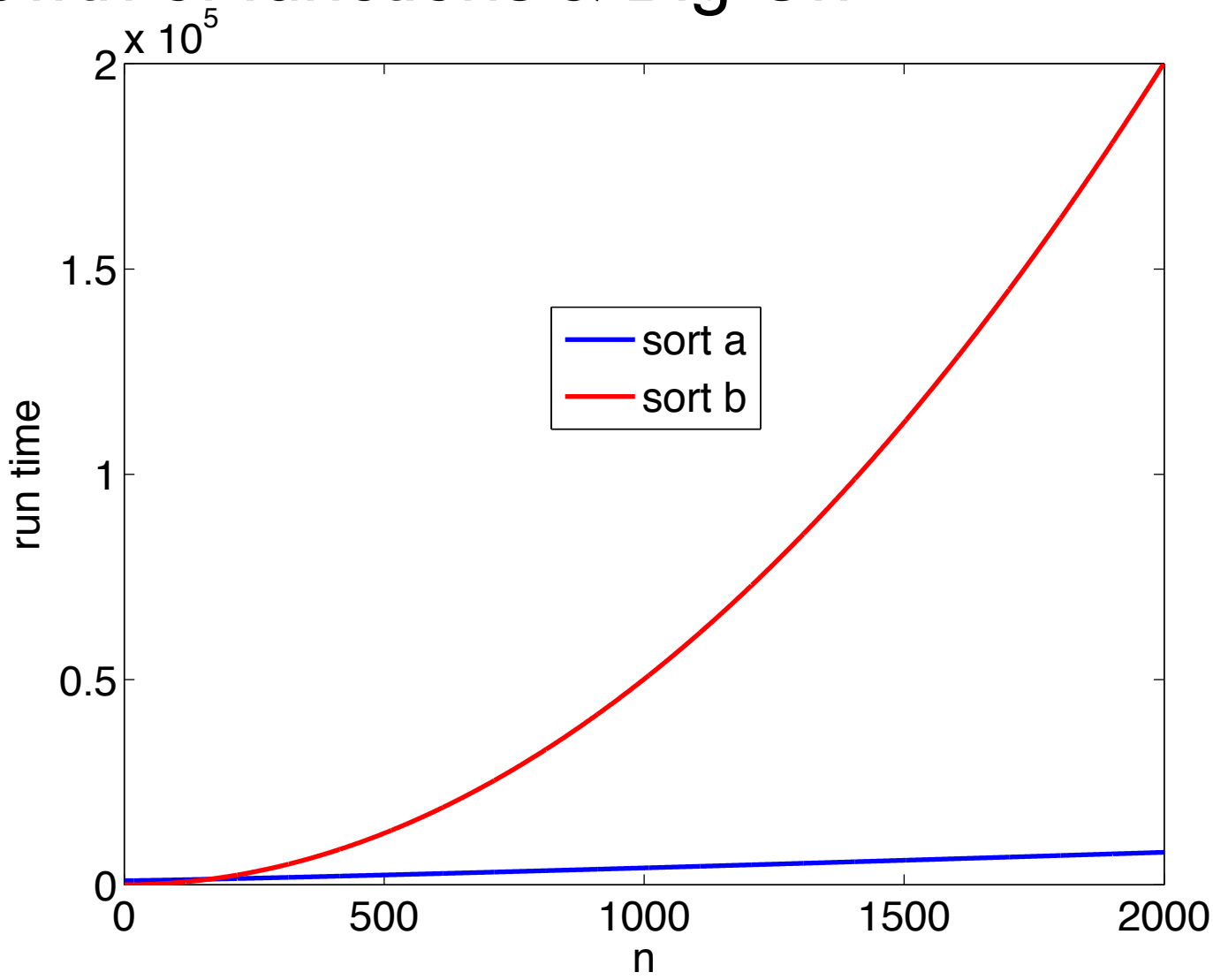


Growth of functions & Big Oh



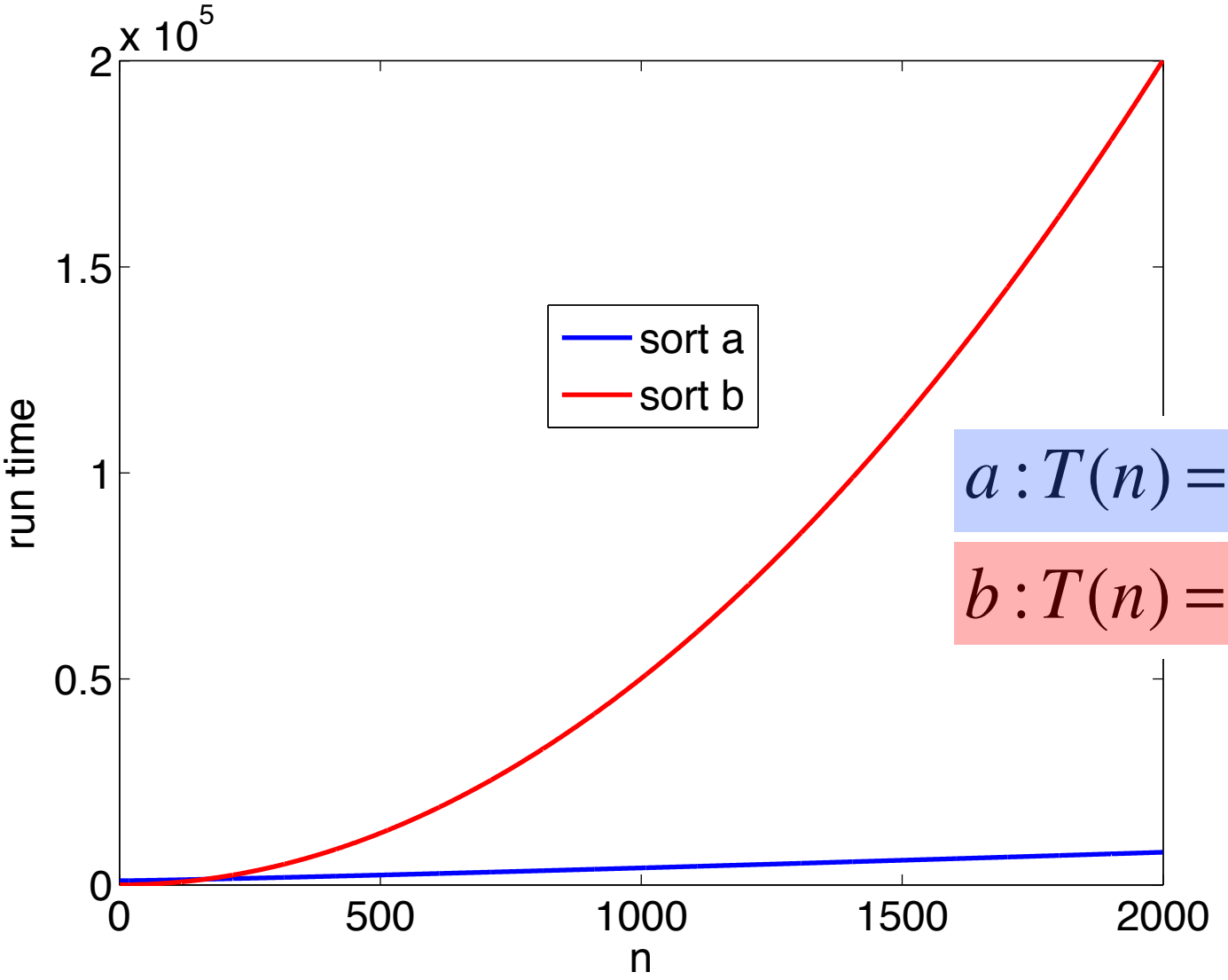
Which sort function is faster?

Growth of functions & Big Oh



Which sort function is faster?

