LECTURE 3: ALGORITHM ANALYSIS

Harold Soh
harold@comp.nus.edu.sg

QUESTIONS BEFORE WE GET STARTED?

LEARNING OUTCOMES

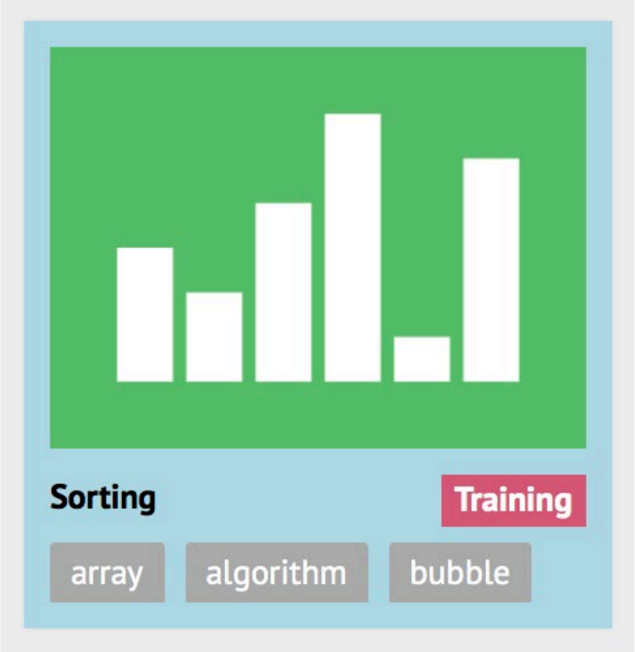
By the end of this session, you should be able to:

- **Determine** the **computational complexity** of an algorithm under the standard sequential computation model
- **Use Big-Oh Notation** to describe algorithm performance

Poll Everywhere

https://bit.ly/2LvG9bq

DID YOU DO YOUR HOMEWORK?



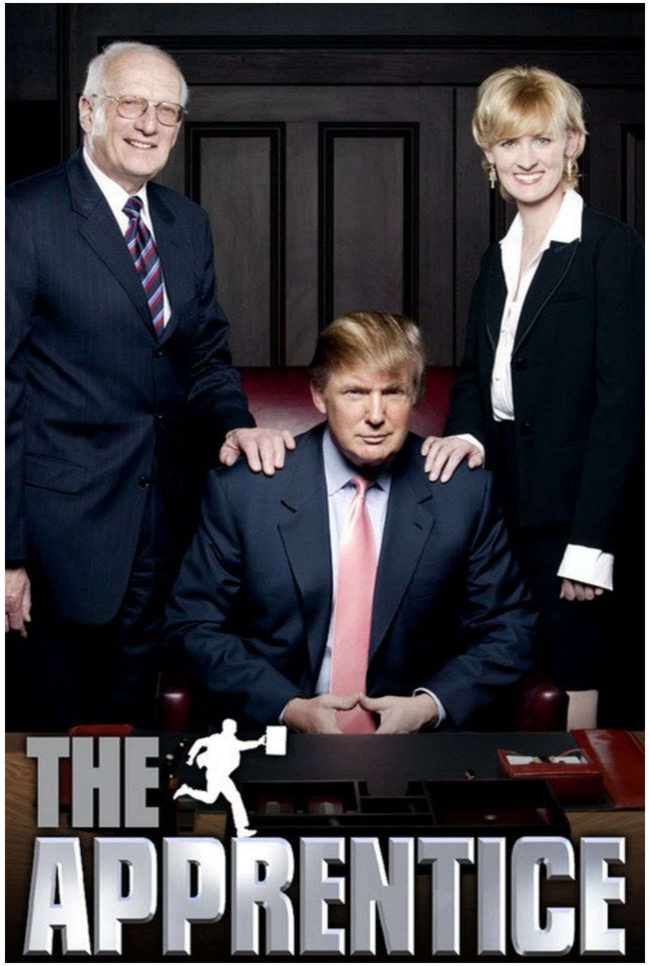
Did you revise the sorting material on Visualgo?

A. Of course!
B. Of course ... not!
C. well, kind of half way...
D. Ummm.. Visualgo?

PROBLEM: CUSTOMER LOYALTY REWARDS

Get a list of customers ordered by their purchasing spend. Reward the n who spent the most.

BOSS HAS AN SOLUTION:



BogoSort

```
while items is not sorted
    permute(items)
```

NARUTO'S SOLUTION

```
mark first element as sorted
for each unsorted element X
  'extract' the element X
  for j = lastSortedIndex down to 0
    if current element j > X
      move sorted element to the right by 1
    break loop and insert X here
```

Poll Everywhere
https://bit.ly/2LvG9bq

NARUTO'S SOLUTION

What kind of sort is this?

A. Bubble Sort
B. Insertion Sort
C. Merge Sort
D. I obviously didn't do my homework

```
mark first element as sorted
for each unsorted element X
  'extract' the element X
  for j = lastSortedIndex down to 0
    if current element j > X
      move sorted element to the right by 1
    break loop and insert X here
```

HOW DO WE GET AN IDEA OF WHICH IS BETTER?

without implementing and running the two algorithms on all kinds of data?

Algorithm Analysis

ALGORITHM ANALYSIS IN A NUTSHELL

How long will my program take?

How much memory will it consume?

BUT OTHER CONCERNS MAY BE RELEVANT:

In general, we want to measure usage of some **resource**:

- communication bandwidth
- number of processing elements required
- human computation
- etc.

but we will focus on time and memory (space)

A SIMPLE MODEL OF COMPUTATION

Poll Everywhere

https://bit.ly/2LvG9bq

Random-Access Machine (RAM)

- instructions are sequential (no concurrency)

Each operation takes some constant amount of time

What are instructions?

- arithmetic operations (+,-, /, * etc.)
- control (branches, function call & returns)

Simple memory: no caching

Is exponentiation a^k considered one operation?

A. Yes

B. No

C. It's up to us to define in our model.

Poll Everywhere
https://bit.ly/2LvG9bq

DO COMMENTS COUNT?

```
int a = 0;
// the following is a loop
for (int i=0; i<10; i++) {
    a = a + 2;
    A[i] = a;
}
```

Do comments count?

A. Yes
B. No
C. Only on Facebook.
D. Only on Piazza

RUNNING TIME

Number of primitive instructions / “steps” executed.

Code	Cost	Times
`int a = 0;`	c_1	
`for (int i=0; i<n; i++) {`	c_2	
`    a = a + 2;`	c_3	
`    A[i] = a;`	c_4	
`}`		

RUNNING TIME

Number of primitive instructions / "steps" executed.

Code	Cost	Times
`int a = 0;`	c_1	1
`for (int i=0; i<n; i++) {`	c_2	
`    a = a + 2;`	c_3	
`    A[i] = a;`	c_4	
`}`		

RUNNING TIME

Number of primitive instructions / "steps" executed.

Code	Cost	Times
`int a = 0;`	c_1	1
`for (int i=0; i<n; i++) {`	c_2	n
`    a = a + 2;`	c_3	
`    A[i] = a;`	c_4	
`}`		

RUNNING TIME

Number of primitive instructions / “steps” executed.

Code	**Cost**	**Times**
`int a = 0;`	c_1	1
`for (int i=0; i<n; i++) {`	c_2	n
`    a = a + 2;`	c_3	n
`    A[i] = a;`	c_4	
`}`		

RUNNING TIME

Number of primitive instructions / “steps” executed.

Code	**Cost**	**Times**
`int a = 0;`	c_1	1
`for (int i=0; i<n; i++) {`	c_2	n
`    a = a + 2;`	c_3	n
`    A[i] = a;`	c_4	n
`}`		

RUNNING TIME

Number of primitive instructions / “steps” executed.

Code	**Cost**	**Times**
`int a = 0;`	c_1	1
`for (int i=0; i<n; i++) {`	c_2	n
`    a = a + 2;`	c_3	n
`    A[i] = a;`	c_4	n
`}`		

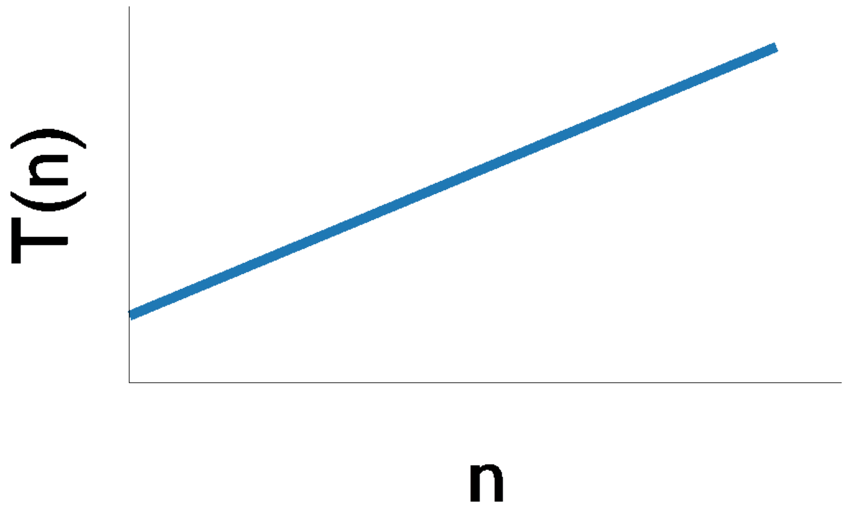
$$T(n) = c_1 + nc_2 + nc_3 + nc_4 = cn + c_1$$

RUNNING TIME

Number of primitive instructions / "steps" executed.

$$T(n) = cn + c_1$$

is a linear function



WHICH TAKES LONGER?

Poll Everywhere

https://bit.ly/2LvG9bq

```
void pushAll(int n) {
    for (int i=0; i<= 100*n; i++) {
        stack.push(i);
    }
}
```

```
void pushAdd(int n) {
    for (int i=0; i<= n; i++) {
        for (int j=0; j<= n; j++) {
            stack.push(i+j);
        }
    }
}
```

Which takes longer?

