CS 350 Algorithms and Complexity

Fall 2019

Lecture 2: Preliminaries, Asymptotic Notation

Andrew P. Black

based in part on material by Mark P. Jones

Department of Computer Science

Portland State University

Form yourselves into groups of 4

- Number yourselves 1, 2, 3 and 4
- Roles:

Facilitator: gets discussion moving and keeps it moving, e.g., by asking the other group members questions, sometimes about what they've just been saying.

Summarizer: Every so often, provides a summary of the discussion for other students to approve or amend.

Reflector: This student will listen to what others say and explain it back in his or her own words, asking the original speaker if the interpretation is correct.

Elaborator: This person seeks connections between the current discussion and past topics or overall course themes.

Here are your Roles:

3. Facilitator: gets discussion moving and keeps it moving, e.g., by asking the other group members questions, sometimes about what they've just been saying.

2. Summarizer: Every so often, provides a summary of the discussion for other students to approve or amend.

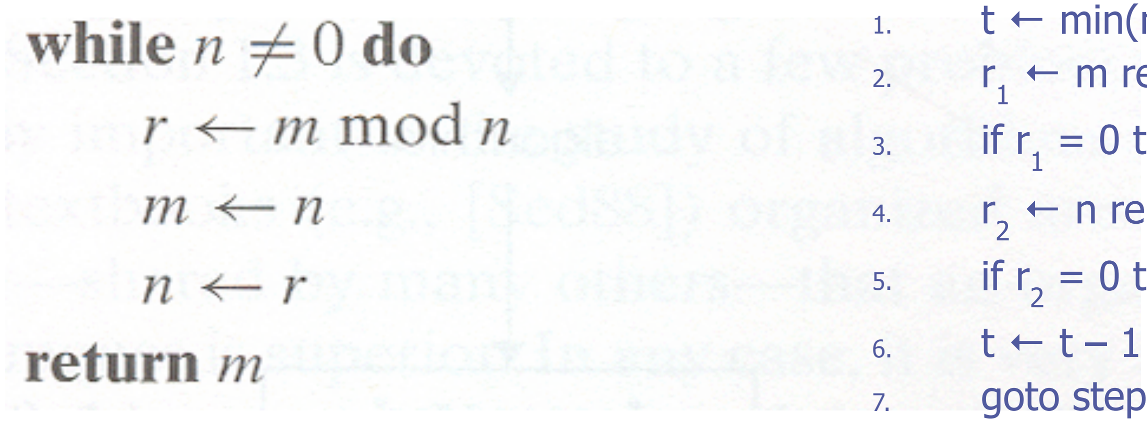
4. Reflector: This student will listen to what others say and explain it back in his or her own words, asking the original speaker if the interpretation is correct.

1. Elaborator: This person seeks connections between the current discussion and past topics or overall course themes.

GCD (again!)

- Find gcd(31415, 13205) using Euclid’s algorithm
- *Estimate* how many times faster that was, compared to using the consecutive algorithm

Euclid



```
while $n \neq 0$ do
    $r \leftarrow m \bmod n$
    $m \leftarrow n$
    $n \leftarrow r$
return $m$
```

Consecutive

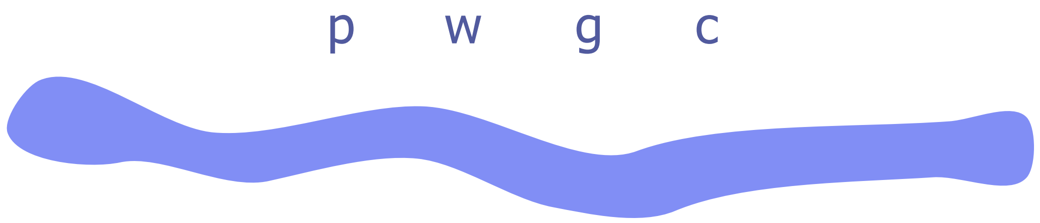
```
1. $t \leftarrow \min(m, n)$
2. $r_1 \leftarrow m$ rem $t$.
3. if $r_1 = 0$ then goto step 4 else goto step 6
4. $r_2 \leftarrow n$ rem $t$.
5. if $r_2 = 0$ then return $t$
6. $t \leftarrow t - 1$
7. goto step 2
```

Old world Puzzle

A peasant finds himself on a riverbank with a wolf, a goat, and a head of cabbage. He needs to transport all three to the other side of the river in his boat. However, the boat has room for only the peasant himself and one other item (either the wolf, the goat, or the cabbage). In his absence, the wolf would eat the goat, and the goat would eat the cabbage.

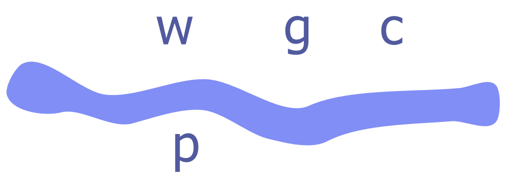
Solve this problem for the peasant or prove it has no solution. (Note: The peasant is a vegetarian but does not like cabbage and hence can eat neither the goat nor the cabbage to help him solve the problem. And it goes without saying that the wolf is a protected species.)

What’s the first move?

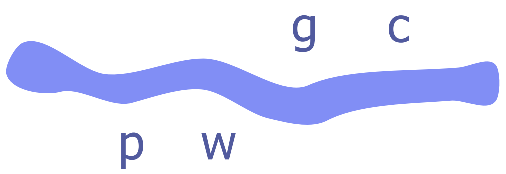


What’s the first move?

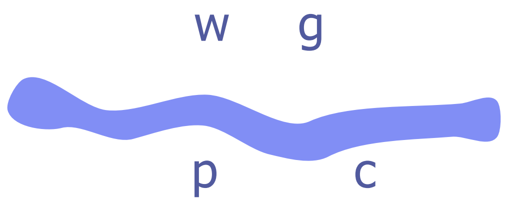
A:



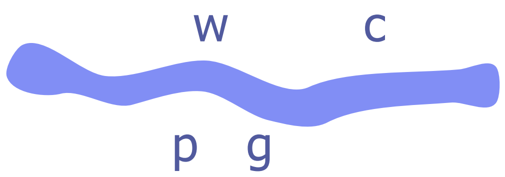
B:



C:

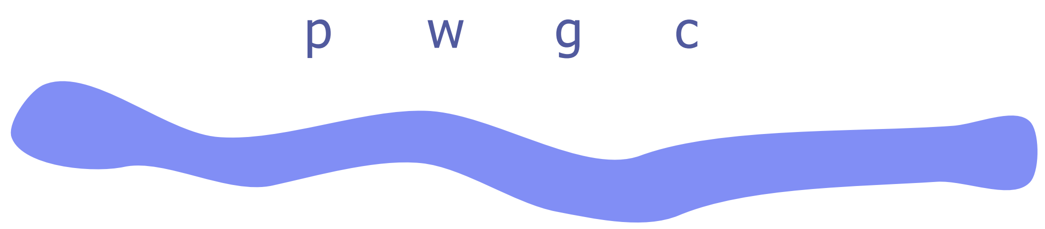


D:



E: none of the above

Solve this problem in your groups



What's an algorithm?

- Which of the following constitute an <u>algorithm</u> for computing the area of a triangle, given positive numbers a, b, c representing the lengths of the sides?

a. $S = \sqrt{p(p-a)(p-b)(p-c)}$, where $p = (a+b+c)/2$

b. $S = \frac{1}{2}bc\sin A$, where A is the angle between sides b and c

c. $S = \frac{1}{2}ah_a$, where h_a is the height to base a

Basic Data Structures

- Array: sequence of n items, stored contiguously in memory
  - element access: **constant time**
- Linked List: sequence of n nodes, each containing a pointer and an item
  - access to element k: time proportional to k
  - insert an element: **constant time**
  - delete an element: **constant time**

- How can we <u>insert</u> and <u>delete</u> at index k in an array?