Growth of functions & Big Oh



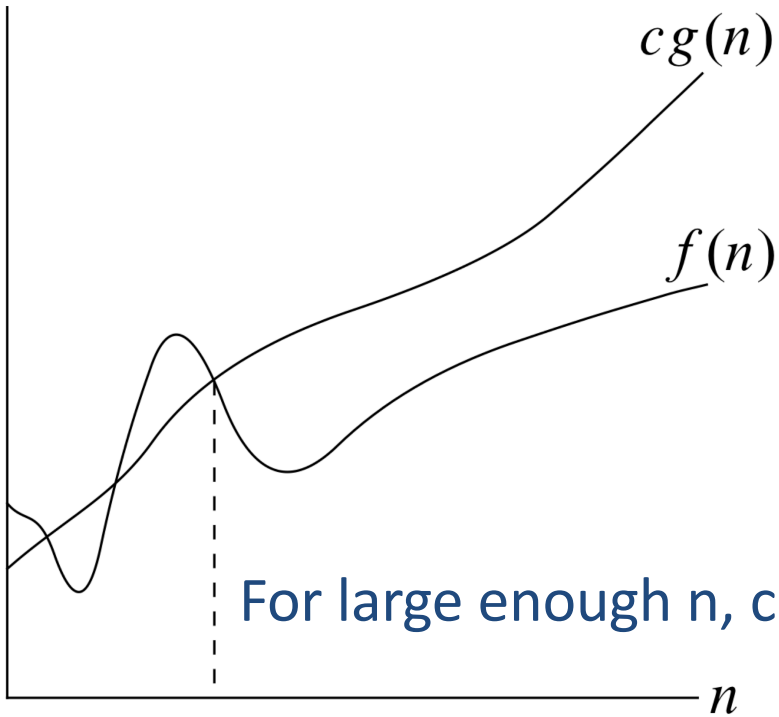
$$a : T(n) = n \log n + 1000$$

$$b : T(n) = 0.2n^2$$

Low asymptotic run time = faster

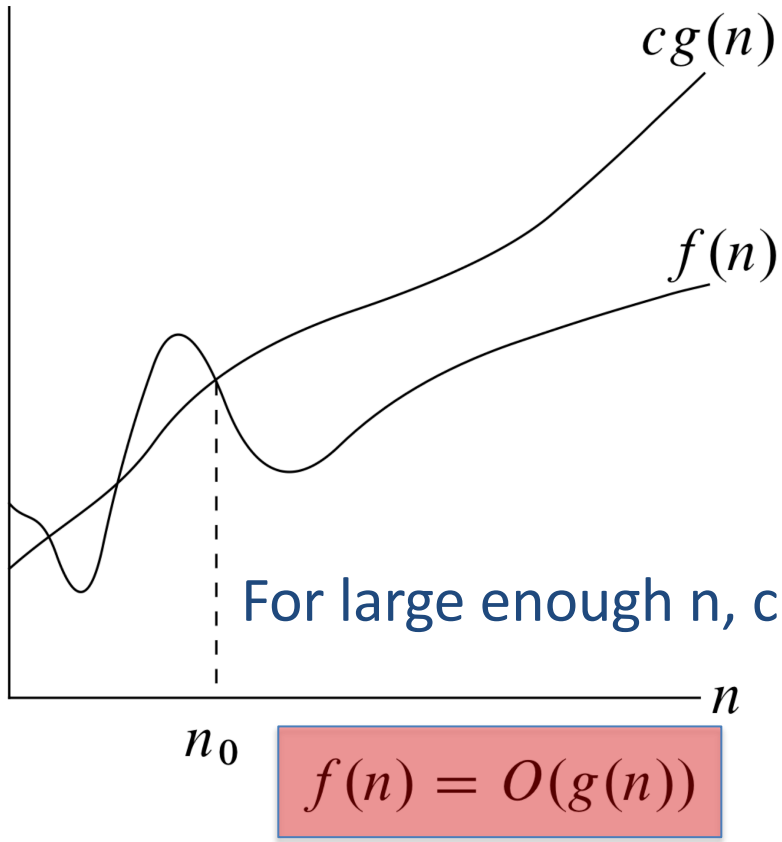
Big Oh notation

- Asymptotic upper bound; f(n) bounded from above by g(n) for large enough n (why do we care?)



Big Oh notation

- Asymptotic upper bound; f(n) bounded from above by g(n) for large enough n (why do we care?)



If an algorithm run time f(n) is bounded above by g(n) = n^2 or $3n^2$ or $500n^2$ (any constant c) that is $O(n^2)$, then complexity won't be larger than n^2 Running the algorithm is n^2 at most!

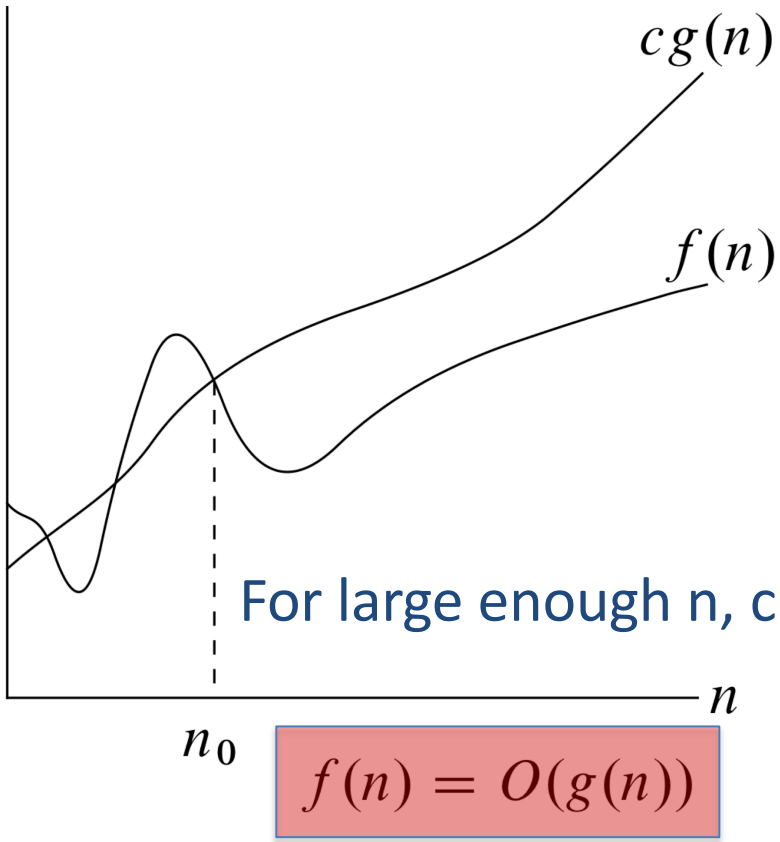
For large enough n, c g(n) is above f(n)

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

Big Oh notation

- Asymptotic upper bound; f(n) bounded from above by g(n) for large enough n (why do we care?)

These graphs should give intuition for what we mean by asymptotic upper bound



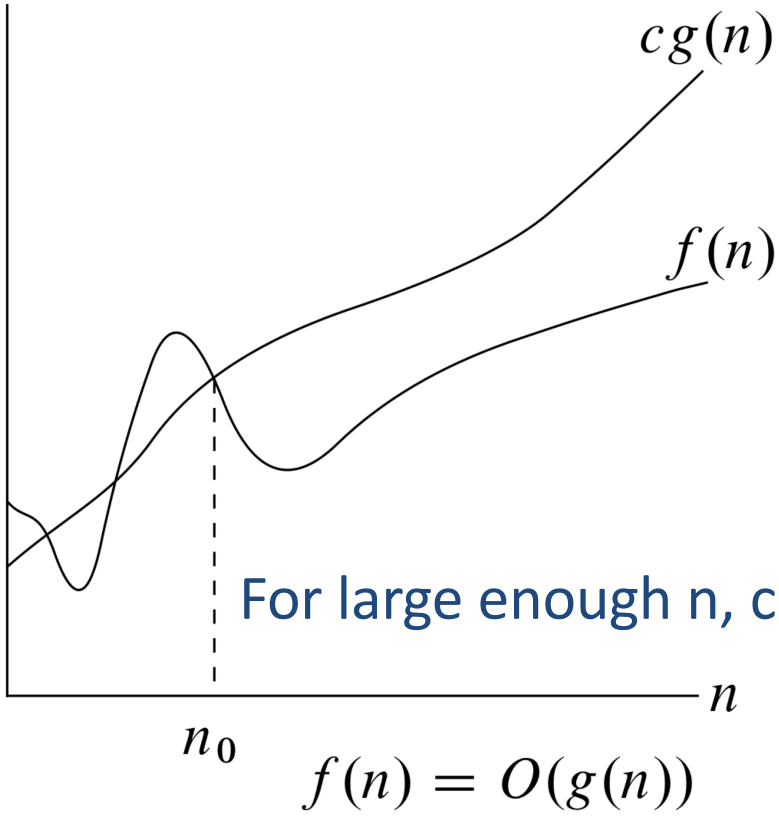
If an algorithm run time f(n) is bounded above by g(n) = n^2 or $3n^2$ or $500n^2$ (any constant c) that is $O(n^2)$, then complexity won't be larger than n^2 Running the algorithm is n^2 at most!

