15-112
Fundamentals of Programming

Week 3 - Lecture 2:
Intro to efficiency + Big O

June 6, 2017

Principles of good programming

Correctness

Your program does what it is supposed to.

Handles all cases (e.g. invalid user input).

Maintainability

Readability, clarity of the code.

Reusability for yourself and others
(proper use of functions/methods and *objects*)

→ programs that are easy to handle and debug.

Efficiency

In terms of running time and memory space used.

The Plan

> How to properly measure running time

> Searching a given list

- Linear search
- Binary search

> Big-Oh notation

Motivating example: searching a list

Given a list of integers, and an integer, determine if the integer is in the list.

3	1	9	4	0	8	7	6	2	5

6

How many steps in the algorithm?

Motivating example: searching a list

Given a list of integers, and an integer, determine if the integer is in the list.

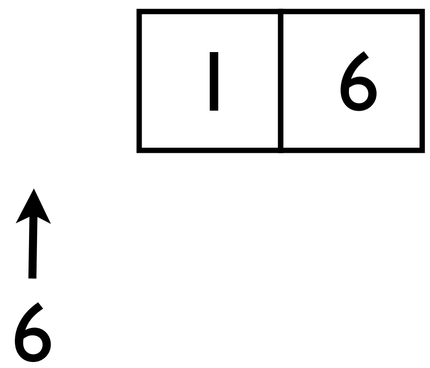
6	1	9	4	0	8	7	3	2	5

6

How many steps in the algorithm?

Motivating example: searching a list

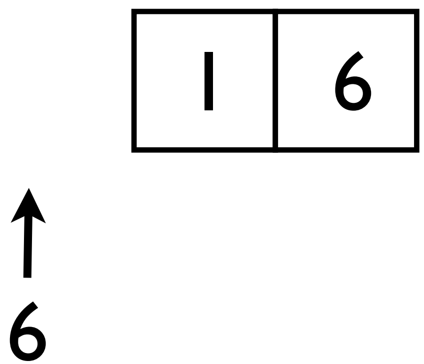
Given a list of integers, and an integer, determine if the integer is in the list.



How many steps in the algorithm?

Motivating example: searching a list

Given a list of integers, and an integer, determine if the integer is in the list.



How many steps in the algorithm?

running time of an algorithm depends on:

- size of input (e.g., size of the list)
- the values in the input

Measuring running time

running time of an algorithm depends on:

- size of input (e.g., size of the list)
- the values in the input

Measuring running time

running time of an algorithm depends on:

- size of input (e.g., size of the list)
- the values in the input

size of the list:

Want to know running time with respect to any list size.

N = list size

Measure running time as a function of N.

Measuring running time

running time of an algorithm depends on:

- size of the list (size of input)
- the values in the input

the values in the input:

Measure running time with respect to *worst input*.

worst input = input that leads to most number of steps

Measuring running time

How to properly measure running time

> Input length/size denoted by N (and sometimes by n)

- for lists: N = number of elements
- for strings: N = number of characters
- for ints: N = number of digits

> Running time is a function of N.

> Look at worst-case scenario/input of length N.

> Count algorithmic steps.

> Ignore constant factors. (e.g. $N^2 \approx 3N^2$)
(use Big-Oh notation)

The Plan

> How to properly measure running time

> Searching a given list

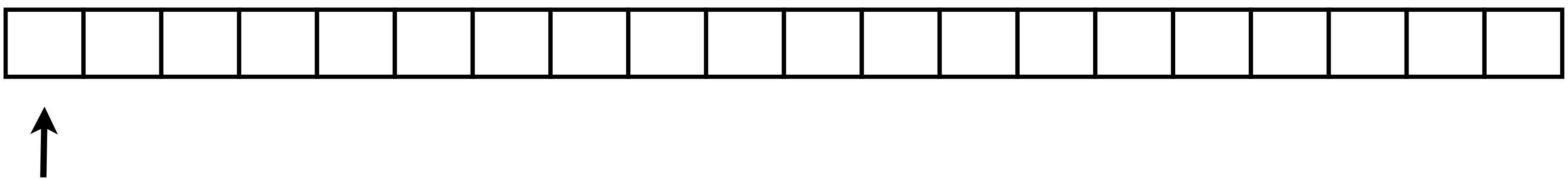
- Linear search

- Binary search

> Big-Oh notation

Searching for an element in a list

Given a list of integers, and an integer, determine if the integer is in the list.