A. pushAll
B. pushAdd
C. Runs the same.
D. It depends!

ANALYSIS OF PUSHALL

```
void pushAll(int n) {
    for (int i=0; i<= 100*n; i++) {
        stack.push(i);
    }
}
```

Cost	Times
c_1	1
c_2	$100n$
c_3	$100n$

$$T(n) = 100cn + c_1$$

ANALYSIS OF PUSHADD

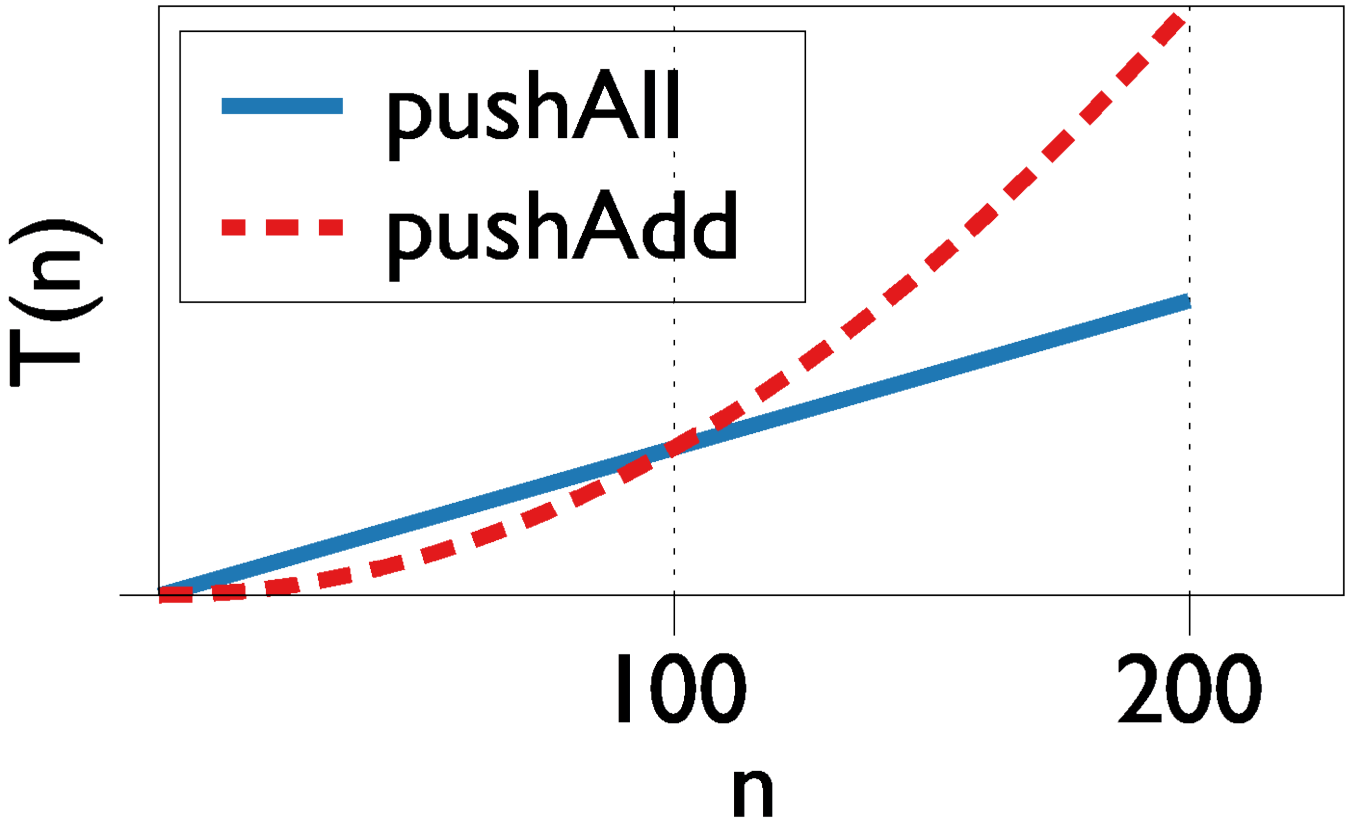
```
void pushAdd(int n) {
    for (int i=0; i<= n; i++) {
        for (int j=0; j<= n; j++) {
            stack.push(i+j);
        }
    }
}
```

Cost	Times
c_1	1
c_2	n
c_3	n^2
c_4	n^2

$$T(n) = c_1 + nc_2 + n^2c_3 + n^2c_4 = cn^2 + c_2n + c_1$$

is a **quadratic** function

IF ALL THE CONSTANTS = 1



Poll Everywhere

https://bit.ly/2LvG9bq

WHICH TAKES LONGER?

```
void pushAll(int n) {
    for (int i=0; i<= 100*n; i++) {
        stack.push(i);
    }
}
```

```
void pushAdd(int n) {
    for (int i=0; i<= n; i++) {
        for (int j=0; j<= n; j++) {
            stack.push(i+j);
        }
    }
}
```

Which takes longer if n is large ($n > 1000$)?

A. pushAll
B. pushAdd
C. Runs the same.
D. It depends!

WHICH TAKES **MORE MEMORY** ON THE STACK?

Poll Everywhere

https://bit.ly/2LvG9bq

```
void pushAll(int n) {
    for (int i=0; i<= 100*n; i++) {
        stack.push(i);
    }
}
```

```
void pushAdd(int n) {
    for (int i=0; i<= n; i++) {
        for (int j=0; j<= n; j++) {
            stack.push(i+j);
        }
    }
}
```

Which takes more memory if n is large ($n > 1000$)?

A. pushAll
B. pushAdd
C. takes the same amount of memory
D. Naruto knows!

ANALYSIS OF PUSHALL

Code	Cost	Times
`void pushAll(int n) {`		
`    for (int i=0; i<= 100*n; i++) {`		
`        stack.push(i);`	c	$100n$
`    }`		
`}`		

$$S(n) = 100cn$$

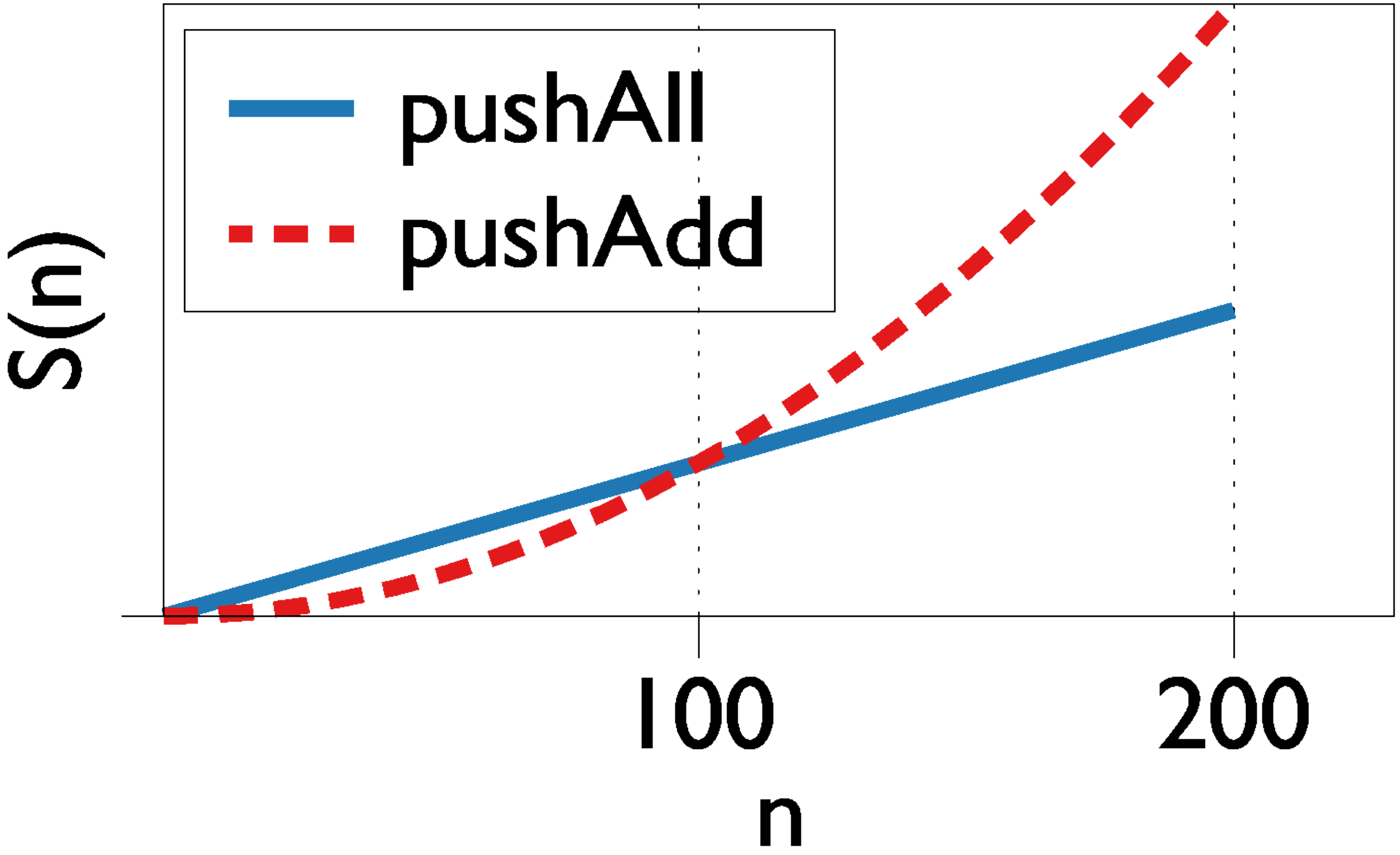
ANALYSIS OF PUSHADD

```
void pushAdd(int n) {
    for (int i=0; i<= n; i++) {
        for (int j=0; j<= n; j++) {
            stack.push(i+j);
        }
    }
}
```