For large enough n, c g(n) is above f(n)

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

Big Oh notation

- Asymptotic upper bound; f(n) bounded from above by g(n) for large enough n (why do we care?)
- **Definition:** O(g(n))={ f(n): there exist positive constants c and n_0 such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$}

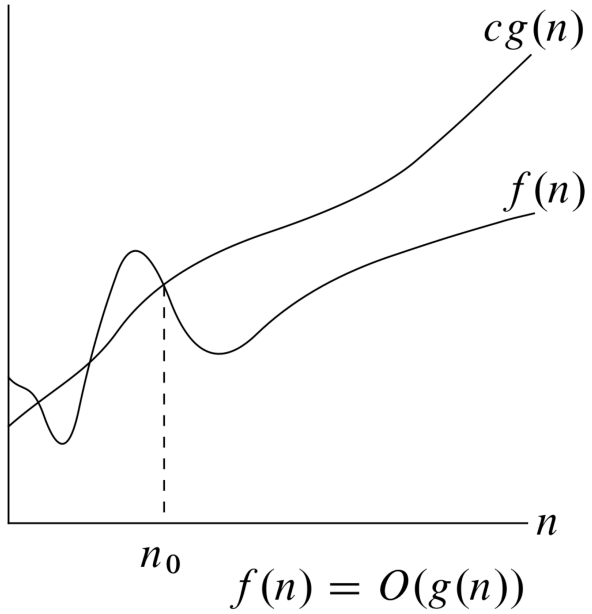


Big Oh notation

- Asymptotic upper bound; f(n) bounded from above by g(n) for large enough n
- **Definition:** O(g(n))={ f(n): there exist positive constants c and n_0 such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$}

There exist -> need to find c and n_0

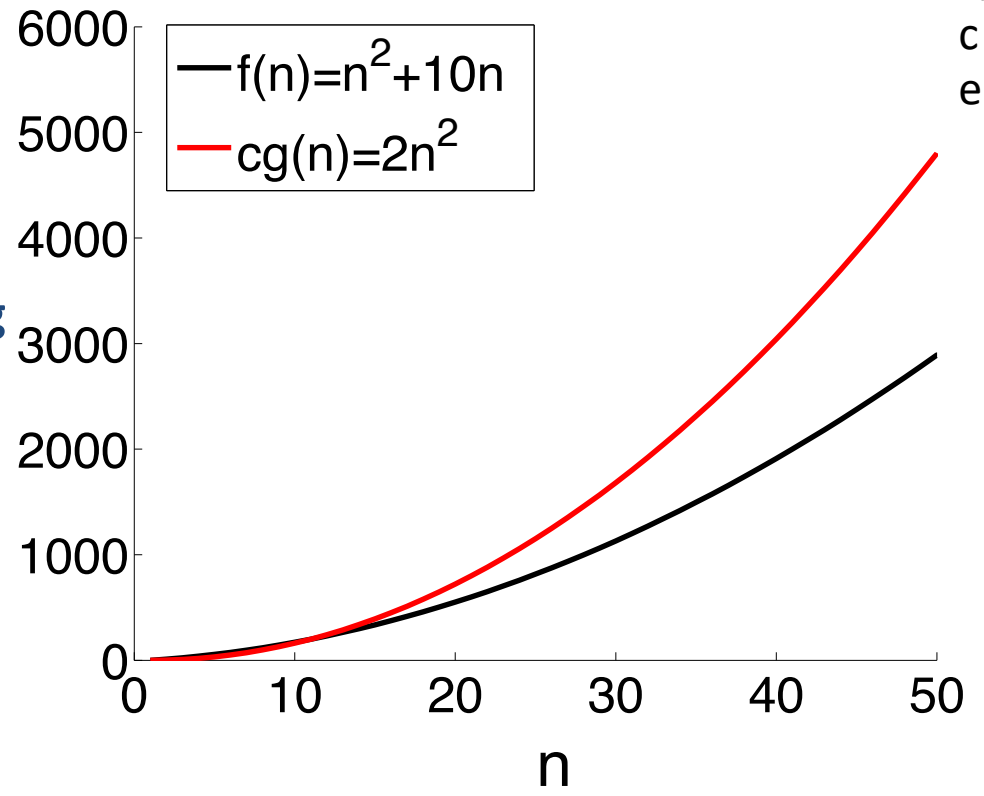
Enough to show one such pair that exists!