- How long do these operations take?

A. constant time?

B. time proportional to insertion position k ?

C. time proportional to size of the array n ?

Group Problem—Unsorted Array

- Suppose that you have an (unsorted) array of size n.
- How can you delete the ith element ($1 \leq i \leq n$) of the array so that the time taken does not depend on n?

Group Problem—Unsorted Array

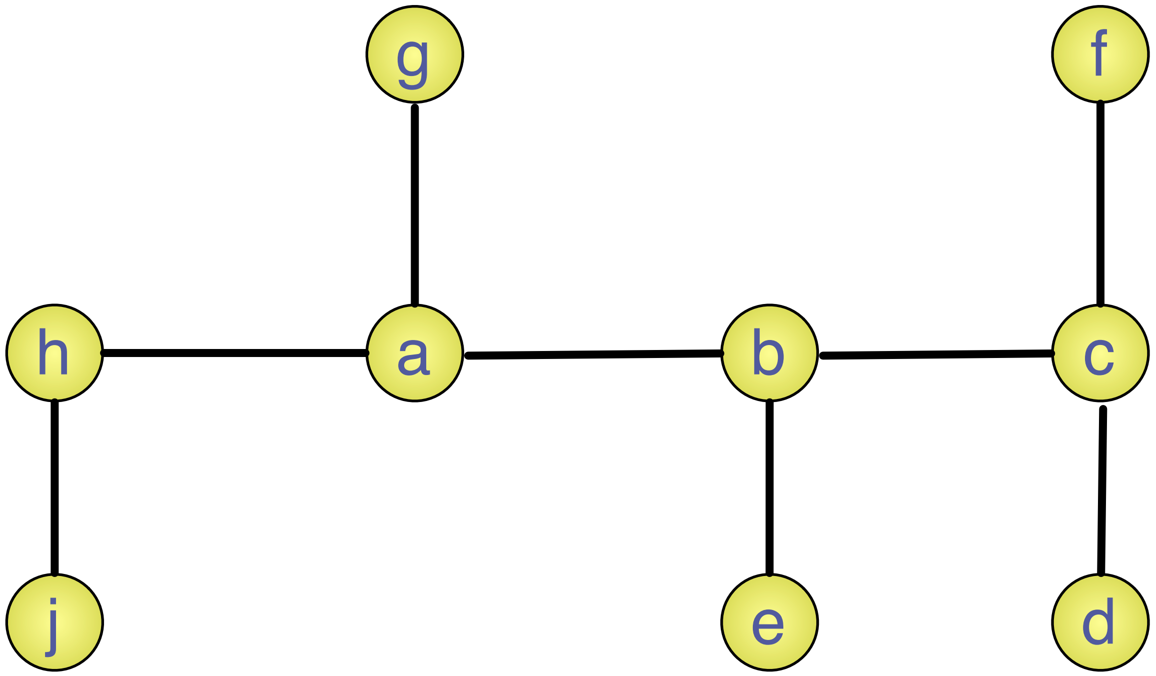
- Suppose that you have an (unsorted) array of size n.
- How can you delete the ith element ($1 \le i \le n$) of the array so that the time taken does not depend on n?
- Is element access still constant time?

Group Problem—Sorted Array

- Suppose that you have a *sorted* array of size n.
- How can you delete the ith element ($1 \le i \le n$) of the array so that the time taken does not depend on n? Yes, the array must remain sorted.
- Is element access still constant time?