Cost	Times
c	n^2

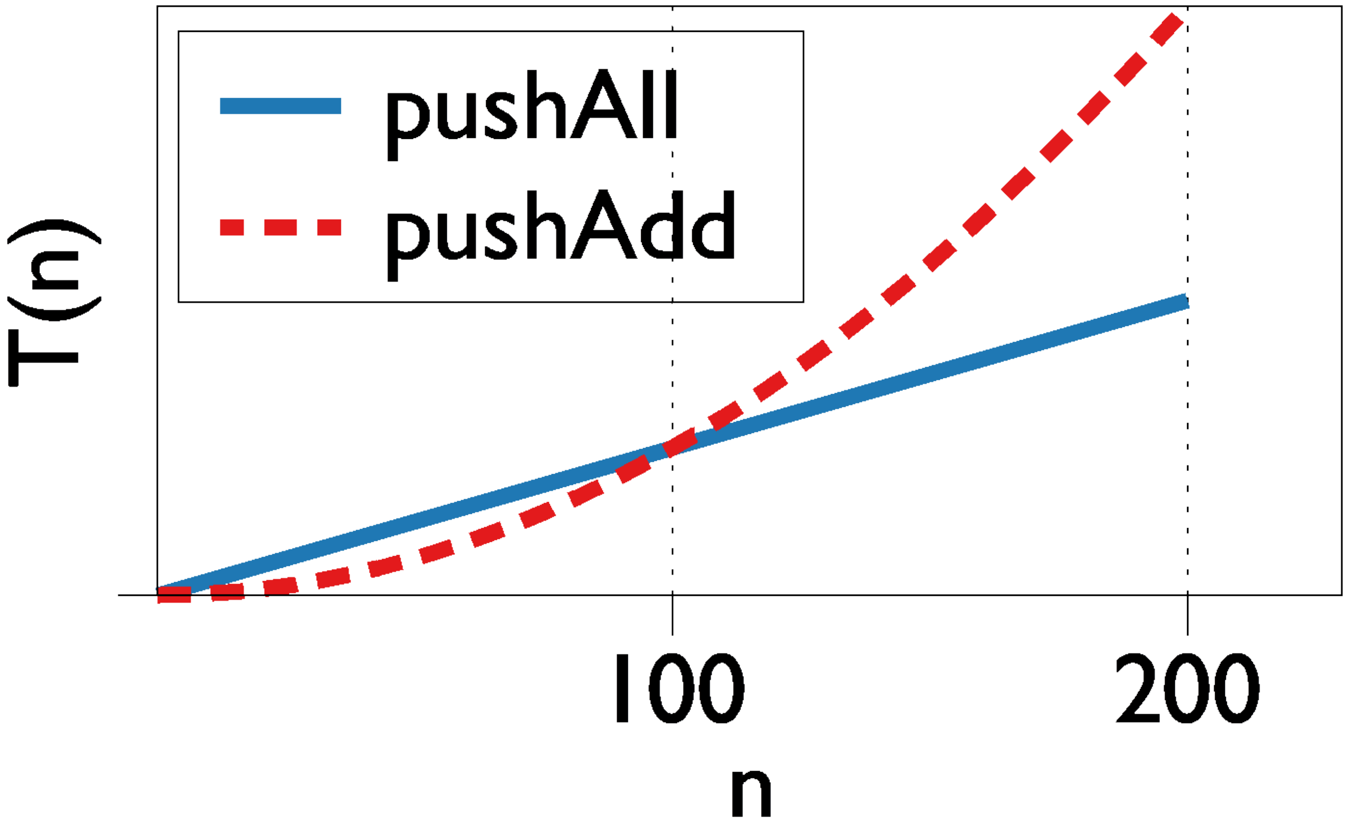
$$S(n) = cn^2$$

SIMILAR GRAPH FOR MEMORY IF C = 1

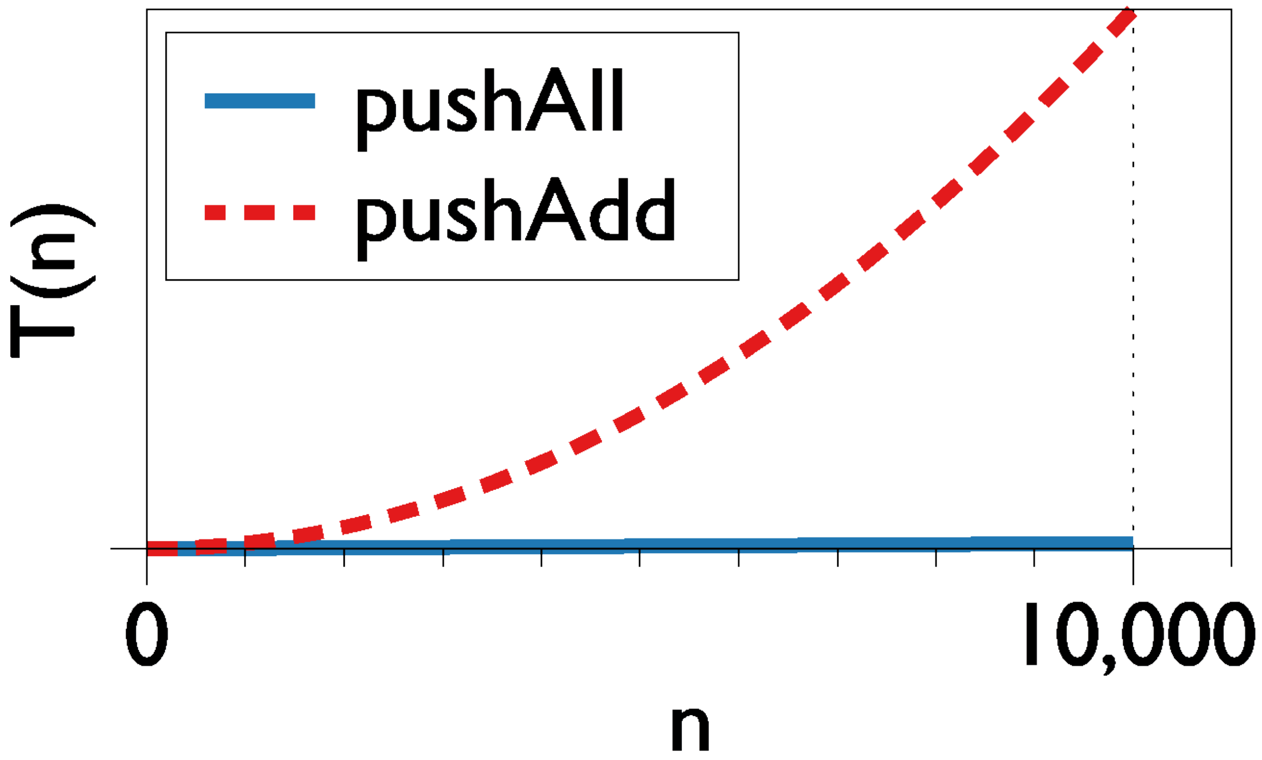


QUESTIONS?

ORDER OF GROWTH



ORDER OF GROWTH



$$T(n) = n^2 + c_2 n$$

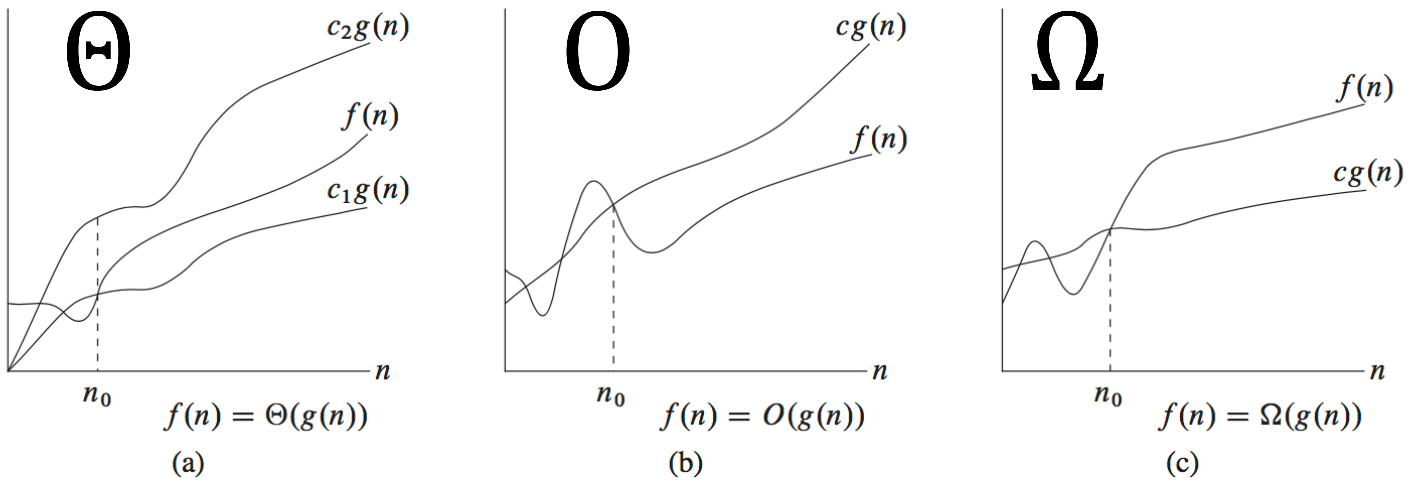
as $n \to \infty$, the term $c_2 n$ becomes insignificant compared to n^2.

We can say the $T(n)$ is "asymptotically equivalent" to n^2.

ASYMPTOTIC EFFICIENCY & ORDER OF GROWTH

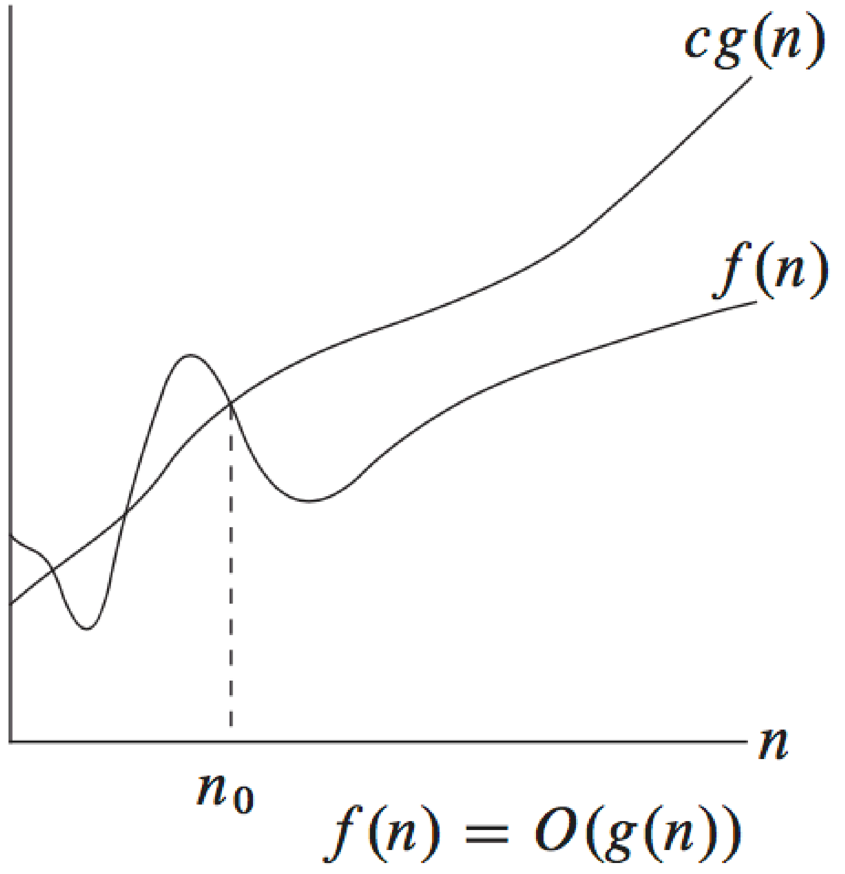
Further simplify: drop constants and lower order terms

How the running time of the algorithm increases with input size **in the limit** (*this is what asymptotic means*).