How many steps does this take?

N steps

Can't do better (in the worst case)

This algorithm is called **Linear Search**.

Searching for an element in a sorted list

Given a sorted list of integers, and an integer, determine if the integer is in the list.

0	1	2	4	5	5	6	8	9	9	50	60	99

50

running time: N steps

Can we do better?

How would you search for a name in a phonebook?

Searching for an element in a sorted list

Binary Search

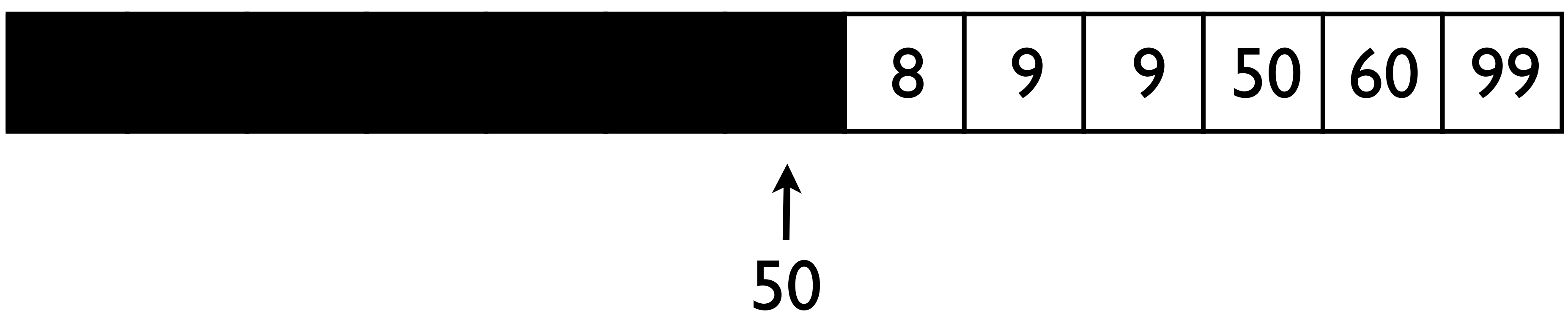
0	1	2	4	5	5	6	8	9	9	50	60	99

↑
50

Start in the middle

Searching for an element in a sorted list

Binary Search



Start in the middle

If (element > middle),

    can ignore <u>left</u> half of the list.

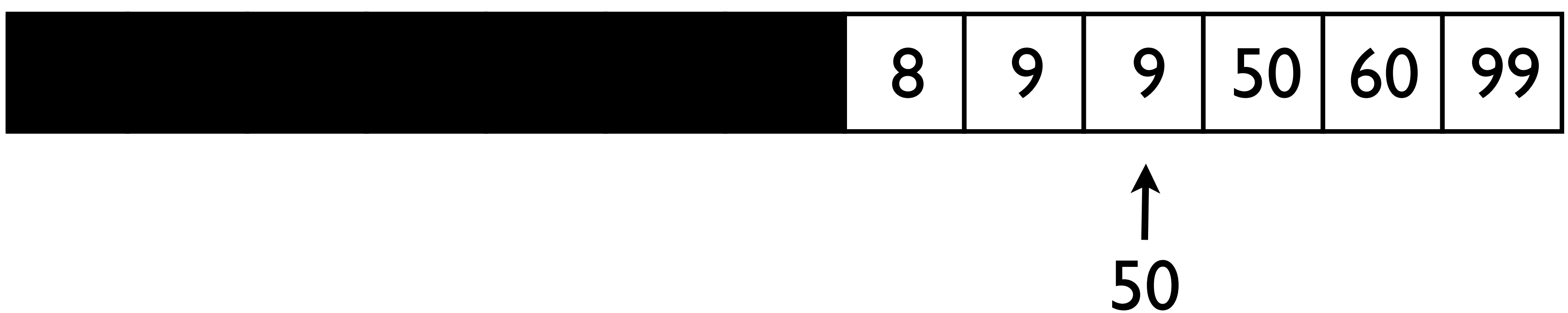
If (element < middle),

    can ignore <u>right</u> half of the list.