Big Oh notation

- Example: $f(n) = n^2 + 10n$ is $O(n^2)$

Can see this holds by plotting



f(n) bounded above by c g(n) or $2n^2$ for large enough n

Big Oh notation

- Example: $f(n) = n^2 + 10n$ is $O(n^2)$

Proof:

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

Big Oh notation

- Example: $f(n) = n^2 + 10n$ is $O(n^2)$

Proof:

$$f(n) = n^2 + 10n \leq n^2 + 10n^2$$

Big Oh notation

- Example: $f(n) = n^2 + 10n$ is $O(n^2)$

Proof:

$$f(n) = n^2 + 10n \leq n^2 + 10n^2 = 11n^2$$

Big Oh notation

- Example: $f(n) = n^2 + 10n$ is $O(n^2)$

Proof:

$f(n) = n^2 + 10n \leq n^2 + 10n^2 = 11n^2$

So there exists $c = 11$ and $n_0 = 1$ such that

$0 \leq n^2 + 10n \leq cn^2$

Big Oh notation

- **Definition:** O(g(n))={ f(n): there exist positive constants c and n_0 such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0$}

- Example of functions f(n) in $O(n^2)$

$$f(n) = n^2;$$

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

$$f(n) = n^2 + 1000n$$

- All bound above asymptotically by n^2
- Intuitively, constants and lower order don't matter…

Big Oh notation

- **Definition:** O(g(n))={ f(n): there exist positive constants c and n_0 such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0$}

- Example of functions f(n) in $O(n^2)$

$$f(n) = n^2;$$

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

$$f(n) = n^2 + 1000n$$

- What about?

$$f(n) = n;$$

Big Oh notation

- **Definition:** O(g(n))={ f(n): there exist positive constants c and n_0 such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$}

- What about?

  $f(n) = n;$

  Yes. $g(n) = n^2$ is **not a tight upper bound but it's an upper bound.**

  $f(n) = n \leq 1n^2$
  For all $n \geq 1$

Big Oh notation

- **Definition:** O(g(n))={ f(n): there exist positive constants c and n_0 such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$}

- What about?