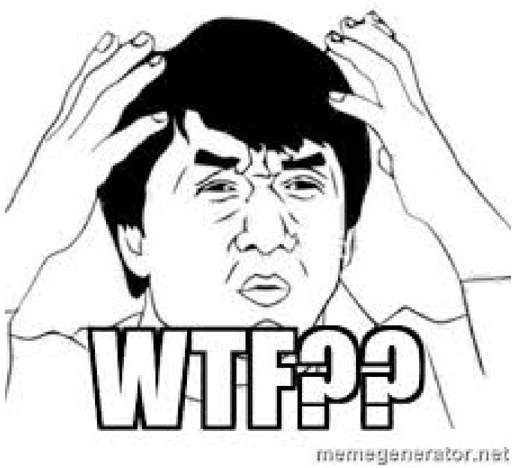
BIG-OH $O(g(n))$

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

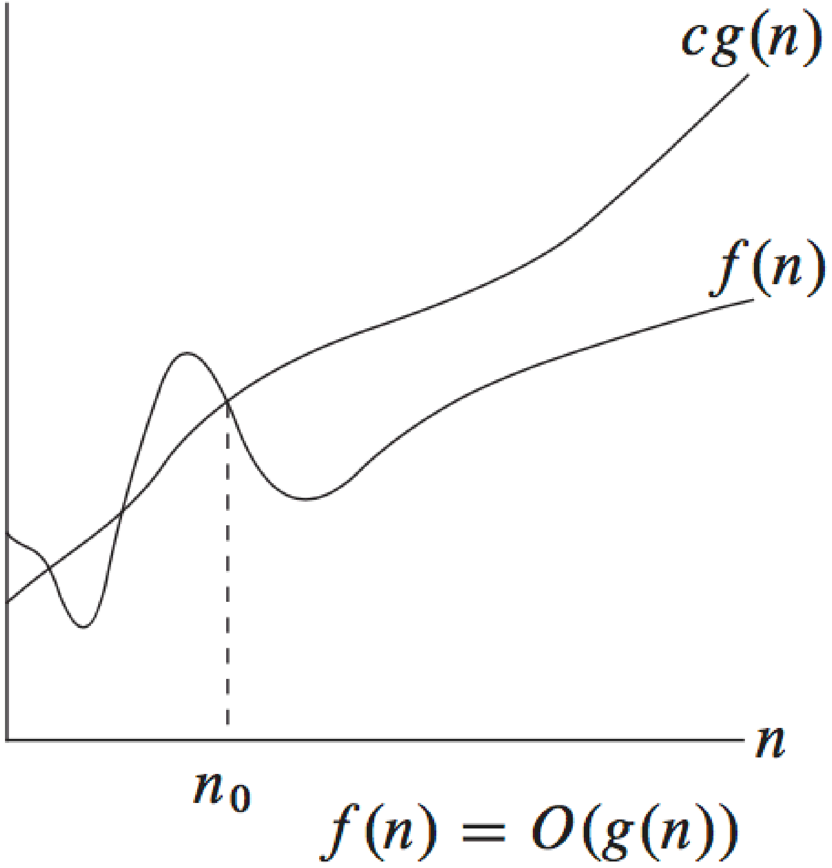


BIG-OH $O(g(n))$

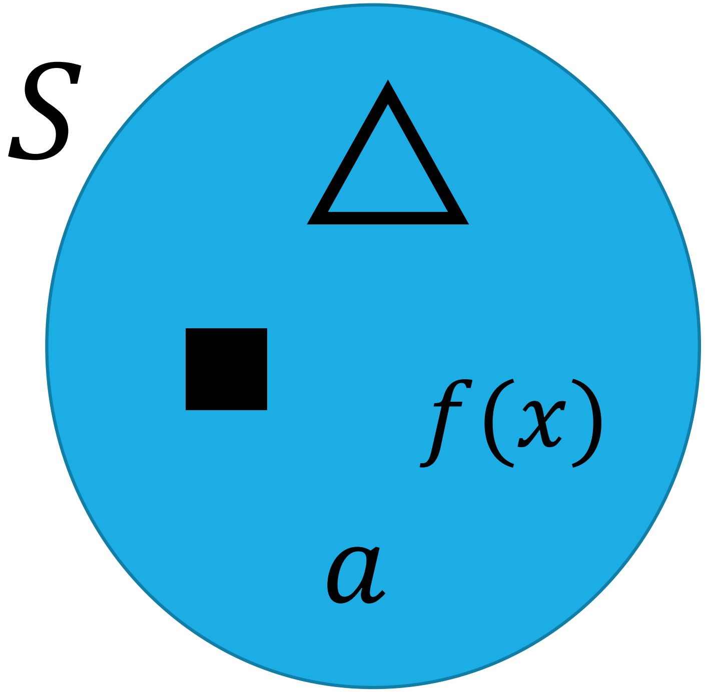
$O(g(n)) = \{f(n)$: there exist positive constants c and n_0 such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0\}$



what does this mean?!



BRIEFLY: SET THEORY



$$S = \{a, \blacksquare, \triangle, f(x)\}$$

We say an element is a member of a set using the notation:

$$a \in S$$

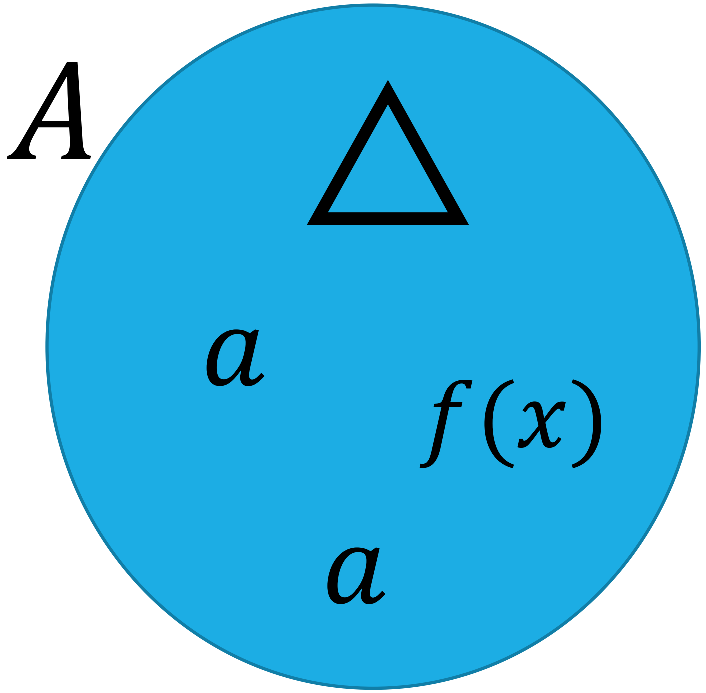
and if b is **not** a member:

$$b \notin S$$

Poll Everywhere

https://bit.ly/2LvG9bq

BRIEFLY: SET THEORY



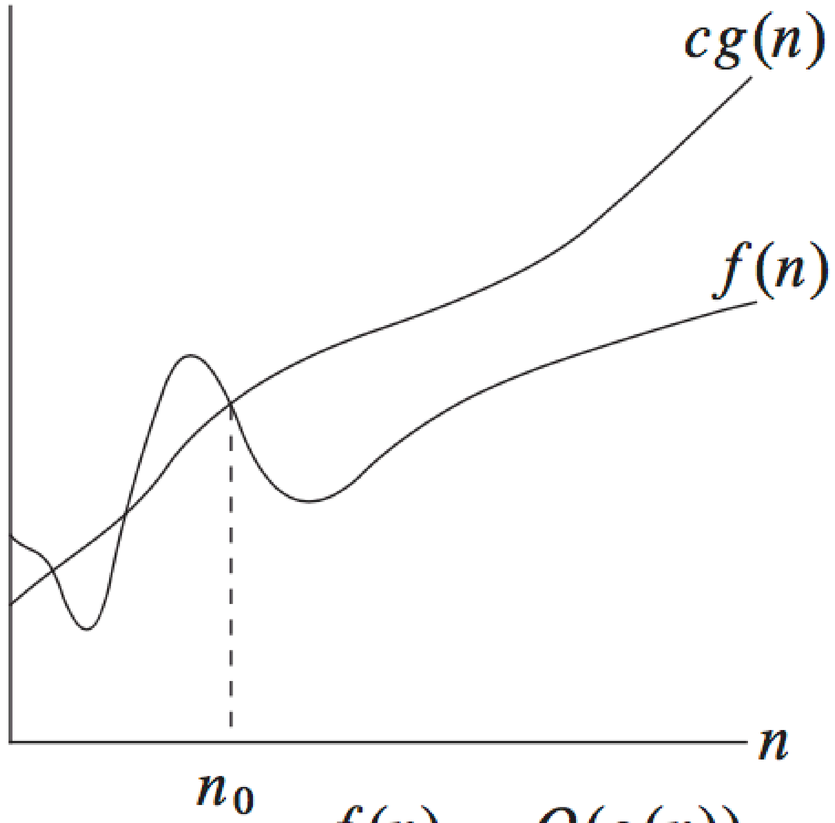
$$\mathrm{A} = \{a, a, \triangle, f(x)\}$$

Is the object on the left a valid set?

A. Yes
B. No
C. I don't know
D. The triangle is pretty
E. I'm so so confused...

BIG-OH $O(g(n))$

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



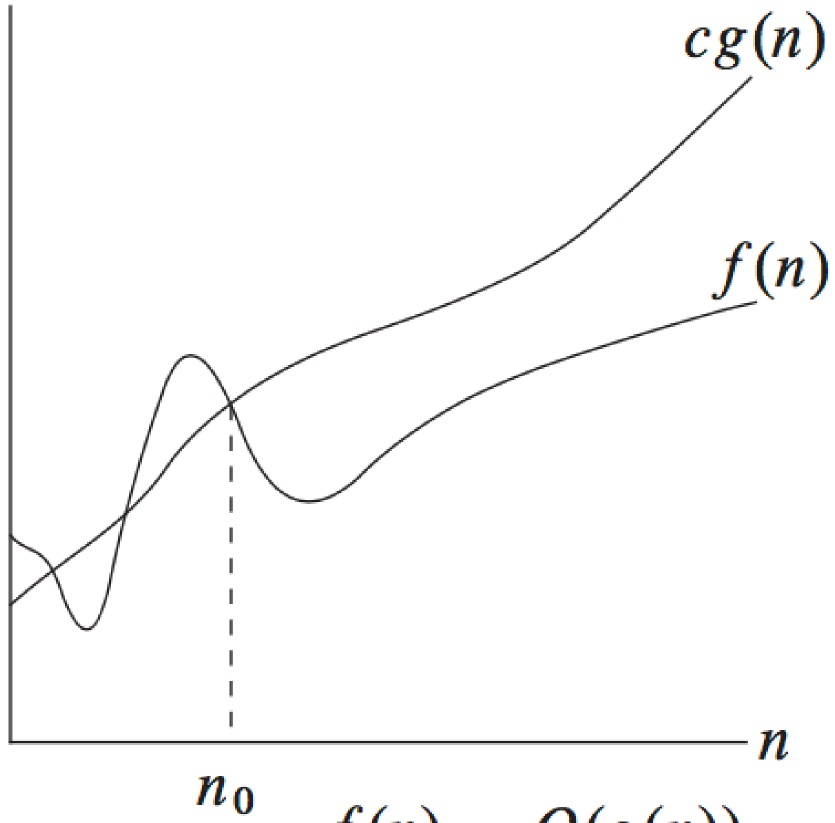
$f(n) = O(g(n))$

BIG-OH $O(g(n))$

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

Parsing the statement:

$O(g(n))$ is a ___SET___ that contains ______________ that are smaller/larger than $cg(n)$ for large n



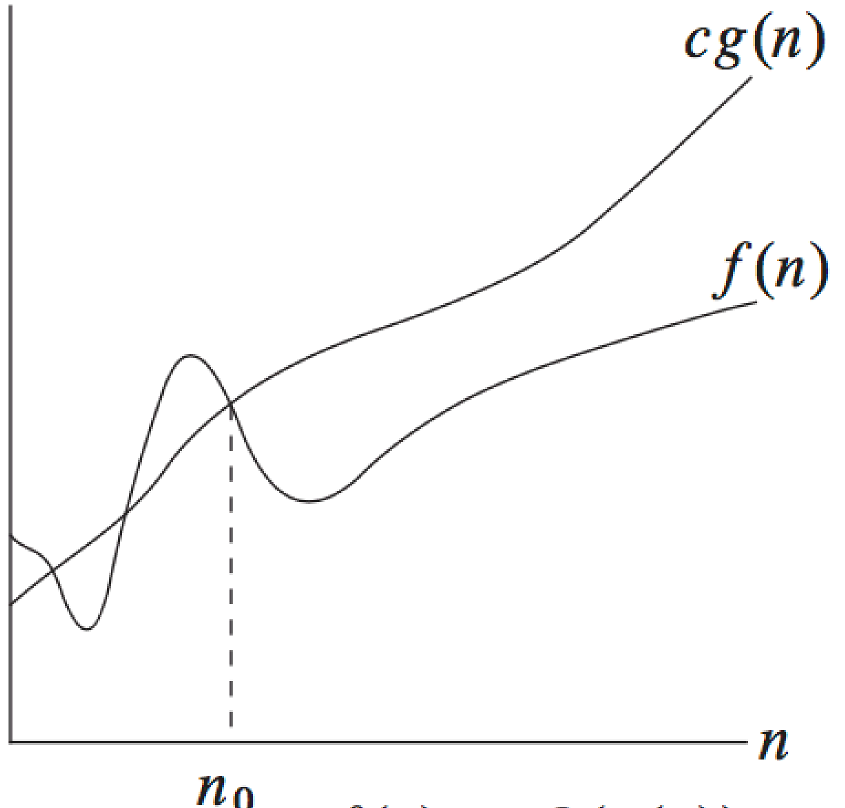
$f(n) = O(g(n))$

BIG-OH $O(g(n))$

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

Parsing the statement:

$O(g(n))$ is a __SET__ that contains __NON-NEGATIVE FUNCTIONS__ that are smaller/larger than $cg(n)$ for large n



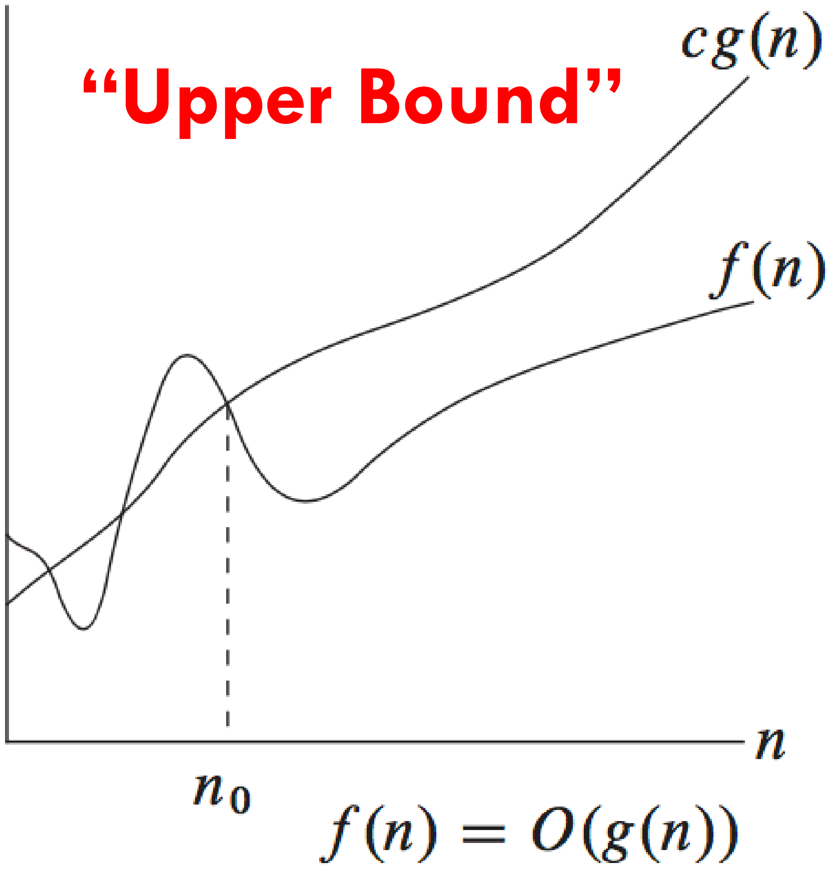
$f(n) = O(g(n))$

BIG-OH $O(g(n))$

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

Parsing the statement:

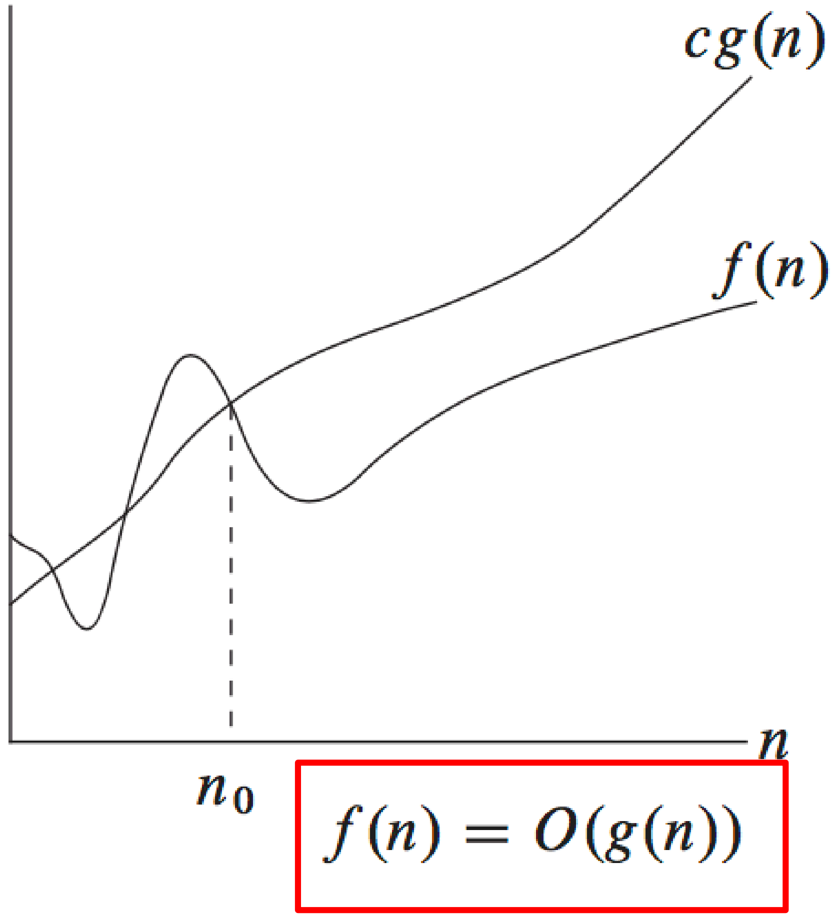
$O(g(n))$ is a __SET__ that contains __NON-NEGATIVE FUNCTIONS__ that are smaller/~~larger~~ than $cg(n)$ for large n



BUT WAIT...

If $O(g(n))$ is a set (a collection of things), how can we say that

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

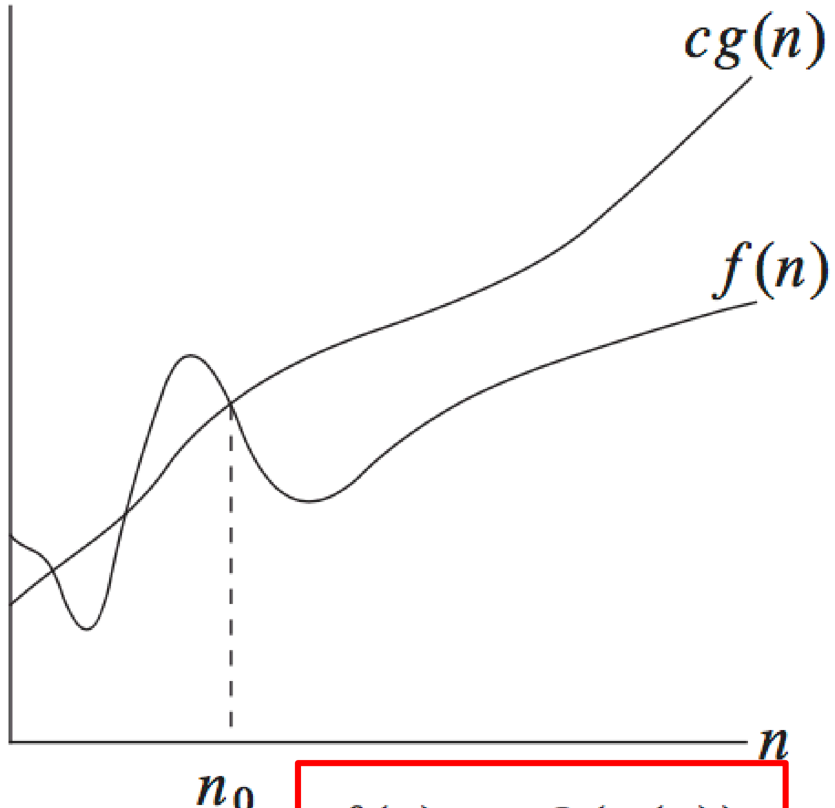


BUT WAIT…

If $O\big(g(n)\big)$ is a set (a collection of things), how can we say that

$f(n) = O\big(g(n)\big)$?

"Abuse of notation": we write $f(n) = O\big(g(n)\big)$, but mean $f(n) \in O\big(g(n)\big)$

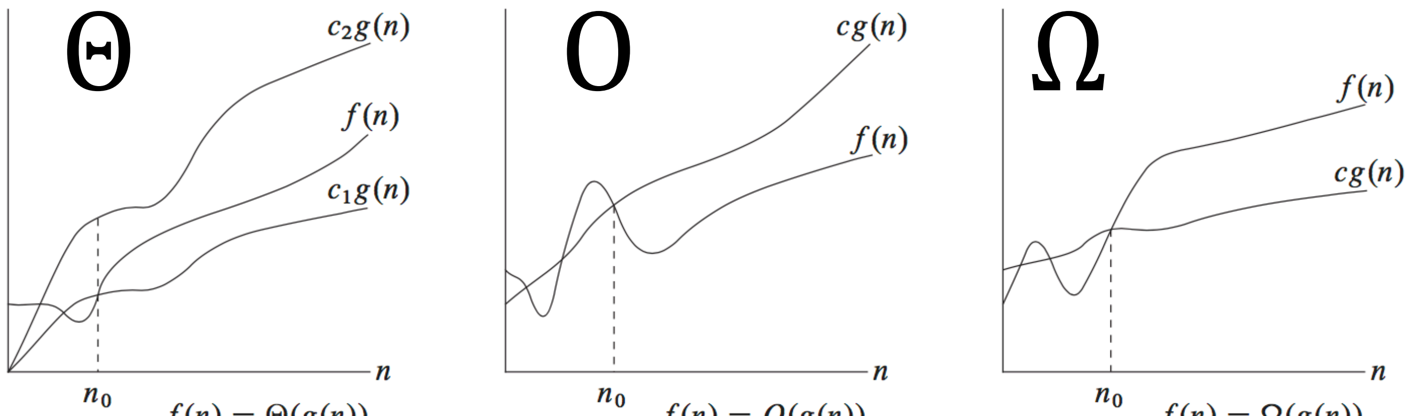


$\boxed{f(n) = O(g(n))}$

ASYMPTOTIC EFFICIENCY & ORDER OF GROWTH

Further simplify: drop constants and lower order terms

How the running time of the algorithm increases with input size **in the limit** (*this is what asymptotic means*).