Repeat process on the remaining half.

Searching for an element in a sorted list

Binary Search



Start in the middle

If (element > middle),

can ignore <u>left</u> half of the list.

If (element < middle),

can ignore <u>right</u> half of the list.

Repeat process on the remaining half.

Searching for an element in a sorted list

Binary Search

Start in the middle

If (element > middle),

can ignore <u>left</u> half of the list.

If (element < middle),

can ignore <u>right</u> half of the list.

Repeat process on the remaining half.

Searching for an element in a sorted list

Binary Search

Start in the middle

If (element > middle),

can ignore <u>left</u> half of the list.

If (element < middle),

can ignore <u>right</u> half of the list.

Repeat process on the remaining half.

Searching for an element in a sorted list

Binary Search

Start in the middle

If (element > middle),

can ignore left half of the list.

If (element < middle),

can ignore right half of the list.

Repeat process on the remaining half.

Searching for an element in a sorted list

Binary Search

How many steps does this take (in the worst case)?

$\sim \log_2 N$

At each step we halve the list.

$$N \to \frac{N}{2} \to \frac{N}{4} \to \frac{N}{8} \to \cdots \to 1$$

After k steps: $\frac{N}{2^k}$ elements left. When is this 1?

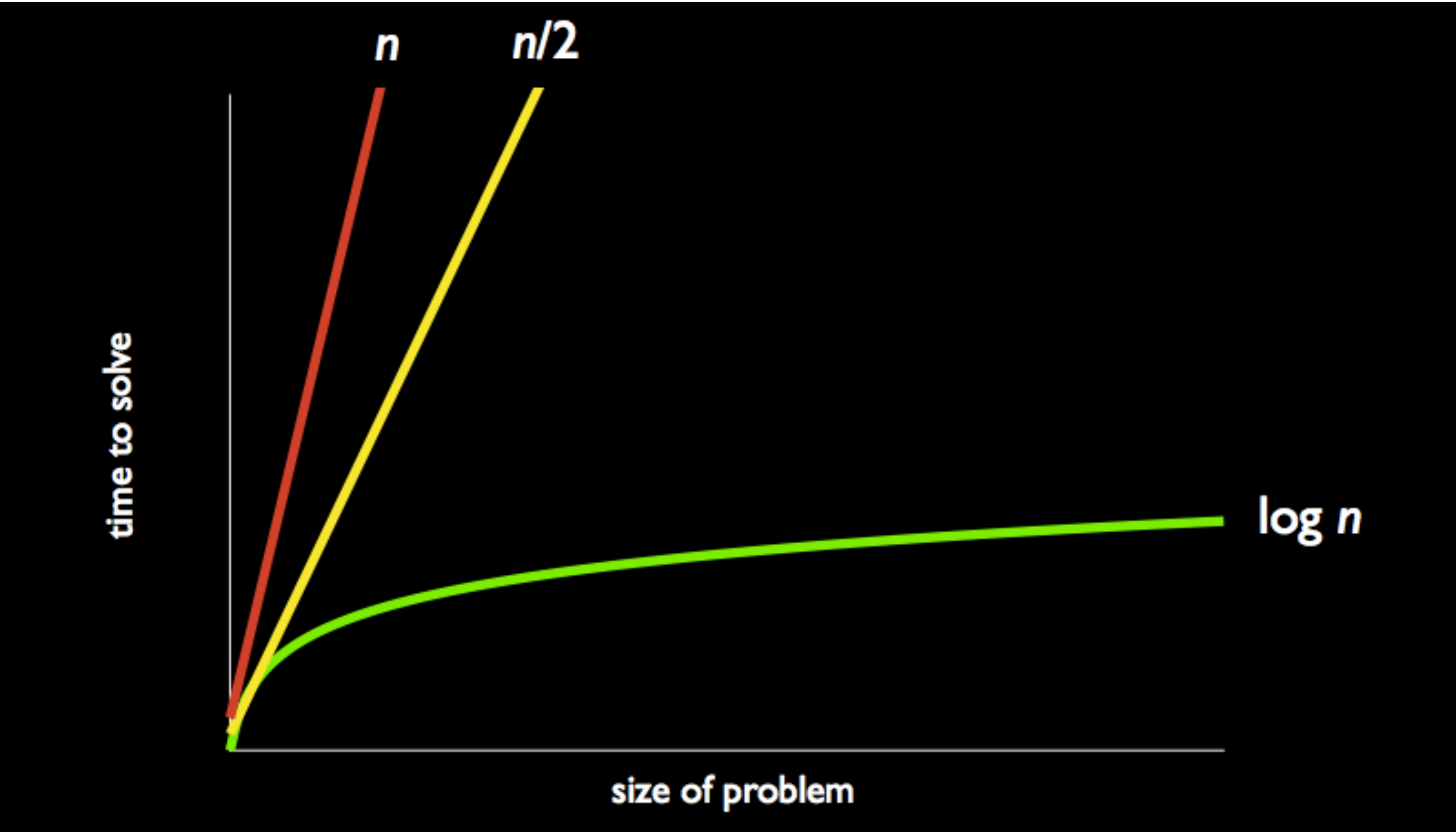
N vs log N

How much better is log N compared to N ?

N	**log N**
2	1
8	3
128	7
1024	10
1,048,576	20
1,073,741,824	30
1,152,921,504,606,846,976	60

~ 1 quintillion

n vs log n



Linear search vs Binary search

Linear Search

Takes $\sim N$ steps.

Works for both sorted and unsorted lists.

Binary Search