$$f(n) = n;$$

Yes. $g(n) = n^2$ is **not a tight upper bound but it's an upper bound.**

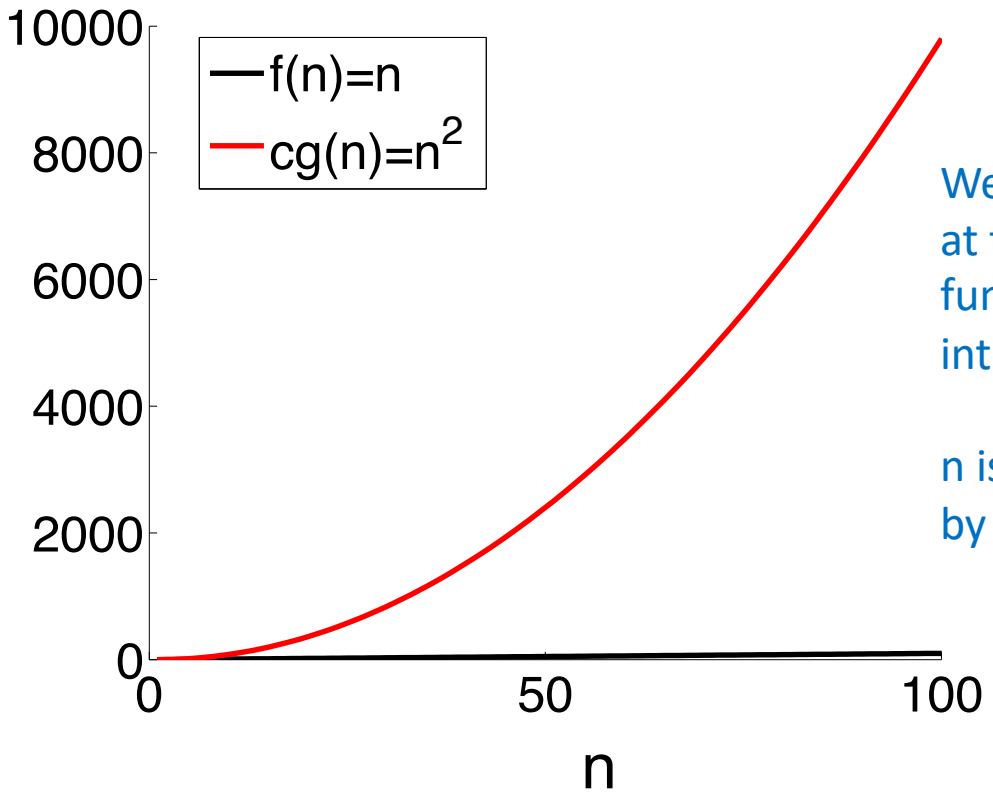
$$n \leq 1n^2$$

For all $n \geq 1$

There thus exists $c = 1; n_0 = 1$ such that $0 \leq f(n) \leq g(n)$

Big Oh notation

- Example: $f(n) = n$ is $O(n^2)$



We can again look at this by graphing the functions and get the intuition

n is clearly bounded above by n^2

Big Oh notation

- **Example:** $f(n) = a_k n^k + .. + a_1 n^1 + a_0$

  Then

  $$f(n) = O(n^k)$$

- Intuition: we can ignore lower order terms and constants

Big Oh notation

- **Example:** $f(n) = a_k n^k + .. + a_1 n^1 + a_0$

  Then

  $$f(n) = O(n^k)$$

- Proof : we want to find $n_0; c$ such that $f(n) \leq cn^k$

Big Oh notation

- **Example:** $f(n) = a_k n^k + .. + a_1 n^1 + a_0;$

$$a_k > 0$$

Then: $f(n) = O(n^k)$

- Proof : we want to find $n_0; c$ such that $f(n) \le cn^k$

$$f(n) = a_k n^k + .. + a_1 n^1 + a_0$$

Big Oh notation

- **Example:** $f(n) = a_k n^k + .. + a_1 n^1 + a_0;$

$$a_k > 0$$

Then: $f(n) = O(n^k)$

- Proof : we want to find $n_0; c$ such that $f(n) \le cn^k$

$$f(n) = a_k n^k + .. + a_1 n^1 + a_0$$

$$\le |a_k| n^k + .. + |a_1| n^1 + |a_0|$$

Big Oh notation

- **Example:** $f(n) = a_k n^k + .. + a_1 n^1 + a_0;$

$$a_k > 0$$

Then: $f(n) = O(n^k)$