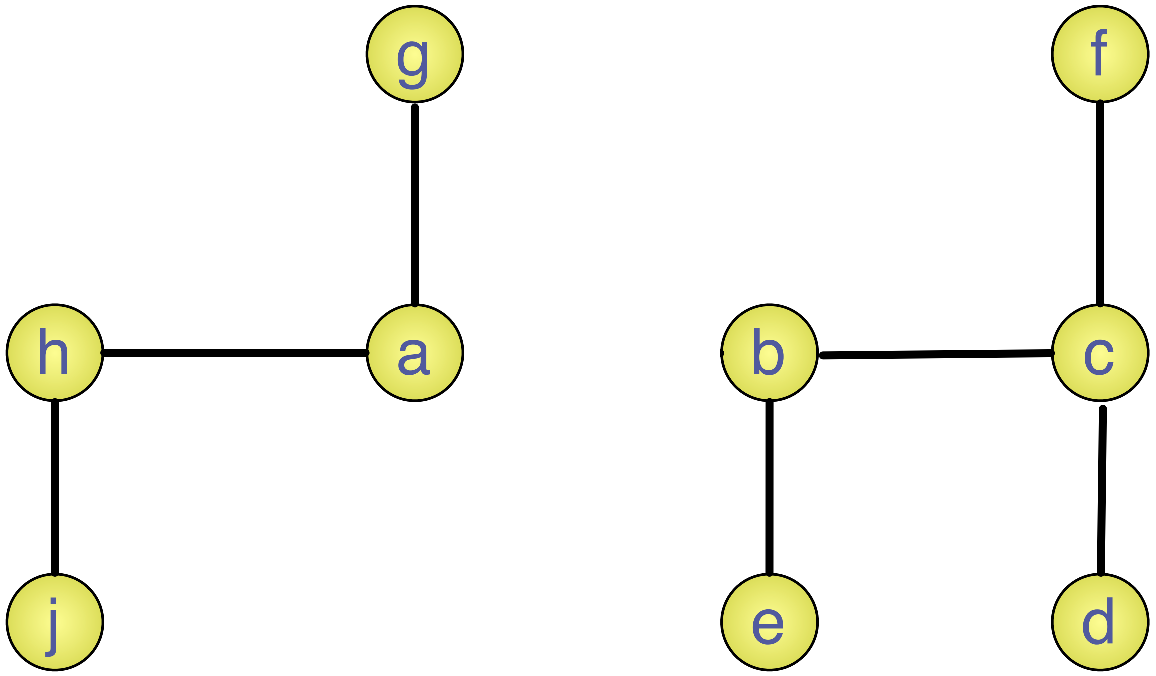
Trees

- Levitin: free tree ≡ connected acyclic graph



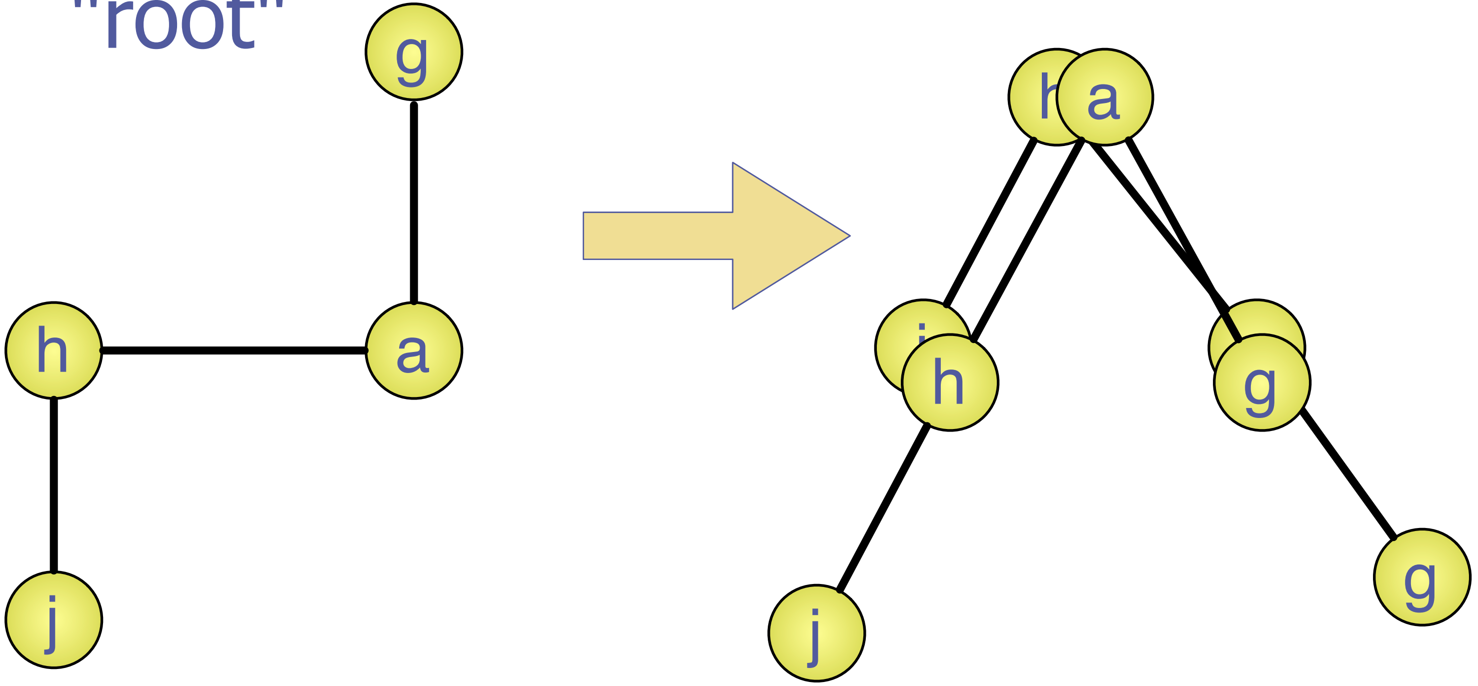
Trees

- Levitin: forest ≡ acyclic graph



Rooted Tree

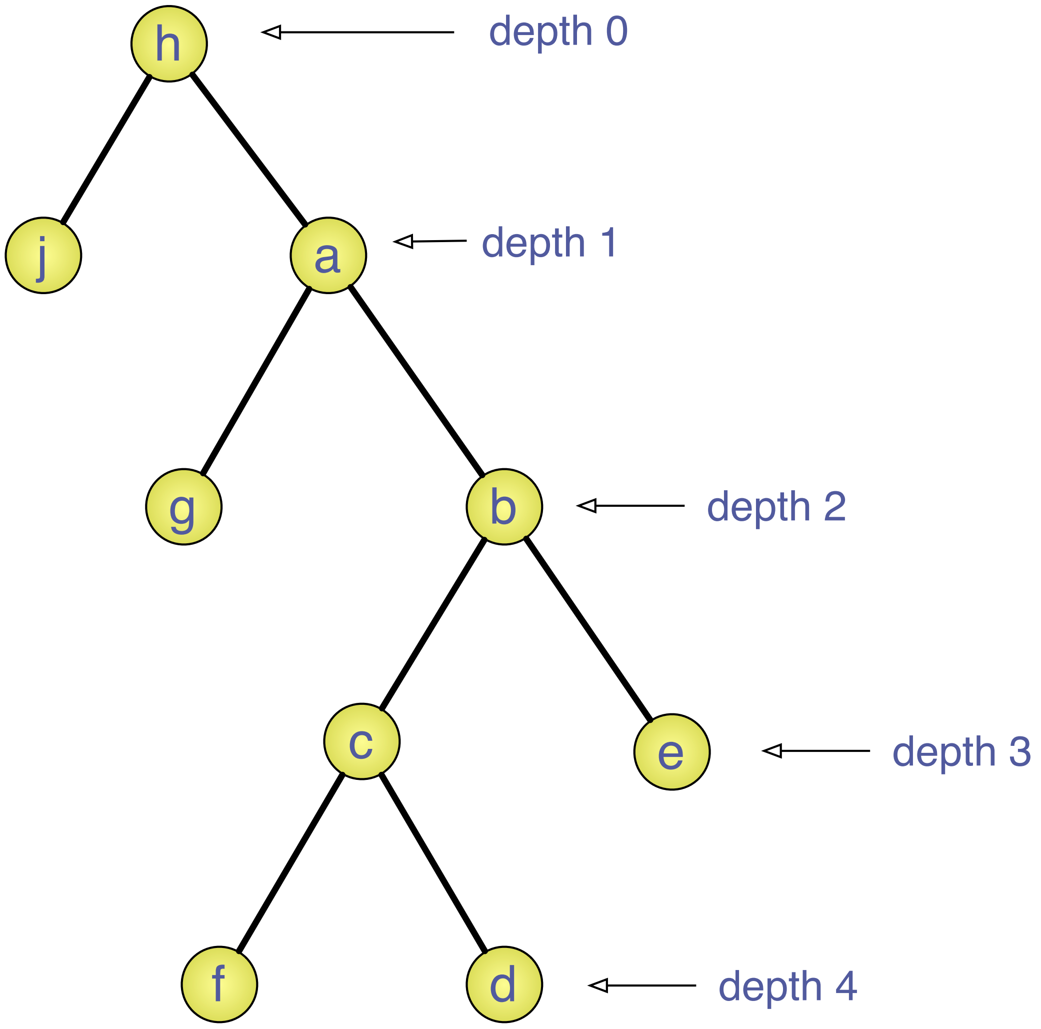
- In a tree, ∃ a unique path from one node to another
- So we can pick an arbitrary node as the "root"



Tree Depth & Tree Height

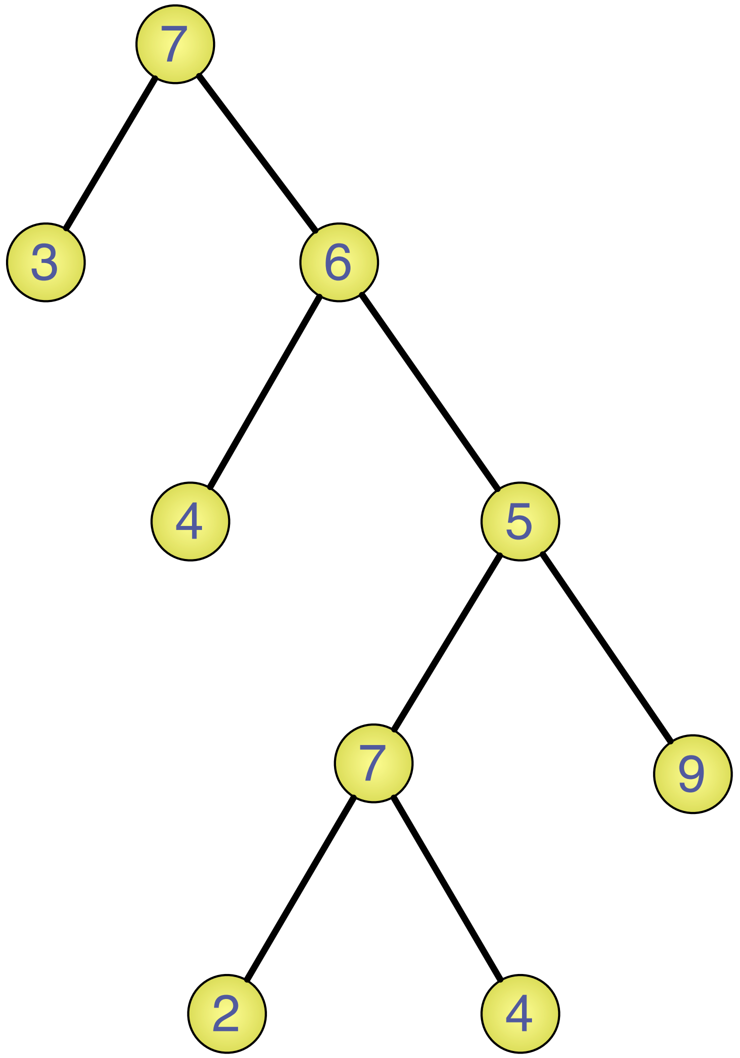
Q: What’s the height of an **empty** tree?