Takes $\sim \log_2 N$ steps.

Works for only sorted lists.

Linear search code

```
def linearSearch(L, target):
    for index in range(len(L)):
        if(L[index] == target):
            return True
    return False
```

How many steps in the worst case?

Binary search code

```
def binarySearch(L, target):
    start = 0
    end = len(L) - 1
    while(start <= end):
        middle = (start + end)//2
        if(L[middle] == target):
            return True
        elif(L[middle] > target):
            end = middle-1
        else:
            start = middle+1
    return False
```

How many steps in the worst case?

The Plan

> How to properly measure running time

> Searching a given list

- Linear search

- Binary search

> Big-Oh notation

The CS way to compare functions:

$$O(\cdot)$$

$$\leq$$

$$f(n) = O(g(n)) \qquad \equiv \qquad f(n) \text{ is } O(g(n))$$

means $f(n) \leq g(n)$, ignoring constant factors and small values of n

The CS way to compare functions:

$$O(\cdot)$$

$$\leq$$

$$10n + 25 = O(n) \qquad \equiv \qquad 10n + 25 \text{ is } O(n)$$

means $10n + 25 \leq n$, ignoring constant factors and small values of n

Big Oh Notation

A notation to ignore constant factors and small n.

$2n$ is $O(n)$	$2\log_2 n$ is $O(\log n)$
$3n$ is $O(n)$	$3\log_2 n$ is $O(\log n)$
$1000n$ is $O(n)$	$1000\log_2 n$ is $O(\log n)$
$0.0000001n$ is $O(n)$	$0.0000001\log_2 n$ is $O(\log n)$
n is $O(n^2)$	$\log_9 n$ is $O(\log n)$
$0.0000001n^2$ is <u>not</u> $O(n)$	$n\log_7 n + 100$ is <u>not</u> $O(n)$

Running time of linear search is $O(N)$

Running time of binary search is $O(\log N)$

Big Oh Notation

Why ignore constant factors and small n?