- Proof : we want to find $n_0; c$ such that $f(n) \le cn^k$

$$f(n) = a_k n^k + .. + a_1 n^1 + a_0$$

$$\le |a_k| n^k + .. + |a_1| n^1 + |a_0|$$

$$\le |a_k| n^k + .. + |a_1| n^k + |a_0| n^k = (|a_k| + .. |a_1| + |a_0|) n^k$$

Big Oh notation

- **Example:** $f(n) = a_k n^k + .. + a_1 n^1 + a_0;$

$$a_k > 0$$

Then: $f(n) = O(n^k)$

- Proof : we want to find $n_0; c$ such that $f(n) \le cn^k$

$$f(n) = a_k n^k + .. + a_1 n^1 + a_0$$

$$\le |a_k| n^k + .. + |a_1| n^1 + |a_0|$$

$$\le |a_k| n^k + .. + |a_1| n^k + |a_0| n^k = (|a_k| + .. |a_1| + |a_0|) n^k$$

What are? $n_0; c$

Big Oh notation

- **Example:** $f(n) = a_k n^k + .. + a_1 n^1 + a_0;$

$$a_k > 0$$

Then: $f(n) = O(n^k)$

- Proof : we want to find $n_0; c$ such that $f(n) \le cn^k$

$$f(n) = a_k n^k + .. + a_1 n^1 + a_0$$

$$\le |a_k| n^k + .. + |a_1| n^1 + |a_0|$$

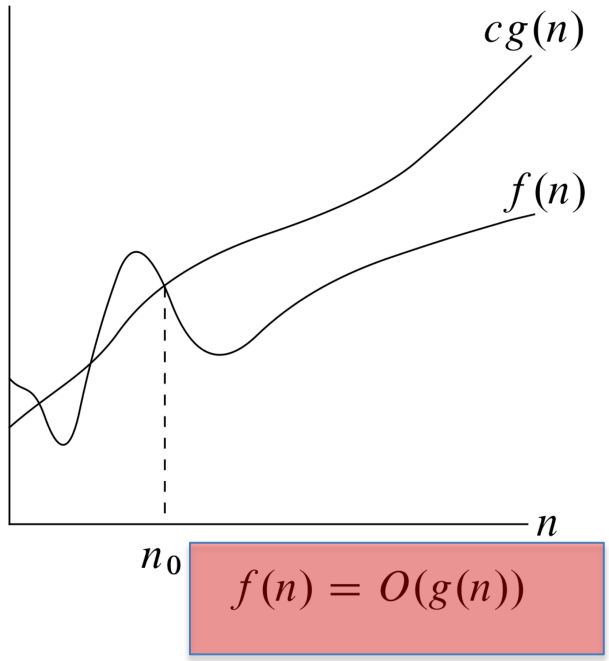
$$\le |a_k| n^k + .. + |a_1| n^k + |a_0| n^k = (|a_k| + .. |a_1| + |a_0|) n^k$$

What are? $n_0; c$

Holds for all $n \ge 1$

Big Oh: Most commonly used!

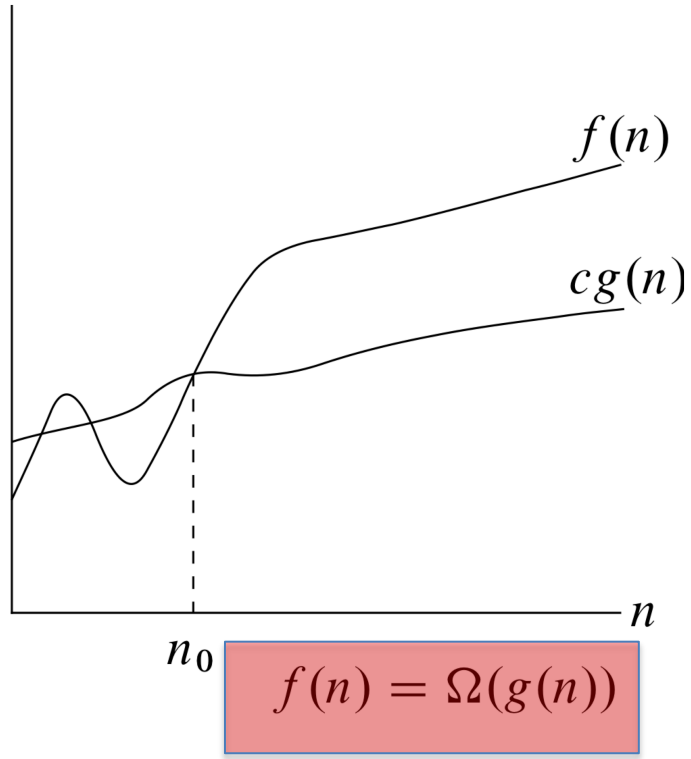
- Asymptotic upper bound; f(n) bounded from above by g(n) for large enough n
- **Definition:** O(g(n))={ f(n): there exist positive constants c and n_0 such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$ }