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

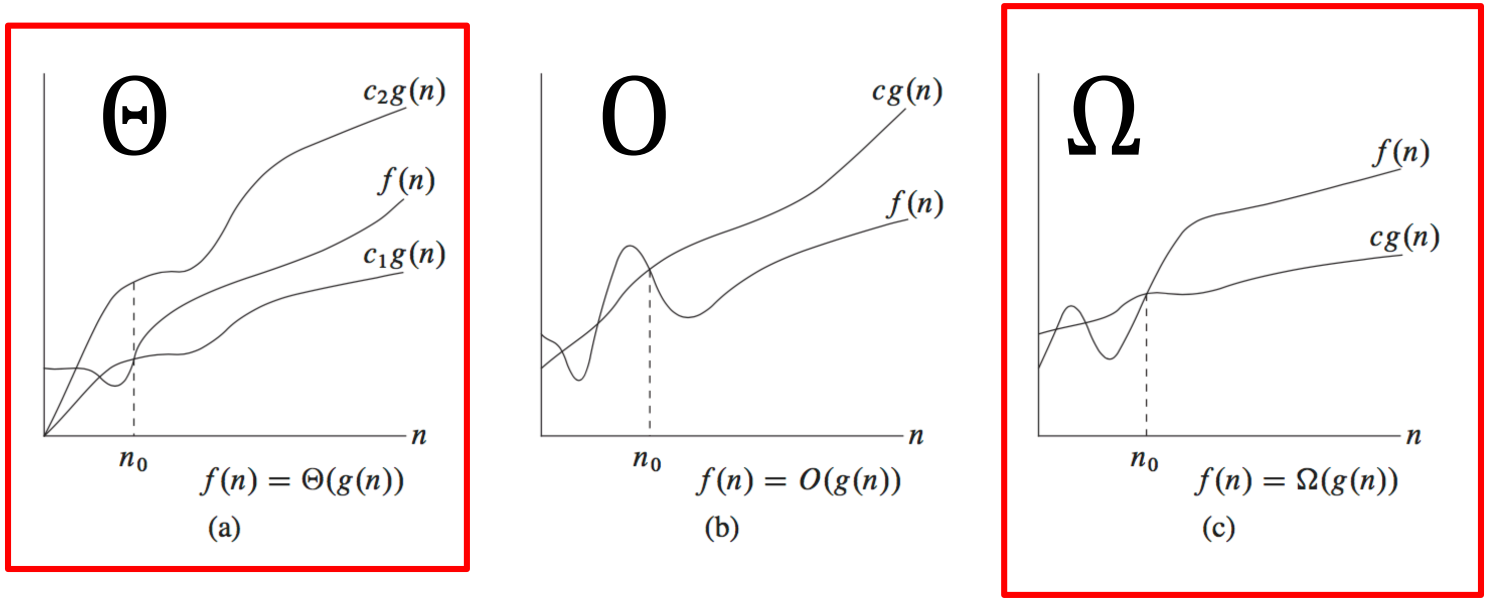
$f(n) = O(g(n))$
(b)

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

BIG OMEGA Ω AND BIG THETA Θ

What about Ω and Θ ?

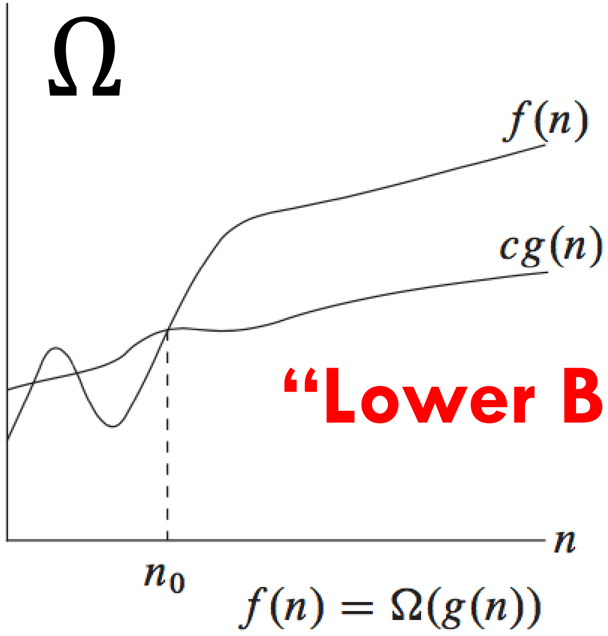
Let's take a look…



BIG OMEGA Ω

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

$O\big(g(n)\big)$ is a set that contains functions that are larger than $cg(n)$ for large n and some constant c



(c)

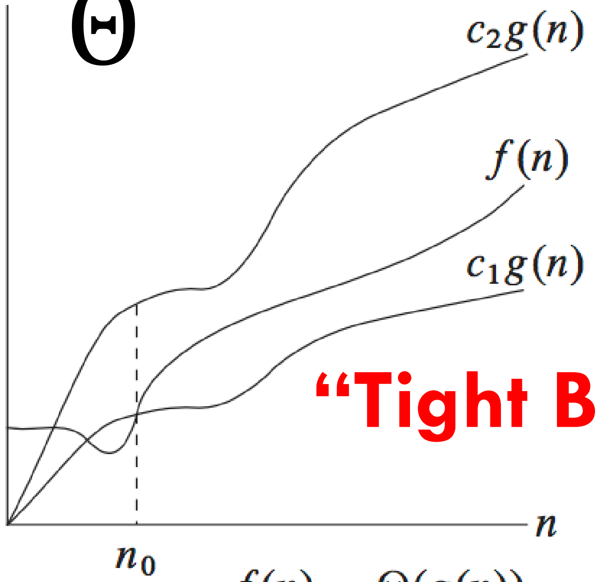
"Lower Bound"

BIG THETA Θ

$\Theta(g(n)) = \{f(n)$: there exist positive constants c_1,c_2 and $n_0$such that $0 \ \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0\}$

$O(g(n))$ is a set that contains functions that are larger than $c_1 g(n)$ and smaller than $c_2 g(n)$ for large n and some constants c_1 and c_2

Θ



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

(a)

"Tight Bound"

Poll Everywhere

https://bit.ly/2LvG9bq

WE FOCUS ON BIG-OH $O(g(n))$

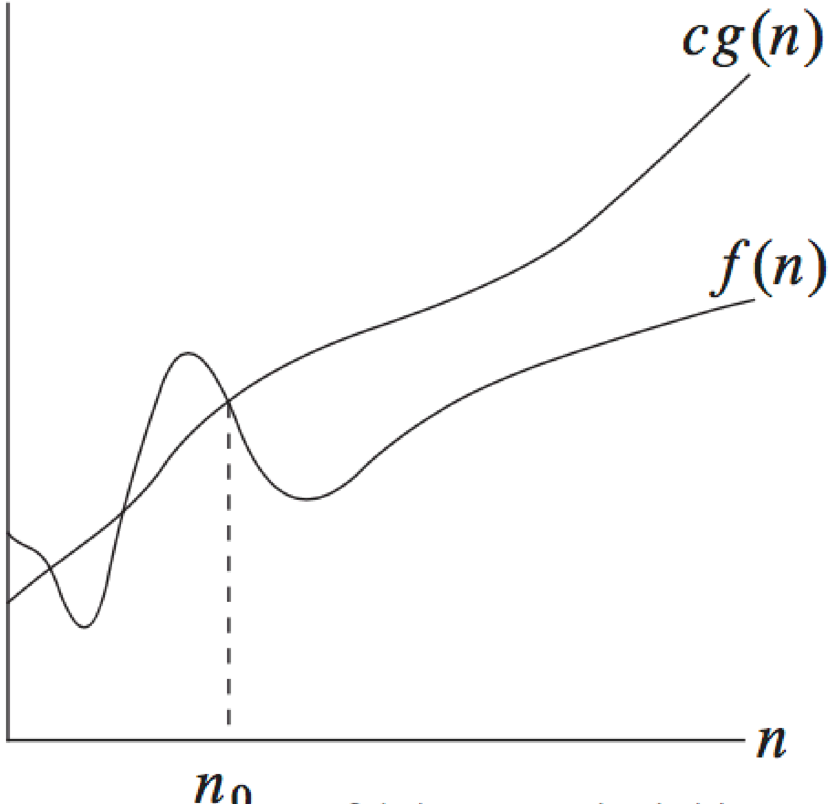
Why do we focus on $O(g(n))$?

A. we want to know the best case.

B. we want to know the worst case.

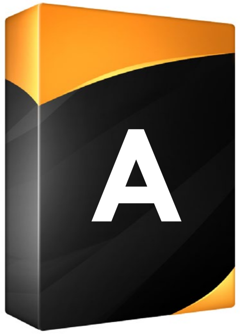
C. We are pessimistic people.

D. I'm so very confused...



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

SOFTWARE A V.S. SOFTWARE B



Guarantees in the **best case**,
it will crash only once a week

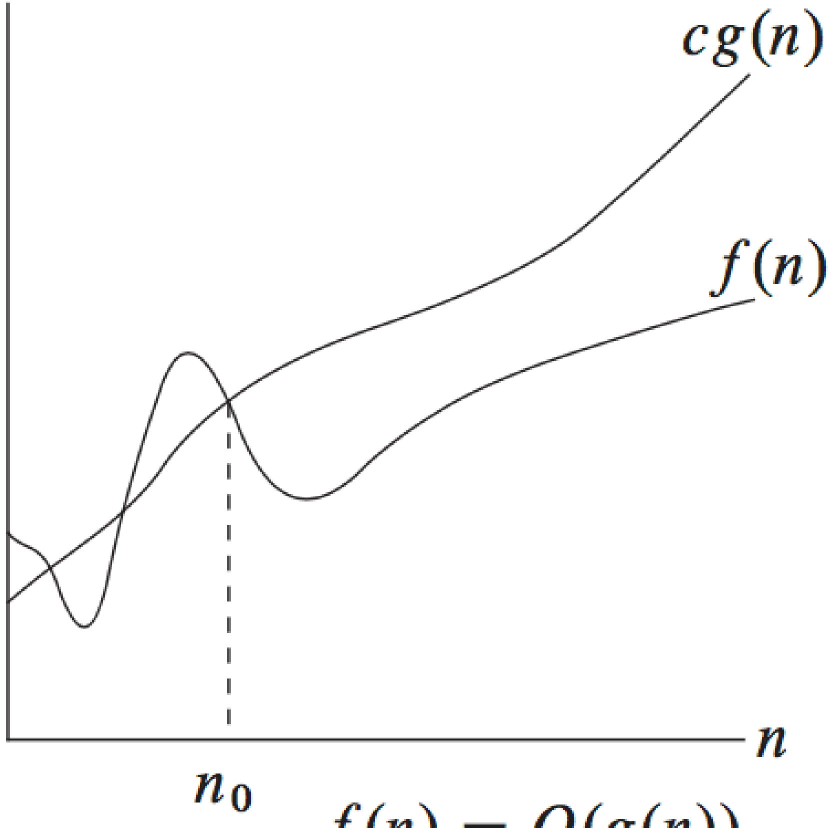
Guarantees in the **worst case**,
it will crash only once a week

Which do you buy?

WHAT DOES BIG-OH MEAN INTUITIVELY

When we say an algorithm is $O(n^2)$:

- the **worst case running time** is $O(n^2)$.
- guarantee that the running time will not exceed cn^2 for large n
- the running time is upper bounded by a cn^2 for large n



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

DROPPING THE LOWER ORDER TERMS?