- We want to capture the essence of an algorithm/problem.

- Difference in Big Oh →

  a really fundamental difference.

Big Oh Notation

Ignoring constant factors means ignoring lower order additive terms.

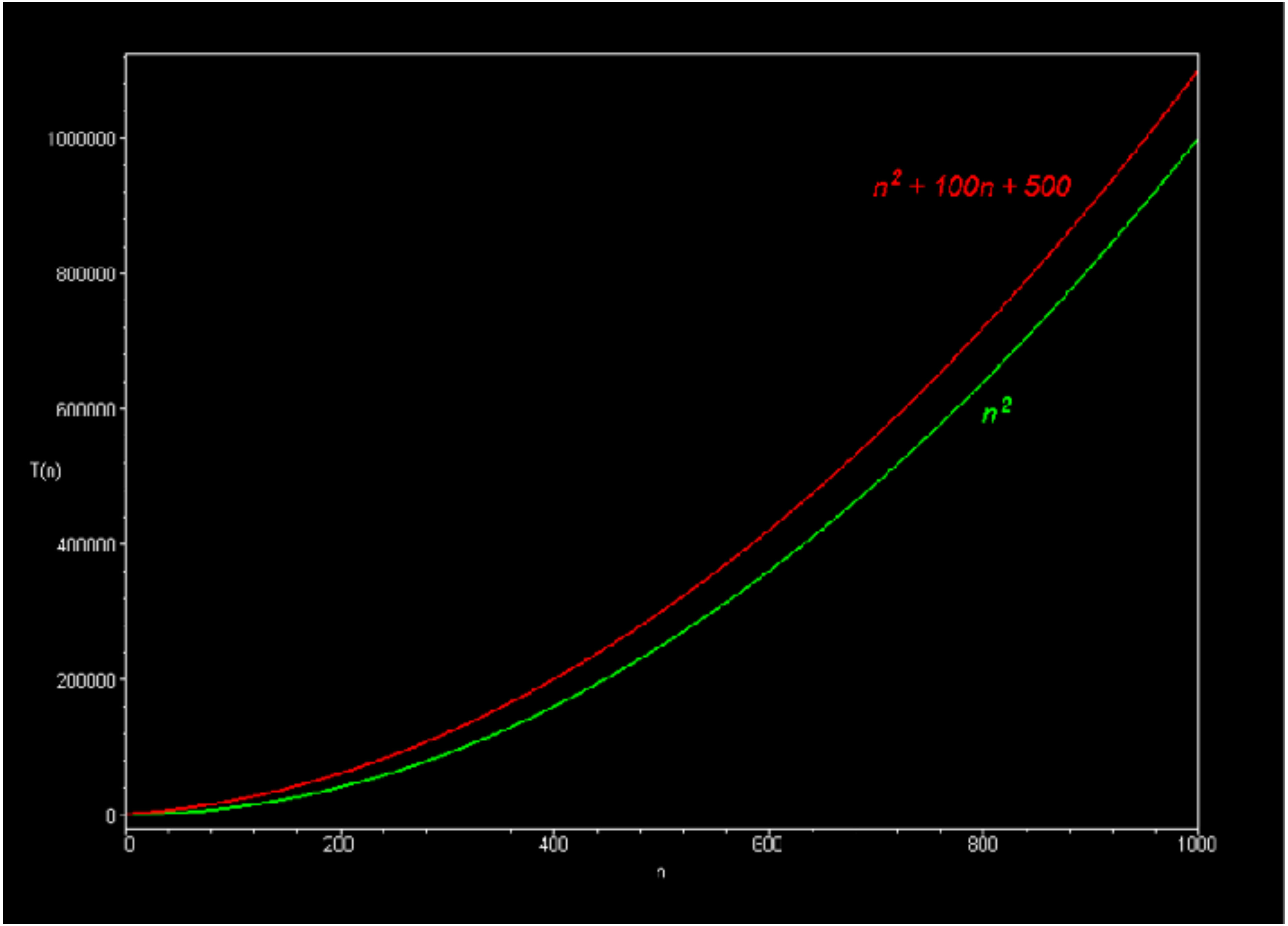
$$n^2 + 100n + 500 \text{ is } O(n^2)$$

Also:

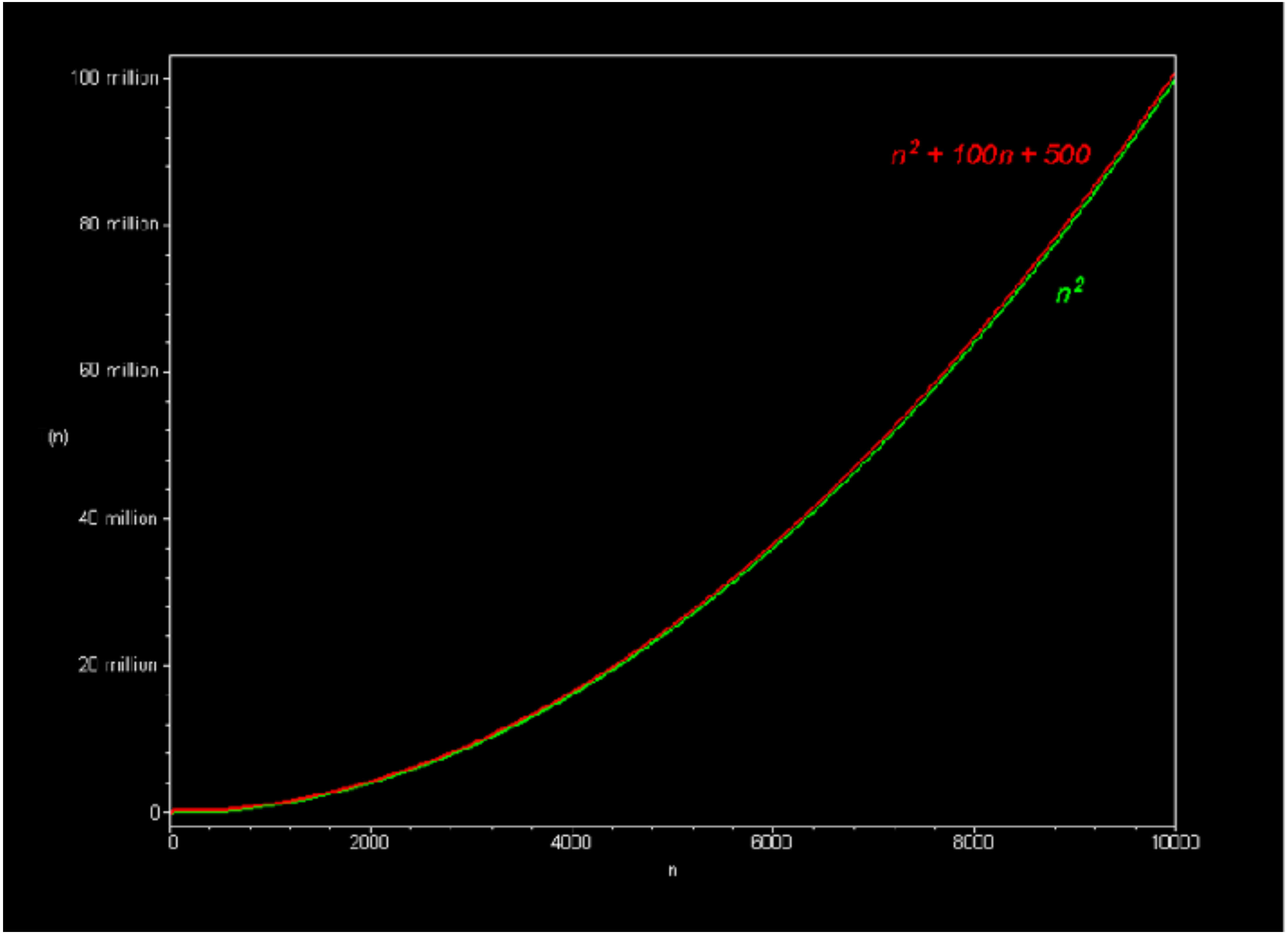
$$\frac{n^2 + 100n + 500}{n^2} = 1 + \frac{100n}{n^2} + \frac{500}{n^2} \quad \longrightarrow 1$$

Lower order terms don't matter!