But there are other bounds

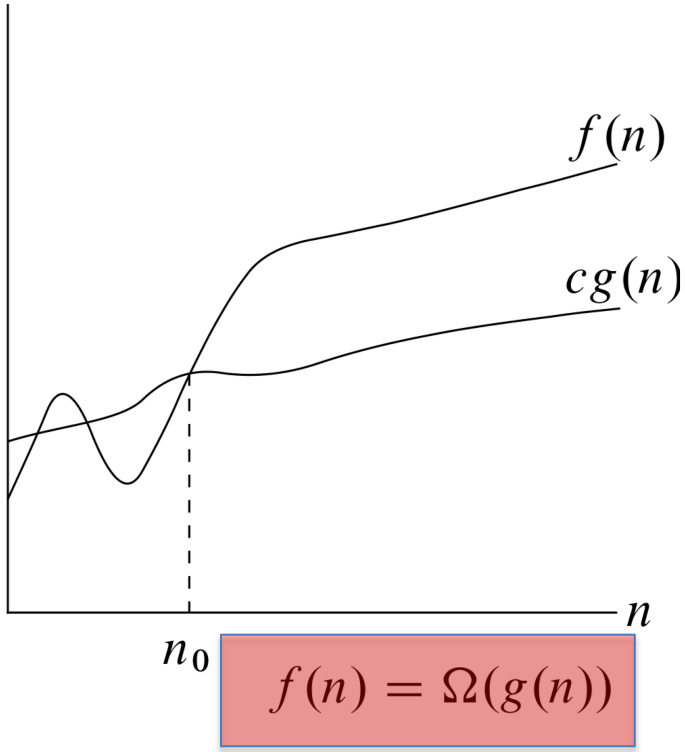
Big Omega

- Asymptotic lower bound; f(n) bounded from below by g(n) for large enough n



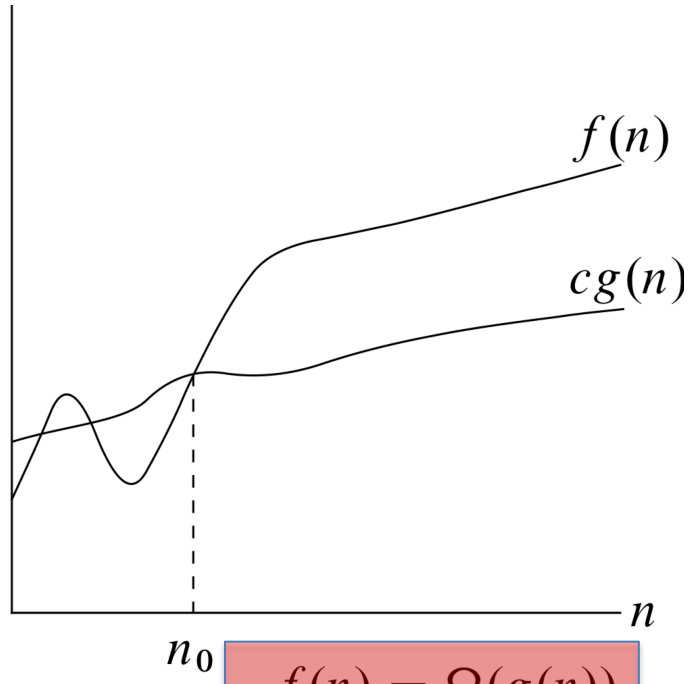
Big Omega

- Asymptotic lower bound; f(n) bounded from below by g(n) for large enough n
- **Definition:** $\Omega(g(n))$={ f(n): there exist positive constants c and n_0 such that $0 \leq cg(n) \leq f(n)$ for all $n \geq n_0$}



Big Omega

- Asymptotic lower bound; f(n) bounded from below by g(n) for large enough n

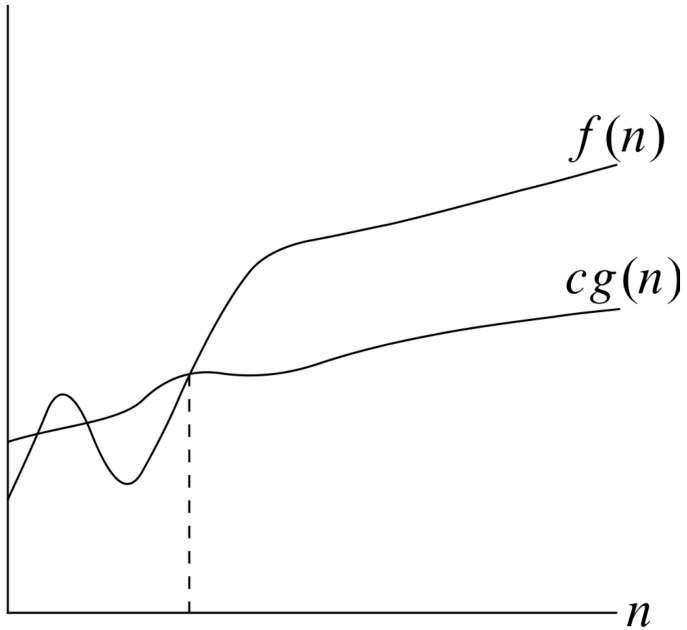


$f(n) = \Omega(g(n))$

Why is this less often used?