- height = maximum depth = number of levels – **1**



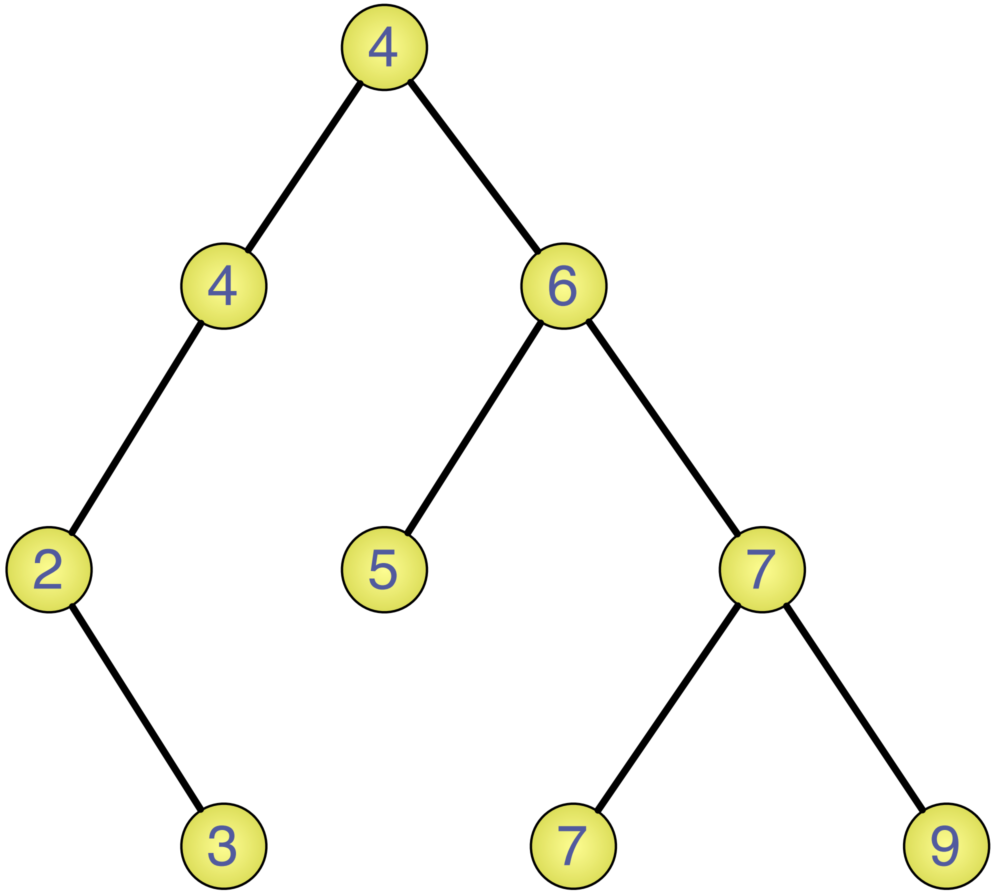
Ordered Tree

- children are ordered (left to right)



Search Tree

- children are ordered (left to right)
- left children ≤ parent < right children



Size of Numbers

- How many bits are there in the binary representation of a decimal number n?

A. $\log_{10} n$

B. $\lg n$

C. $\lfloor \lg n \rfloor$

D. $\lceil \lg n \rceil$

E. none of the above

Binary Representations

- The number of bits b in the binary representation of a number n is given by

$$b = \lfloor \lg n \rfloor + 1$$

Using this formula, how many bits are required to represent the number 2^{30} ?

A. 29

B. 30

C. 31

D. none of these

Binary Representations

- The number of bits b in the binary representation of a number n is given by

$$b = \lfloor \lg n \rfloor + 1$$

Using this formula, how many bits are required to represent the number (2^{30} - 1)?

A. 29

B. 30

C. 31

D. none of these

Basic operation

An algorithm has multiplication as its basic operation. A multiplication takes time t_m, on average. On an input of size n, the algorithm performs its basic operation $C(n)$ times.

What's the approximate run time $T(n)$ of the algorithm?

A. $C(n)$

B. $O(C(n))$

C. $t_m \, C(n)$

D. $t_m / C(n)$

E. $C(n) / t_m$

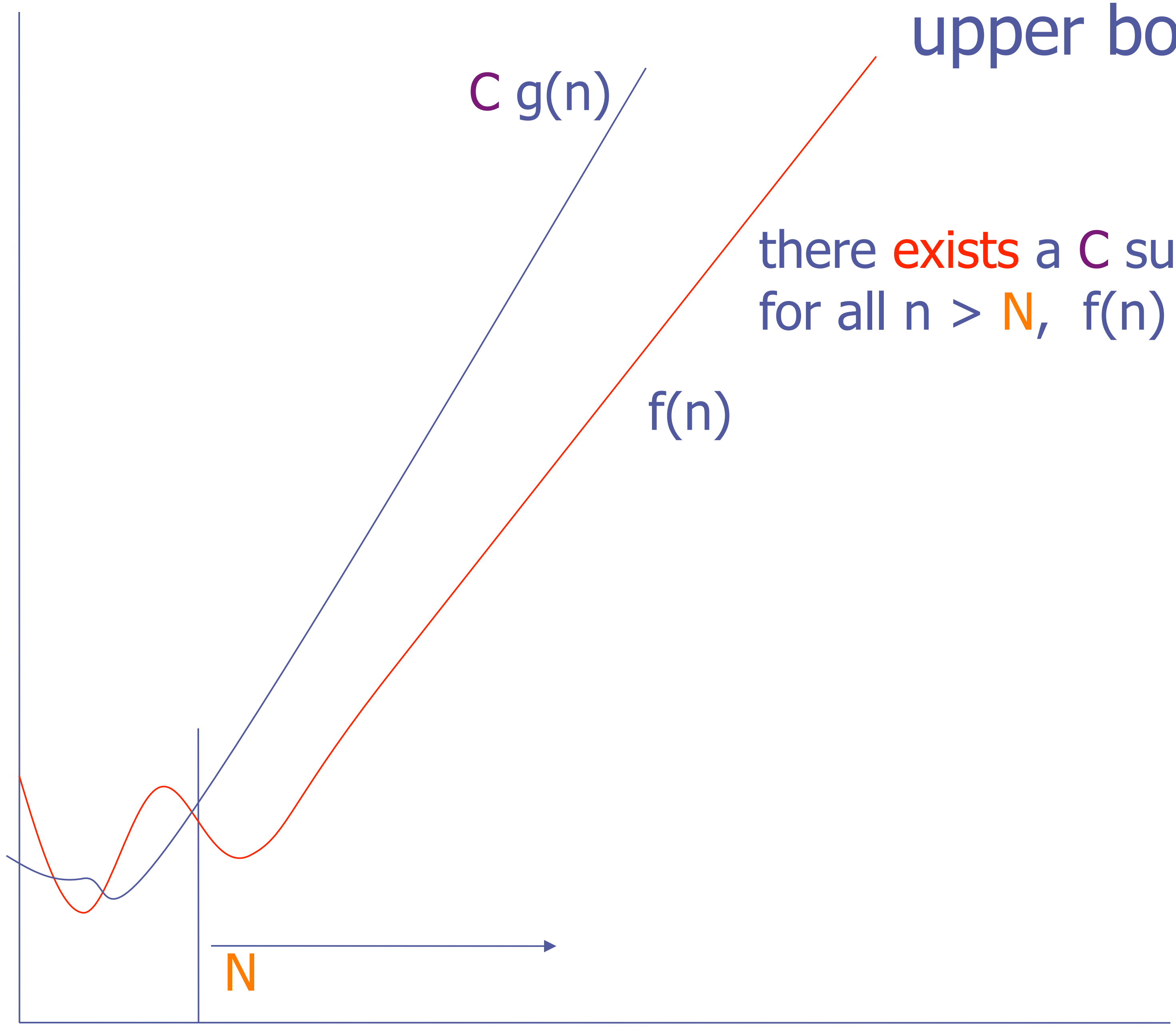
F. none of the above

Running Time

- Suppose $T(n) = 4n^3$ seconds
- So $T(10) = 4000$ s   ($4000\text{s} \approx 1.1$ hour)
- What's $T(1\,000)$?