Big Oh Notation



Big Oh Notation



Important Big Oh Classes

Again, not much interested in the difference between n and $n/2$.

We are **very** interested in the differences between

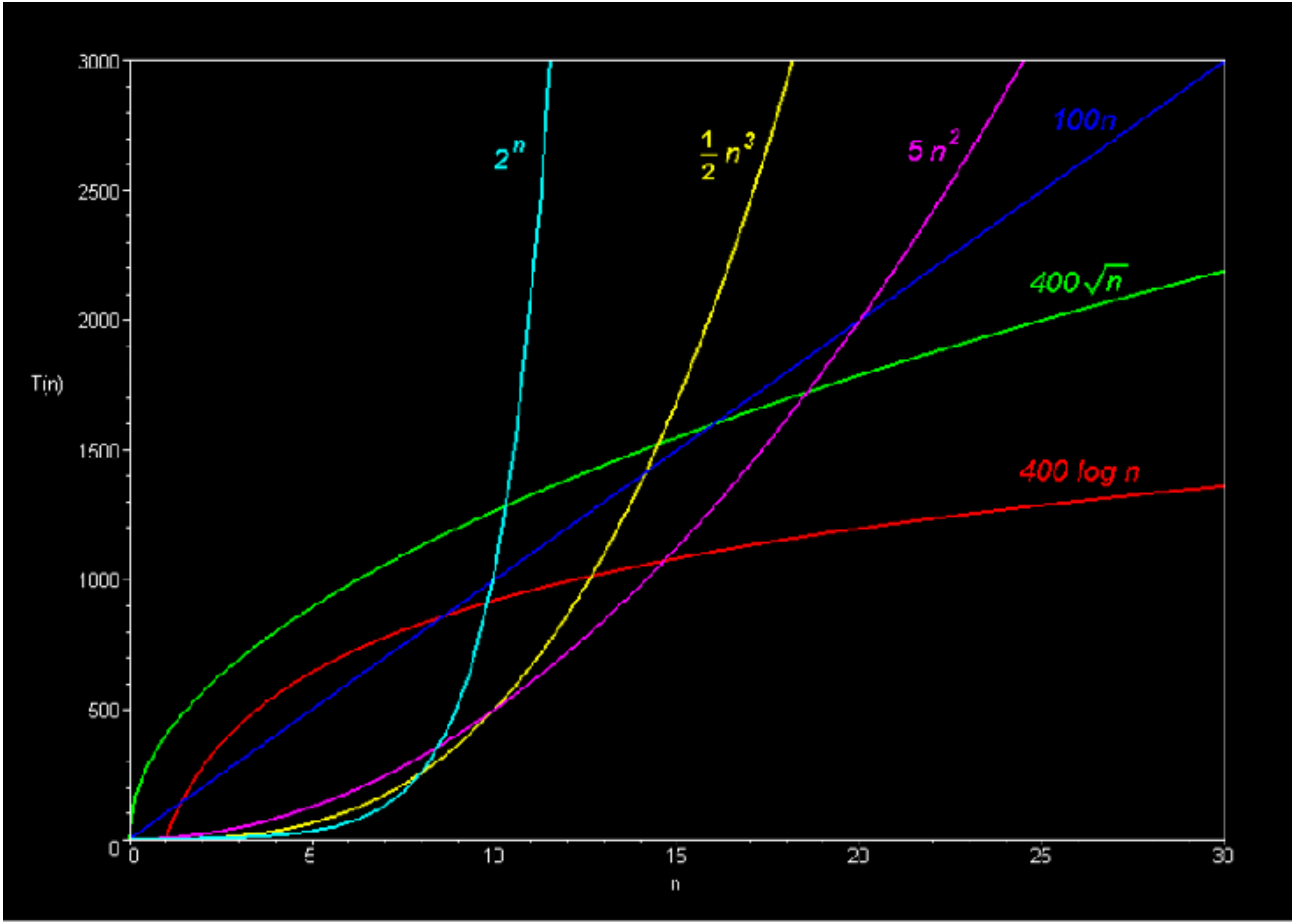
$$\log n \ <<< \ \sqrt{n} \ << \ n \ << \ n^2 \ << \ n^3 \ <<< \ 2^n$$