Big Omega

- Asymptotic lower bound; f(n) bounded from below by g(n) for large enough n



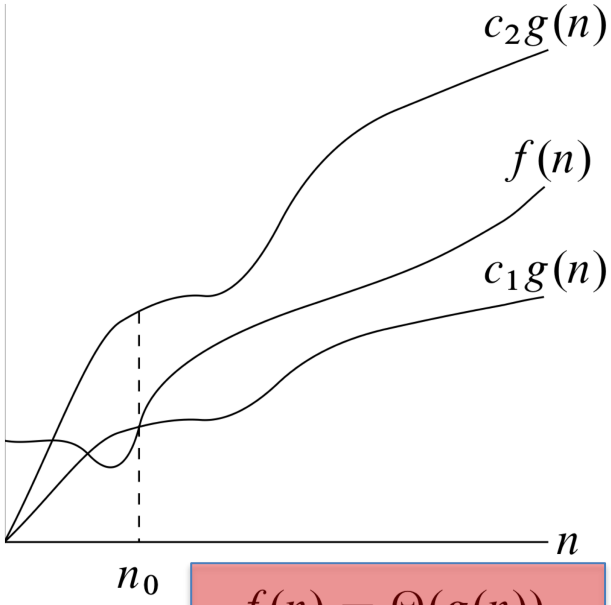
$f(n) = \Omega(g(n))$

Example:
If $f(n) = n^{100}$
bounded below by
g(n)=1, that's less
useful to know!!
Like if I said Insertion
Sort is bounded below
By g(n)=1

Why is this less often used?

Big Theta

- Asymptotic tight bound; f(n) bounded from below and above by g(n) for large enough n

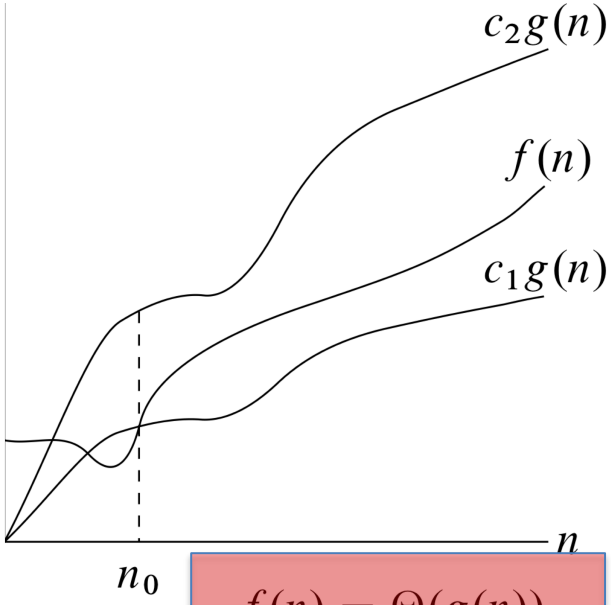


$f(n) = \Theta(g(n))$

Stronger statement (note literature sometimes sloppy and says Oh when actually Theta)

Big Theta

- Asymptotic tight bound; f(n) bounded from below and above by g(n) for large enough n
- **Definition:** $\Theta(g(n))$={ f(n): there exist positive constants $c_1; c_2$ and n_0 such that $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ for all $n \ge n_0$}

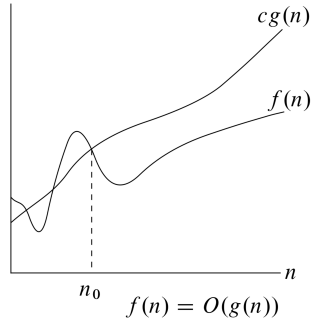


Stronger statement (note literature sometimes sloppy and says Oh when actually Theta)

Summary Oh, Omega, Theta

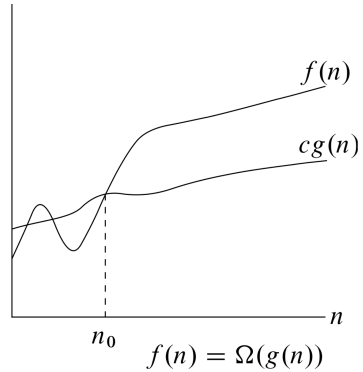
Oh

- $O(n)$ asymptotic upper, like $f(n) \leq cg(n)$



Omega

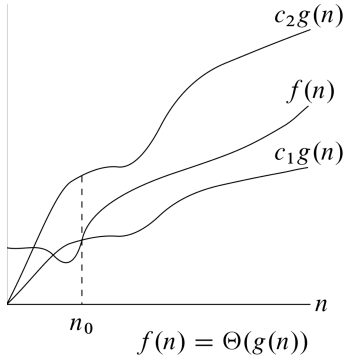
- $\Omega(n)$ asymptotic lower, like $f(n) \geq cg(n)$



Theta

- $\Theta(n)$ asymptotic tight, like

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



Examples Oh, Omega, Theta

Polls…

Examples Oh, Omega, Theta

- Example of functions f(n) in $O(n^2)$

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

- Example of functions f(n) in $\Omega(n^2)$

$$f(n) = n^2; f(n) = n^2 + n; f(n) = n^5$$

- Example of functions f(n) in $\Theta(n^2)$

$$f(n) = n^2; f(n) = n^2 - n$$

More on Oh, Omega, Theta

Theorem: $f(n) = \theta(n)$

if and only if (iff)

More on Oh, Omega, Theta

Theorem: $f(n) = \theta(n)$

if and only if (iff)

$f(n) = O(n)$ and $f(n) = \Omega(n)$