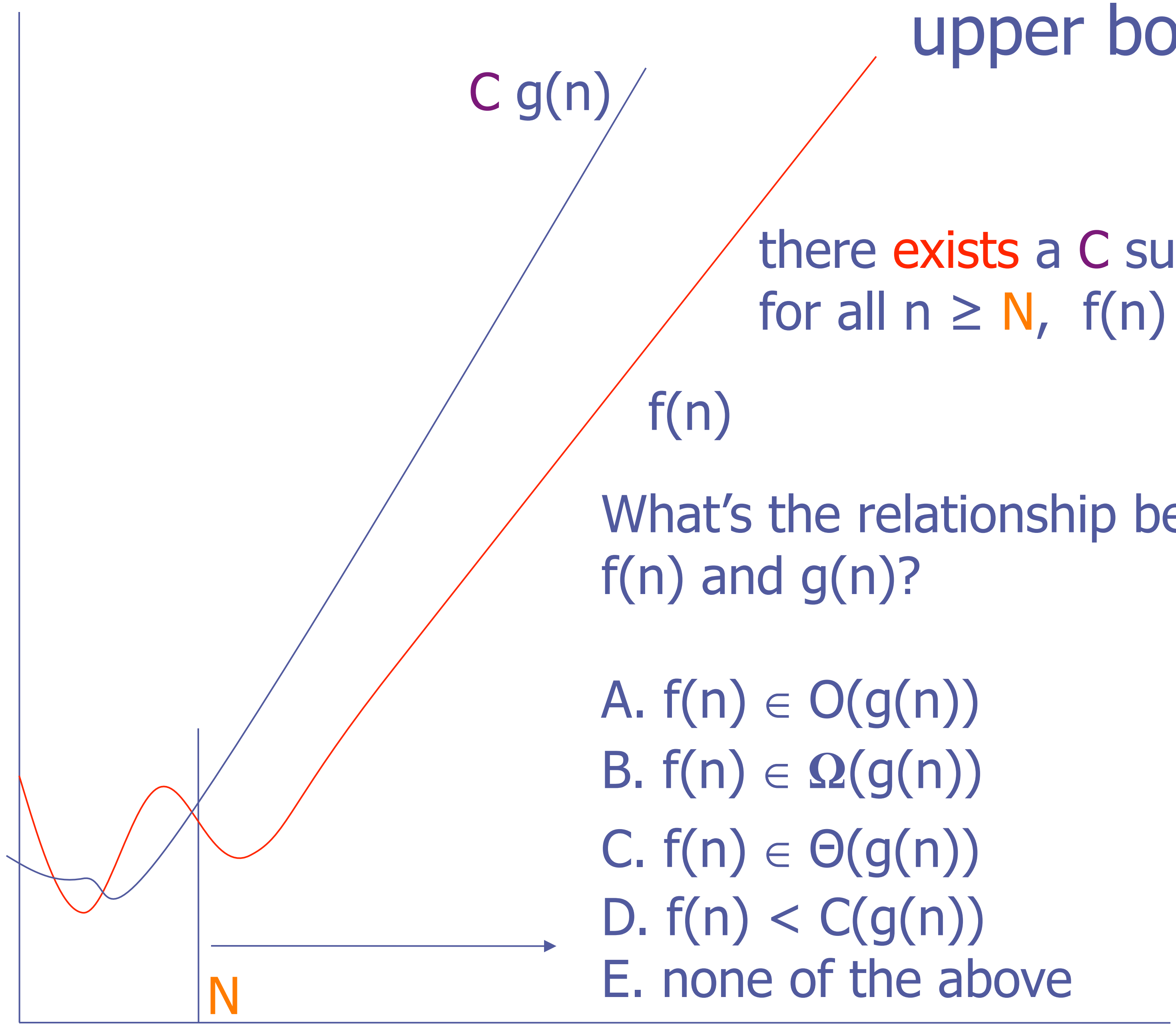
A.	10 000 s	(≈ 2.8 hours)
B.	40 000 s	(≈ 11 hours)
C.	4 000 000 s	(≈ 6.6 weeks)
D.	1 000 000 000 s	(31 years)
E.	4 000 000 000 s	(127 years)

upper bounds



there exists a C such that,
for all n > N, f(n) ≤ C g(n).

upper bounds



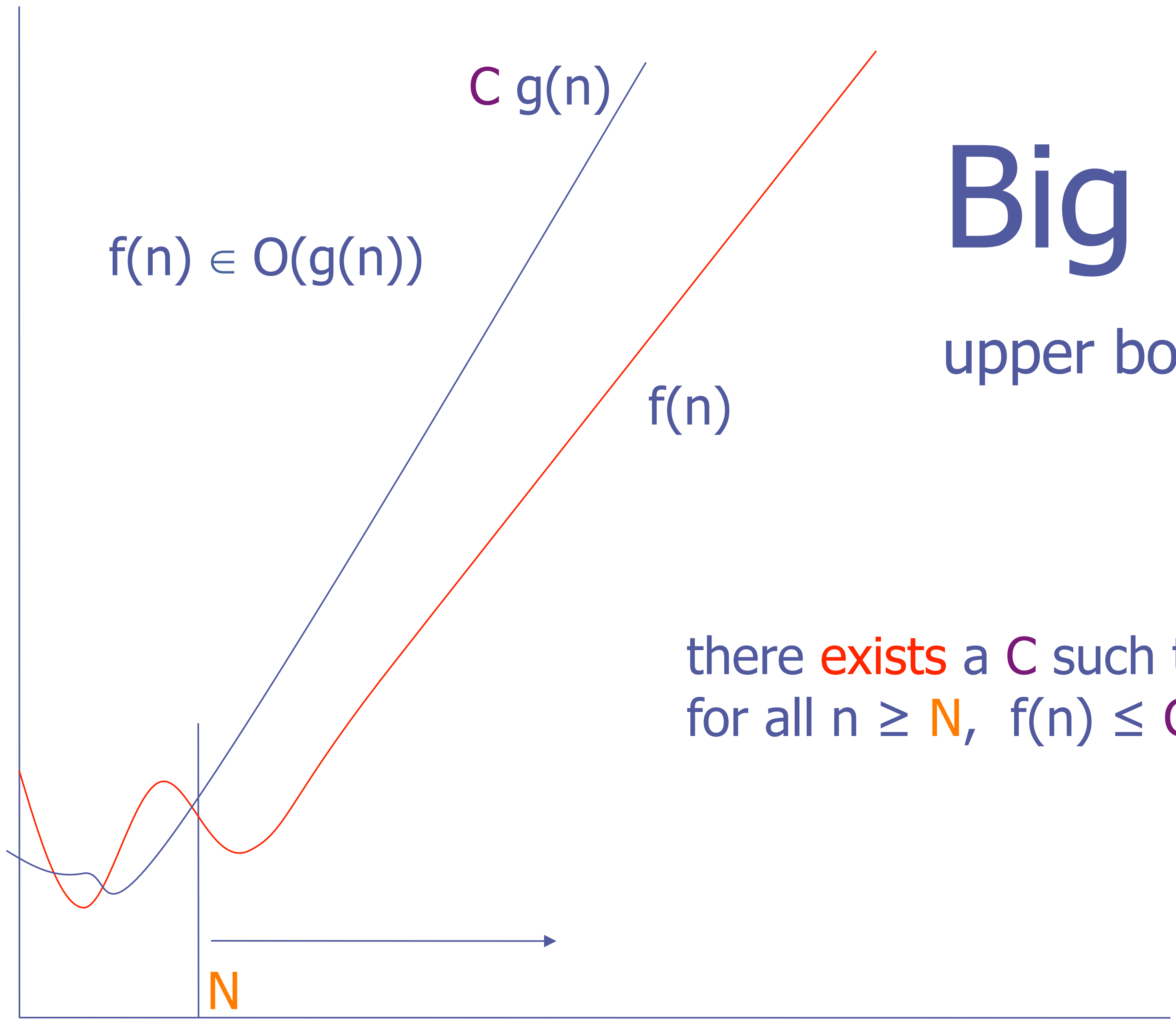
there exists a C such that,
for all n ≥ N, f(n) ≤ C g(n).