Given $f(n) = an^2 + bn$ where $a, b > 0$.

Show that $f(n) = O(n^2)$, i.e., that we can drop the lower order terms and ignore the coefficient for the function.

Poll Everywhere

https://bit.ly/2LvG9bq

SOME PRACTICE WITH BIG-OH

Is $2^{n+1} = O(2^n)$?

A. Yes
B. No
C. It depends.
D. My dog ate my math homework.

Poll Everywhere

https://bit.ly/2LvG9bq

SOME PRACTICE WITH BIG-OH

What is the worst case running time for the statement x = 1?

A. $O(n)$
B. $O(n^2)$
C. $O(1)$
D. Should have stayed at home...
E. ... and watched Netflix

Poll Everywhere

https://bit.ly/2LvG9bq

SOME PRACTICE WITH BIG-OH: IF/ELSE

```
if (n > 100) {
    for (int i=0; i<n; i++) {
        for (int j=0; j<n; j++) {
            result += 2*i + 3*j + i*j;
        }
    }
    return result;
} else {
    for (int i=0; i<n; i++) {
        result += 2*i + i*i;
    }
    return result;
}
```

The algorithm on the left is:

A. $O(n)$
B. $O(n^2)$
C. $O(n^3)$
D. I'm confused!

Poll Everywhere

https://bit.ly/2LvG9bq

SOME PRACTICE WITH BIG-OH: RECURSION

```
int addSum(int n) {
    if (n<=1) return n;
    if (isOdd(n)) return n;

    return addSum(n-2) + 10;
}
```

What is the worst case running time for addSum(n)?

A. $O(n)$

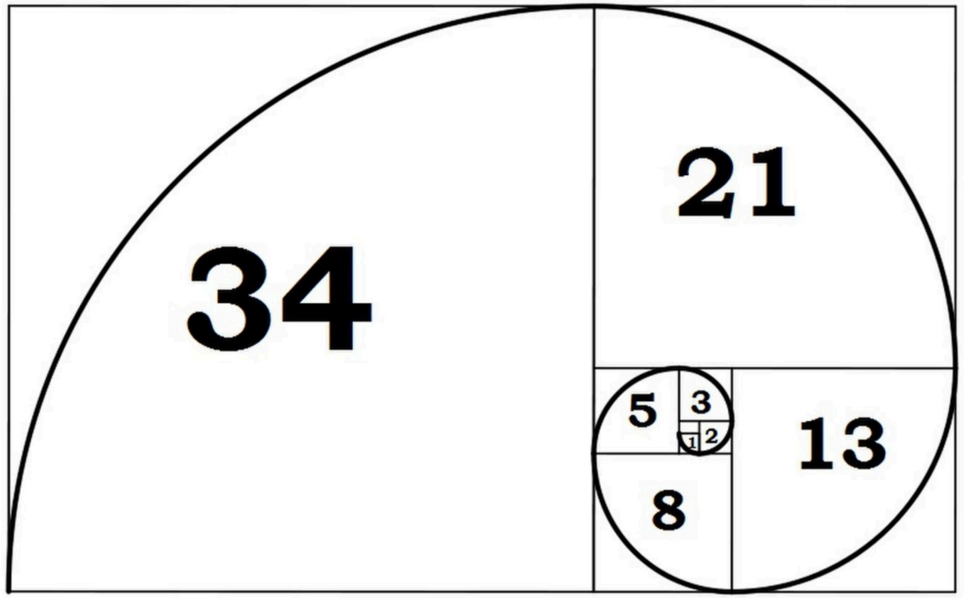
B. $O(n^3)$

C. $O(2^n)$

D. 

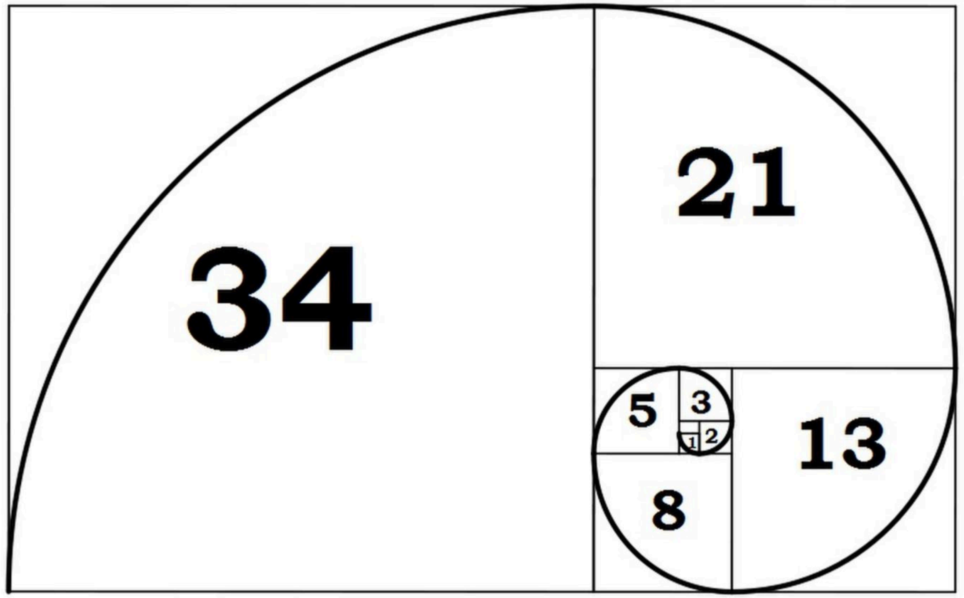
THE SHAPES OF SPIRAL GALAXIES AND HURRICANES FOLLOW THIS SEQUENCE. WHAT IS IT?

0, 1, 2, 3, 5, 8, 13, 21, 34, …



FIBONACCI SEQUENCE

0, 1, 2, 3, 5, 8, 13, 21, 34, …



Poll Everywhere

https://bit.ly/2LvG9bq

SOME PRACTICE WITH BIG-OH: ITERATION

```
static int fibItr(int n)
{
    if (n == 0) return 0;
    if (n == 1) return 1;
    int prev_2 = 0;
    int prev_1 = 1;
    int result = 0;
    for (int i = 2; i <= n; i++)
    {
        result = prev_1 + prev_2;
        prev_2 = prev_1;
        prev_1 = result;
    }
    return result;
}
```

What is the worst case running time for fibItr(n)?

A. $O(n)$

B. $O(n^3)$

C. $O(2^n)$

D. 

Poll Everywhere

https://bit.ly/2LvG9bq

SOME PRACTICE WITH BIG-OH: LINKED LISTS

What is the worst case running time for inserting an object into a linked list?

A. $O(1)$
B. $O(n)$
C. $O(n^2)$
D. how to remember? that was yesterday!

SOME PRACTICE WITH BIG-OH: STACKS

Poll Everywhere

https://bit.ly/2LvG9bq

What is the worst case running time for pushing an item onto a stack (implemented as an array)?

A. $O(1)$
B. $O(n)$
C. $O(n^2)$
D. Longer than it takes to read this question.

Poll Everywhere

https://bit.ly/2LvG9bq

SOME PRACTICE WITH BIG-OH: LIMITATIONS

Both Selection Sort and Bubble Sort are $O(N^2)$ algorithms. Does this mean they are equivalent in terms of performance?

A. Yes
B. No
C. I don't know
D. Why so many questions today ???

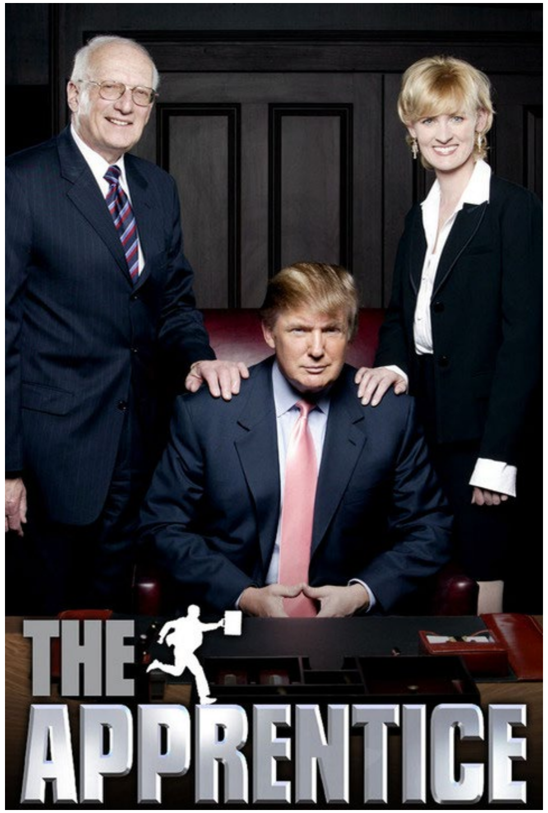
BIG-OH LIMITATIONS

is a useful but **coarse** measure.

It hides the constants and lower order terms that can make a difference.

BACK TO OUR INITIAL PROBLEM

V.S.



Poll Everywhere

https://bit.ly/2LvG9bq

NARUTO'S IDEA: INSERTION SORT

```
int n = array.length;
for (int j = 1; j < n; j++) {
    int key = array[j];
    int i = j-1;
    while ( (i > -1) && ( array [i] > key ) ) {
        array [i+1] = array [i];
        i--;
    }
    array[i+1] = key;
}
```

What is the worst case running time for insertion sort?