Important Big Oh Classes

Common function families:

Constant:	$O(1)$
Logarithmic:	$O(\log n)$
Square-root:	$O(\sqrt{n}) = O(n^{0.5})$
Linear:	$O(n)$
Loglinear:	$O(n \log n)$
Quadratic:	$O(n^2)$
Exponential:	$O(k^n)$

Important Big Oh Classes



Exponential running time

If your algorithm has exponential running time
e.g. $\sim 2^n$

No hope of being practical.

n vs 2^n

2^n	n
2	1
8	3
128	7
1024	10
1,048,576	20
1,073,741,824	30
1,152,921,504,606,846,976	60

Exponential running time example

Given a list of integers, determine if there is a subset of the integers that sum to 0.

4	-3	-2	7	99	5	1

Exponential running time example

Given a list of integers, determine if there is a subset of the integers that sum to 0.

4	-3	-2	7	99	5	1

Exhaustive Search

Try every possible subset and see if it sums to 0.

Number of subsets is 2^N

So running time is at least 2^N

Review: Measuring running time

How to properly measure running time

> Input length/size denoted by N (and sometimes by n)

- for lists: N = number of elements
- for strings: N = number of characters
- for ints: N = number of digits

> Running time is a function of N.

> Look at worst-case scenario/input of length N.

> Count algorithmic steps.

> Ignore constant factors. (e.g. $N^2 \approx 3N^2$)
(use big Oh notation)

Review

Give 2 definitions of $\log_2 N$

Number of times you need to divide N by 2 to reach 1.

The number *k* that satisfies $2^k = N$.

What is the big Oh notation used for?

Describe the performance or complexity of an algorithm by ignoring:

- constant factors
- small *N*.

ignore small order additive terms.

Review

Big-Oh is the right level of abstraction!

$$8N^2 - 3n + 84$$

is analogous to "too many significant figures".

$$O(N^2)$$

"Sweet spot"

- coarse enough to suppress details like programming language, compiler, architecture,...
- sharp enough to make comparisons between different algorithmic approaches.

Review

$10^{10} n^3$ is $O(n^3)$? Yes

n is $O(n^2)$? Yes

n^3 is $O(2^n)$? Yes

n^{10000} is $O(1.1^n)$? Yes

$100n \log_2 n$ is $O(n)$? No

$1000 \log_2 n$ is $O(\sqrt{n})$? Yes

$1000 \log_2 n$ is $O(n^{0.00000001})$? Yes

When we ask "what is the running time…" you must give the tight bound!

Does the base of the log matter?

$$\log_b n = \frac{\log_c n}{\boxed{\log_c b}}$$

constant

Review

Constant:	$O(1)$
Logarithmic:	$O(\log n)$
Square-root:	$O(\sqrt{n}) = O(n^{0.5})$
Linear:	$O(n)$
Loglinear:	$O(n \log n)$
Quadratic:	$O(n^2)$
Polynomial:	$O(n^k)$
Exponential:	$O(k^n)$

Review

$$\log n <<< \sqrt{n} << n < n \log n << n^2 << n^3 <<< 2^n <<< 3^n$$

Review

You have an algorithm with running time $O(N)$.

If we double the input size,
by what factor does the running time increase?

If we quadruple the input size,
by what factor does the running time increase?

You have an algorithm with running time $O(N^2)$.

If we double the input size,
by what factor does the running time increase?

If we quadruple the input size,
by what factor does the running time increase?