What's the relationship between f(n) and g(n)?

A. $f(n) \in O(g(n))$
B. $f(n) \in \Omega(g(n))$
C. $f(n) \in \Theta(g(n))$
D. $f(n) < C(g(n))$
E. none of the above

Big Oh

upper bound



there exists a C such that,
for all $n \geq N$, $f(n) \leq C\ g(n)$.

Formalizing Asymptotic Nota

"Big Oh"

- A function f(n) is said to be O(g(n)) if there are constants c>0 and N>0 such that:

$$f(n) \leq c\, g(n) \quad \text{for all } n \geq N$$

- In other words: for large enough input n, f(n) is no more than a constant multiple of g(n).
- Big Oh is used for stating <u>upper bounds</u>.

Which of the following is true:

A. $3n^2 + 500 \in O(n)$

B. $3n^2 + 500 \in O(n^2)$

C. $3n^2 + 500 \in O(n^3)$

D. A & B

E. B & C

F. none of the above

Which of the following is true:

A. $3n^3/500 \in O(n)$

B. $3n^3/500 \in O(n^2)$

C. $3n^3/500 \in O(n^3)$

D. A & B

E. B & C

F. none of the above

Examples:

- $4n^2 + 3 \in O(n^2)$
- $4n^3 + 3 \in O(n^3)$
- $n^2/1000 + 3000n \in O(n^2)$

In general:

- Can ignore all but the highest power
- Can ignore coefficients

Logarithms

- Which of the following is true?

  A. $O(\ln n) = O(\log_{10} n)$

  B. $O(\lg n) = O(\ln n)$

  C. $\lg n = \ln n$

  D. all of the above are true

  E. none of the above is true

  F. A and B are true

  G. B and C are true

Powers

- Which of the following is true

A. $O(4^n) = O(2^n)$

B. $O(2 \times 2^n) = O(10 \times 2^n)$

C. both of the above are true

D. neither of the above is true

More Examples:

Logarithms:

- Can ignore base because:

$$\log_a b = \log_c b / \log_c a.$$

- Thus $O(\log_2 n)$ is the same as $O(\log_{10} n)$.

Exponents:

- Can ignore non-exponential terms
- Base of exponentiation is important; for example, $O(4^n)$ is bigger than $O(2^n)$.

More Properties of Big Oh:

- $\mathcal{O}$ notation is additive and multiplicative:

  If $f(n) \in \mathcal{O}(s(n))$ and $g(n) \in \mathcal{O}(t(n))$, then:

  - $f(n) + g(n) \in \mathcal{O}(s(n) + t(n))$;
  - $f(n)g(n) \in \mathcal{O}(s(n)t(n))$.

- $\mathcal{O}$ notation is transitive:

  If $f(n) \in \mathcal{O}(g(n))$, and $g(n) \in \mathcal{O}(h(n))$, then $f(n) \in \mathcal{O}(h(n))$.

Classes of Algorithm:

There are standard names for some of the most common complexity classes:

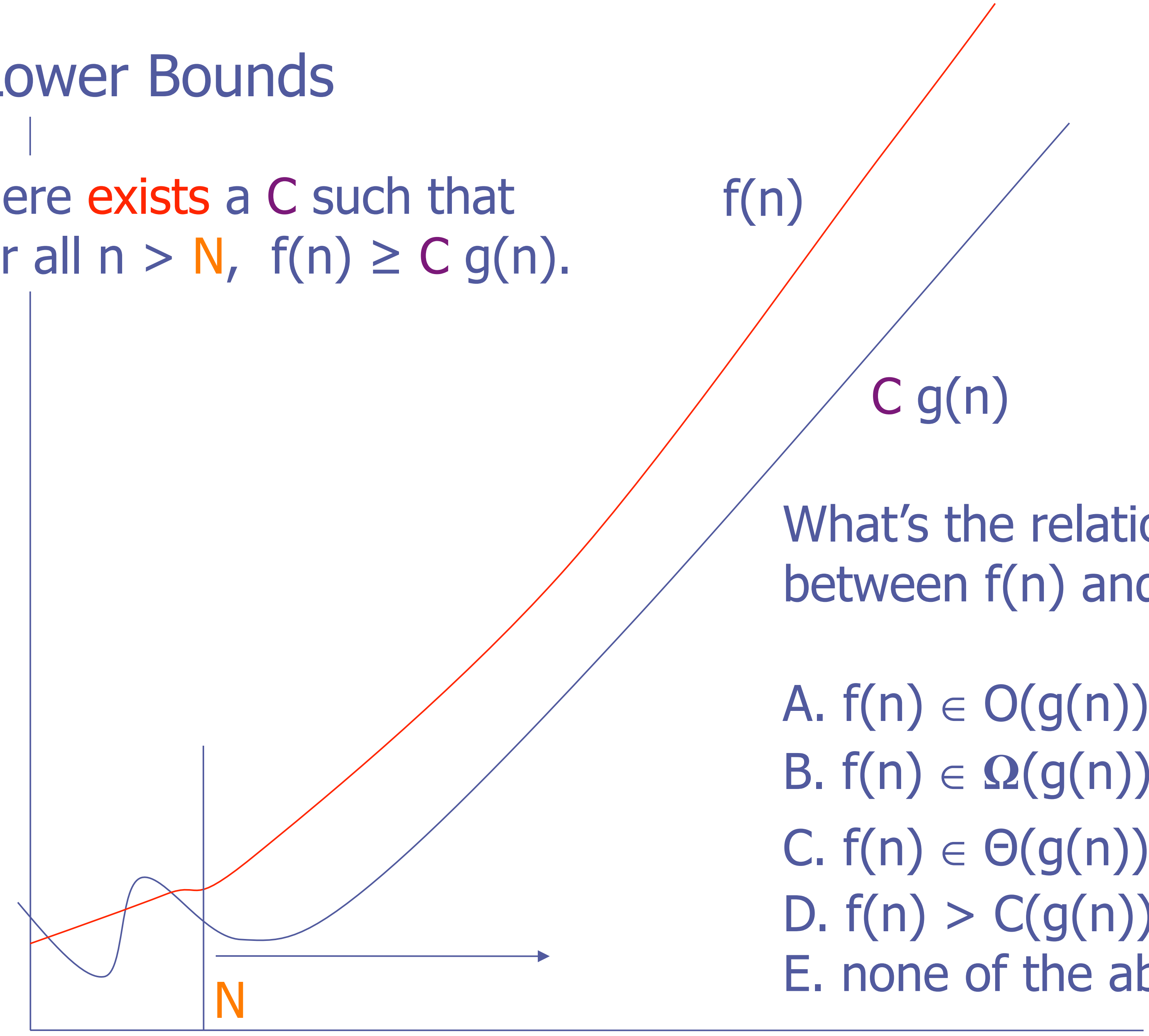
- Constant: $O(1)$
- Logarithmic: $O(\log n)$
- Linear: $O(n)$
- Linearithmic: $O(n \log n)$
- Quadratic: $O(n^2)$
- Exponential: $O(2^n)$
- Double Exponential: $O(2^{2^n})$

Polynomial Algorithms:

- An algorithm is said to be polynomial if it is $O(n^p)$ for some integer p.
- Terminology:
  - Problems with polynomial algorithms are generally considered to be tractable.
  - Problems for which no polynomial algorithm has been found are often considered intractable.

Lower Bounds

there exists a C such that
for all n > N, $f(n) \geq C\, g(n)$.