A. $O(n)$

B. $O(n^2)$

C. $O(2^n)$

D. Boss says $O(n!)$

Poll Everywhere

https://bit.ly/2LvG9bq

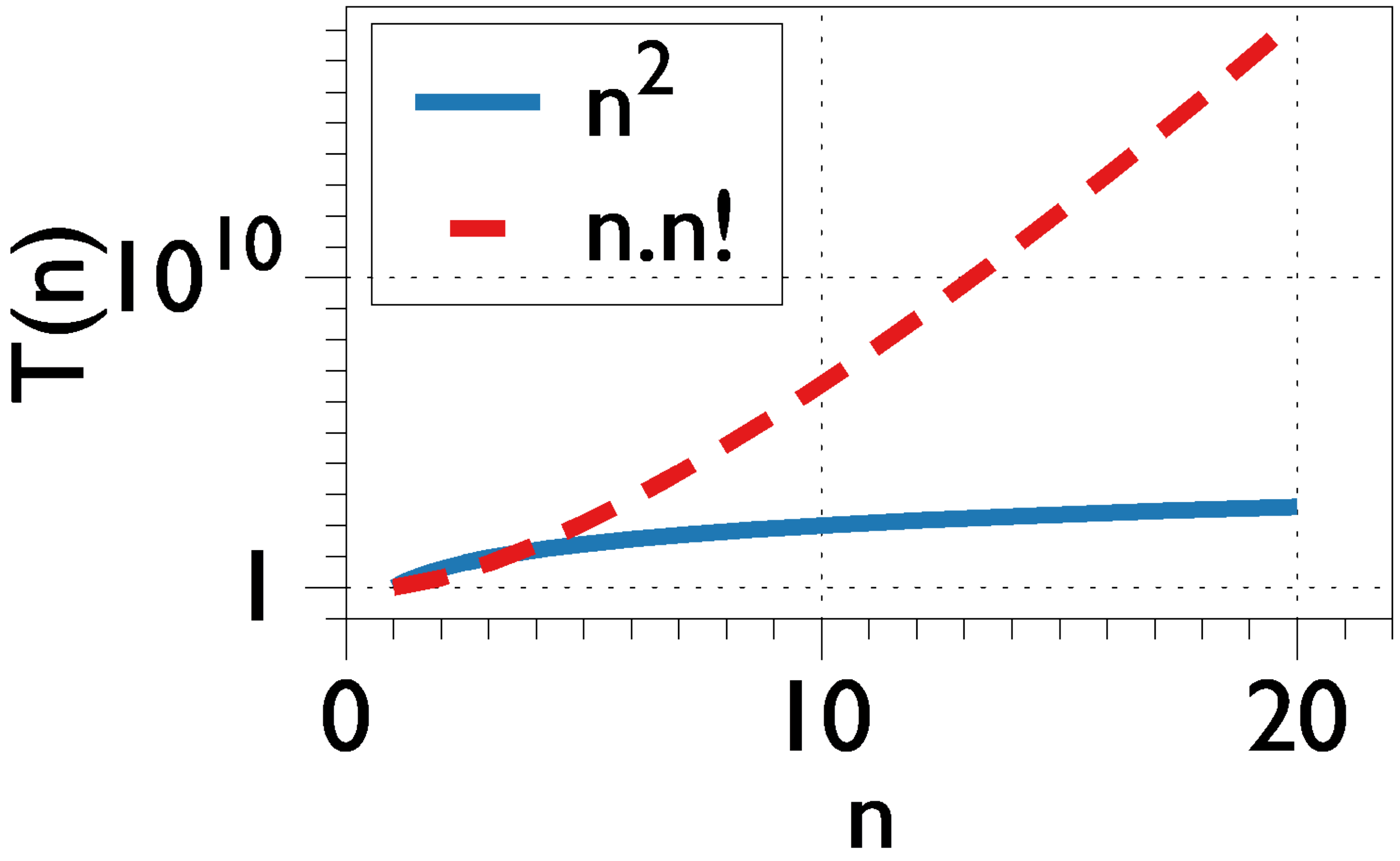
BOSS'S IDEA: BOGOSORT

```
while items is not sorted
   permute(items)
```

What is the worst case running time for bogosort?

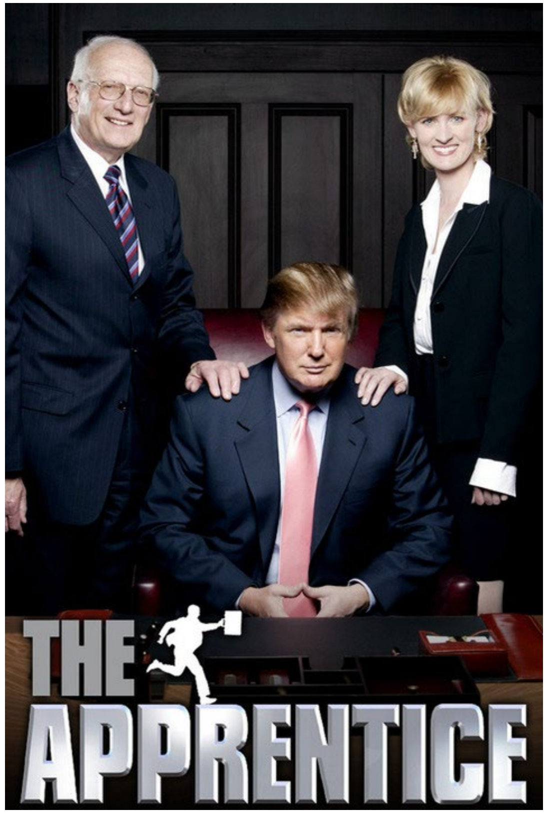
A. $O(n)$
B. $O(n^2)$
C. $O(2^n)$
D. $O(n \cdot n!)$
E. Boss says $O(1)$

THE DIFFERENCE?



WHO HAS THE BETTER IDEA?

V.S.



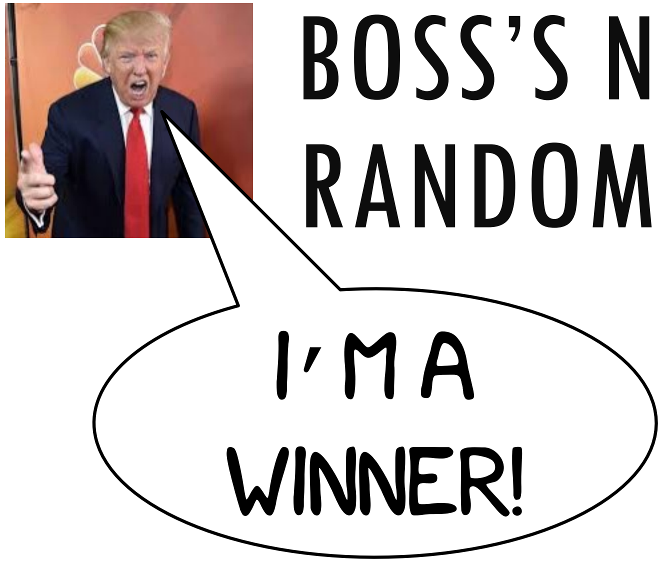
Poll Everywhere

https://bit.ly/2LvG9bq

BOSS'S NEXT IDEA: RANDOMIZED BOGOSORT


```
while items is not sorted
   randomShuffle(items)
```

BOSS'S NEXT IDEA: RANDOMIZED BOGOSORT



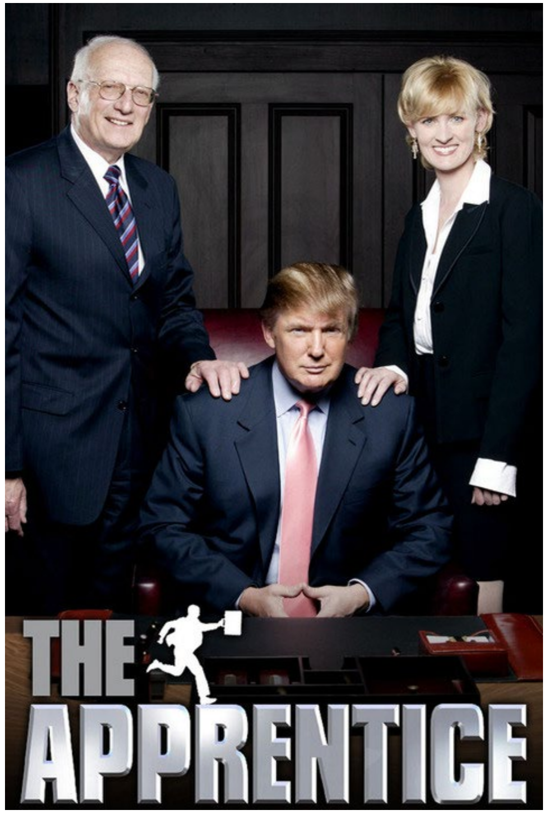

```
while items is not sorted
    randomShuffle(items)
```

What is the worst case running time for randomized bogosort?

A. $O(n)$
B. $O(n^2)$
C. $O(2^n)$
D. Boss says $O(1)$
E. Unbounded

WHO HAS THE BETTER IDEA?

V.S.



QUESTIONS?

Poll Everywhere

https://bit.ly/2LvG9bq

BOSS'S NEXT IDEA: RANDOMIZED BOGOSORT


```
while items is not sorted
    randomShuffle(items)
```

What is the **average** case running time for randomized bogosort?

A. $O(n)$
B. $O(2^n)$
C. $O(n \cdot n!)$
D. Boss still says $O(1)$
E. Unbounded

International Conference on Fun with Algorithms
FUN 2007: Fun with Algorithms pp 183-197 | Cite as

Sorting the Slow Way: An Analysis of Perversely Awful Randomized Sorting Algorithms

Hermann Gruber[1], Markus Holzer[2], and Oliver Ruepp[2]

[1] Institut für Informatik, Ludwig-Maximilians-Universität München,
Oettingenstraße 67, D-80538 München, Germany
gruberh@tcs.ifi.lmu.de

[2] Institut für Informatik, Technische Universität München,
Boltzmannstraße 3, D-85748 Garching bei München, Germany
{holzer,ruepp}@in.tum.de

Abstract. This paper is devoted to the "Discovery of Slowness." The archetypical perversely awful algorithm bogo-sort, which is sometimes referred to as Monkey-sort, is analyzed with elementary methods. Moreover, practical experiments are performed.

1 Introduction

To our knowledge, the analysis of perversely awful algorithms can be tracked back at least to the seminal paper on pessimal algorithm design in 1984 [2]. But what's a perversely awful algorithm? In the "The New Hacker's Dictionary" [7] one finds the following entry:

> **bogo-sort:** /boh'goh-sort'/ /n./ (var. 'stupid-sort') The archetypical perversely awful algorithm (as opposed to → **bubble sort**, which is merely the generic *bad* algorithm). Bogo-sort is equivalent to repeatedly throwing a deck of cards in the air, picking them up at random, and then testing whether they are in order. It serves as a sort of canonical example of awfulness. Looking at a program and seeing a dumb algorithm, one might say "Oh, I see, this program uses bogo-sort." Compare → **bogus**, → **brute force**, → **Lasherism**.

LEARNING OUTCOMES

By the end of this session, you should be able to:

- **Determine** the **computational complexity** of an algorithm under the standard sequential computation model
- **Use Big-Oh Notation** to describe algorithm performance

OTHER TAKE AWAYS

Big-Oh is a good but coarse measurement (use it wisely)

You can use $O(g(n))$ to measure the performance of other kinds of resource use

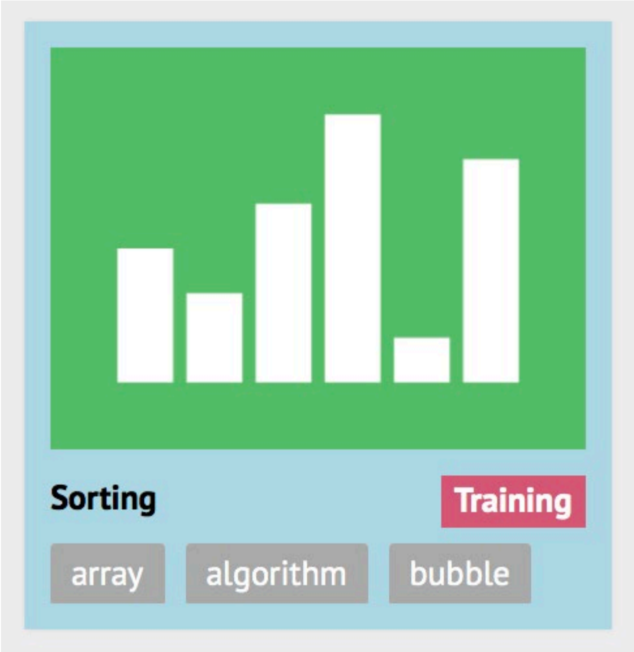
BEFORE NEXT WEEK'S LECTURE

Go to Visualgo.net and do the Sorting Module:

https://visualgo.net/en/sorting

Review: 11 (Quick Sort)

Optional: 12 onwards



QUESTIONS?