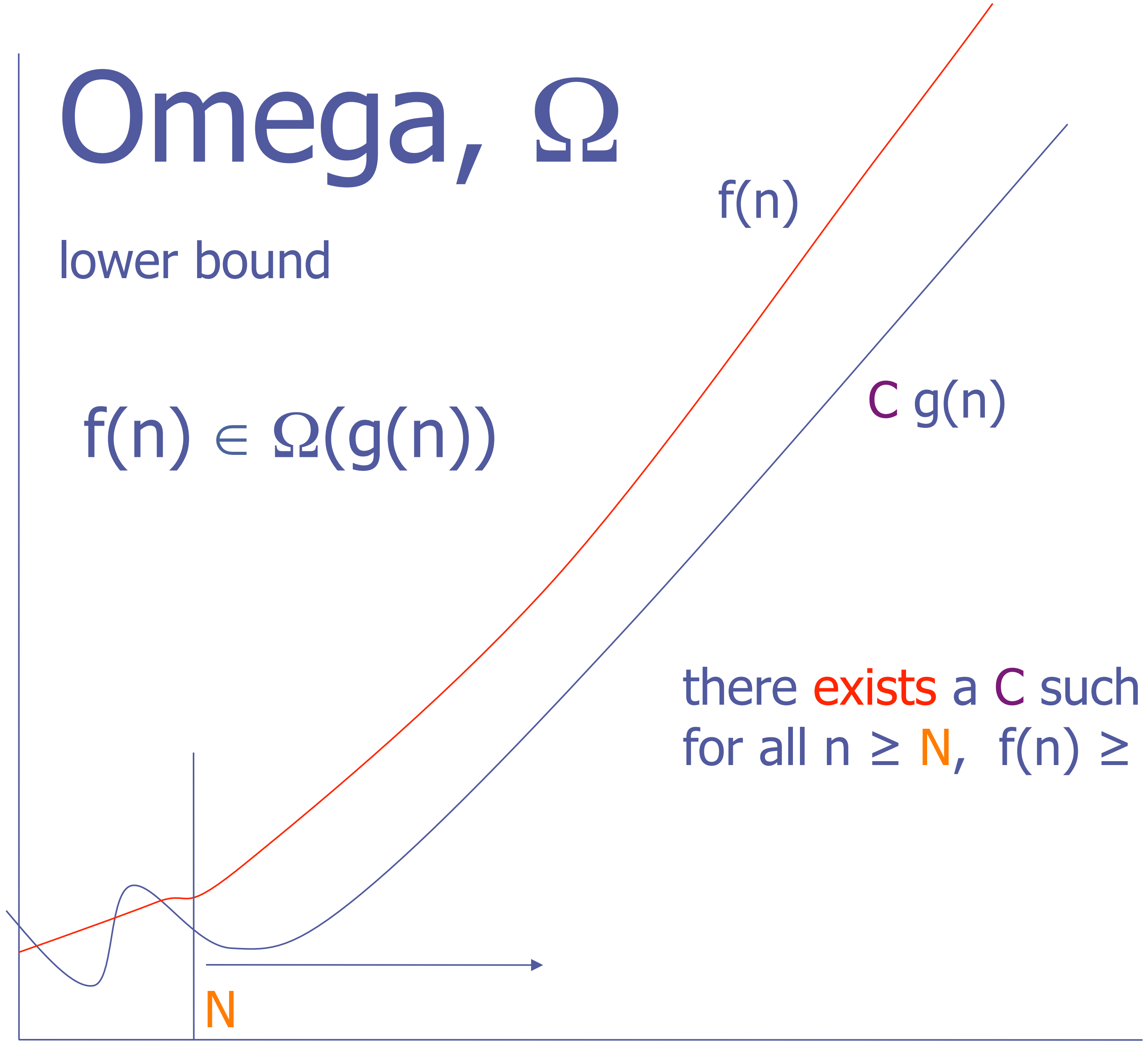
What's the relationship between f(n) and g(n)?

A. $f(n) \in O(g(n))$
B. $f(n) \in \Omega(g(n))$
C. $f(n) \in \Theta(g(n))$
D. $f(n) > C(g(n))$
E. none of the above

Omega, Ω

lower bound

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



there exists a C such that
for all $n \geq N$, $f(n) \geq C\, g(n)$.

Dealing with Lower Bounds:

- "This algorithm takes at least …"

Omega

- A function $f(n)$ is said to be in $\Omega(g(n))$ if there are constants $c>0$ and $N>0$ such that:

$$f(n) \geq c\ g(n) \quad \text{for all } n \geq N$$

- Note that $f(n) \in \Omega(g(n))$ if and only if $g(n) \in O(f(n))$.

Mnemonics

- Big Oh is really a Capital greek letter Omicron; pronounce it O-**micron**.
  Pronounce **Ω** O-**mega.**
- Read $f(n) \in O(g(n))$ as f is O-smaller-than g
- Read $f(n) \in \Omega(g(n))$ as f is O-larger-than g
  - The large O (O, **Ω**) says: f may be equal to g
  -

Tight Bounds:

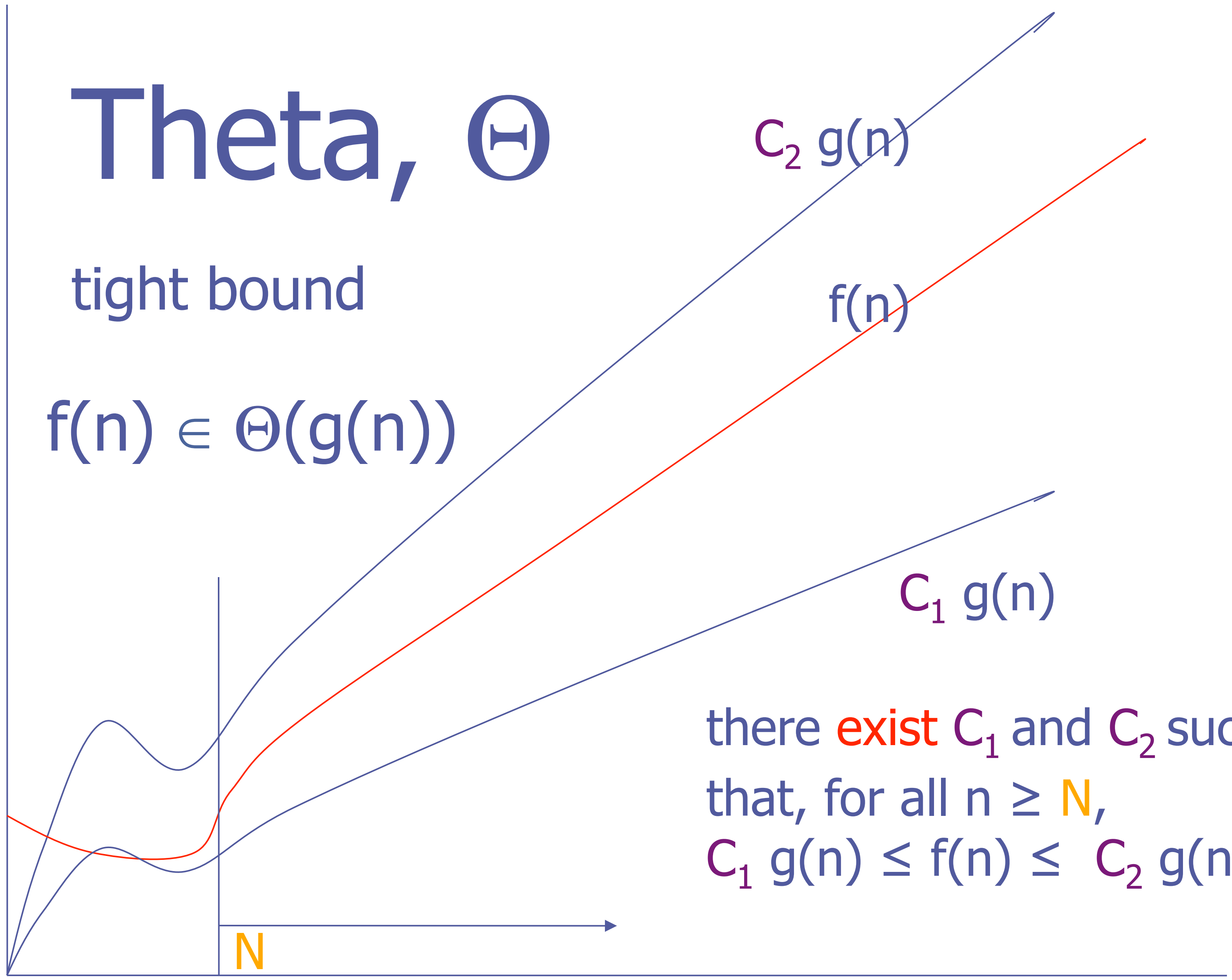
- A function f(n) is said to be in Θ(g(n)) if it is in both O(g(n)) and Ω(g(n)).
  - If f(n) $\in$ Θ(g(n)), then it is eventually "sandwiched" between constant multiples of g(n).

- $f(n) \in \Theta(g(n))$ if and only if $\lim_{n\to\infty} \frac{g(n)}{f(n)} = c$

Theta, Θ

tight bound

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



there exist C_1 and C_2 such that, for all $n \geq N$,
$C_1\ g(n) \leq f(n) \leq C_2\ g(n)$.

Simple laws of $\Theta(..)$ notation:

- Addition:

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

- Scaling: for any constant $c>0$,

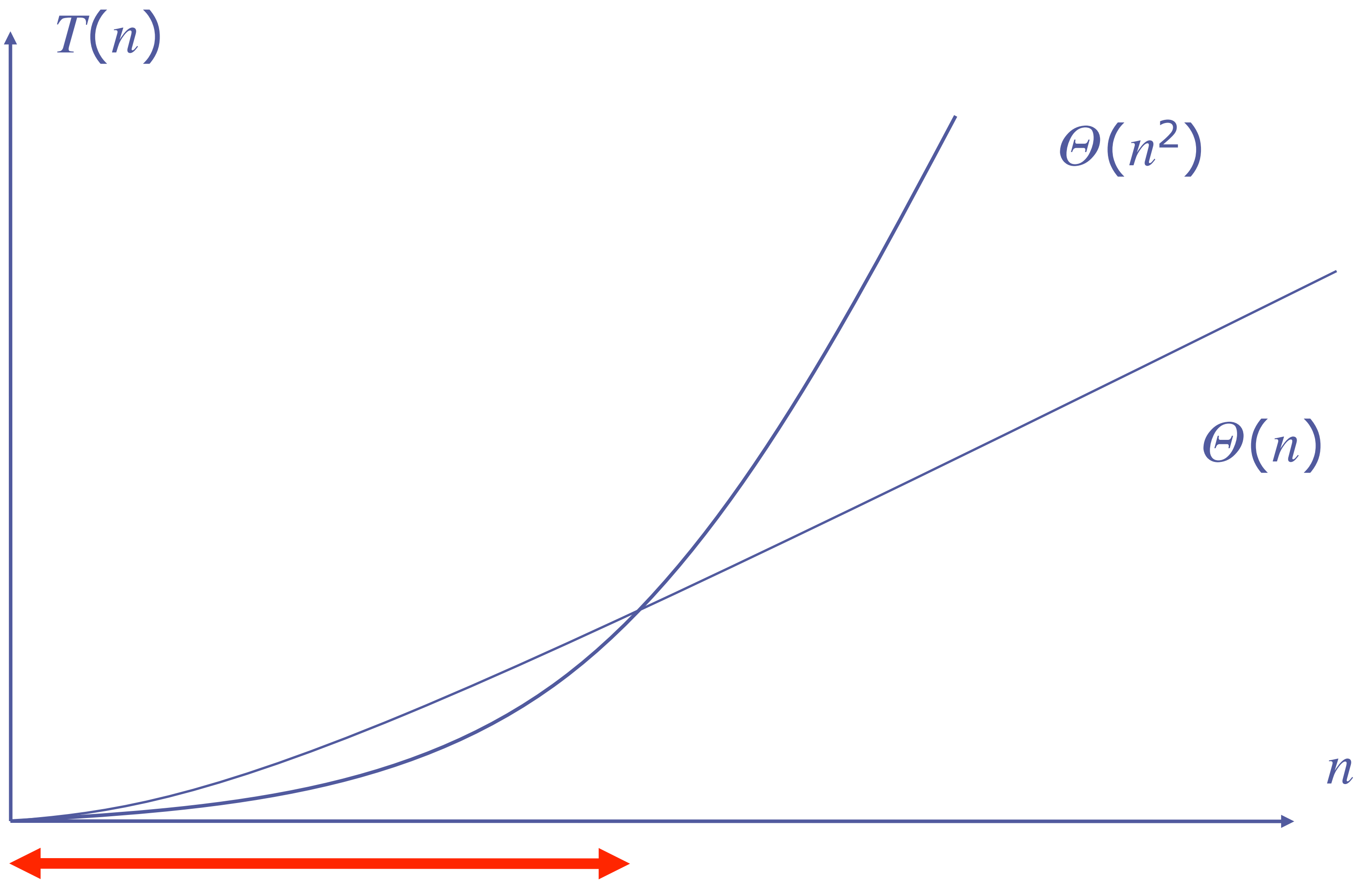
  $$\Theta(cf(n)) = c\,\Theta(f(n)) = \Theta(f(n))$$

True or False

- You have two sorting algorithms:
  B is $O(n^2)$, while
  Q is $O(n \lg n)$.
- True or false: Q is always faster than B
  - A. True
  - B. False

Beware Constant Factors!

- Use complexity measures with care!
- A $\Theta(n^2)$ algorithm might actually be faster than a $\Theta(n)$ algorithm for all values of n encountered in some real application!



The $\Theta(n^2)$ algorithm is faster than the $\Theta(n)$ alternative if we're working within this particular range …

Beware Constant Factors!

- Use complexity measures with care!
- A $\Theta(n^2)$ algorithm might actually be faster than a $\Theta(n)$ algorithm for all values of n encountered in some real application!
- How would you find out?

  A. more careful analysis for different n

  B. measure the implementation for different n

  C. neither of the above

Comparing Orders of Growth

- If you need to compare the rates of growth of two functions, t and g, the easiest way is often to take limits:

$$\lim_{n\to\infty} \frac{t(n)}{g(n)} = \begin{cases} 0 & \Rightarrow t(n) \text{ has a smaller order of growth than } g(n) \\ c > 0 & \Rightarrow t(n) \text{ has the same order of growth as } g(n) \\ \infty & \Rightarrow t(n) \text{ has a larger order of growth than } g(n) \end{cases}$$

Comparing Orders of Growth

- If you need to compare the rates of growth of two functions, t and g, the easiest way is often to take limits:

$$\lim_{n\to\infty} \frac{t(n)}{g(n)} = \begin{cases} 0 & \Rightarrow t(n) \text{ has a smaller order of growth than } g(n) \\ c > 0 & \Rightarrow t(n) \text{ has the same order of growth as } g(n) \\ \infty & \Rightarrow t(n) \text{ has a larger order of growth than } g(n) \end{cases}$$

- Which function goes on top of the limit?
  - A. The one you think grows slower
  - B. The one you think grows faster
  - C. It doesn't matter

Example

- Prove that the functions a^n and b^n have different orders of growth if $a \neq b$

$$\lim_{n\to\infty} \frac{a^n}{b^n} = \lim_{n\to\infty} \left(\frac{a}{b}\right)^n =$$

Summary:

- Asymptotic notation using O, Ω, and Θ
  - $O(f(n))$ is an upper bound
  - $\Omega(f(n))$ is a lower bound
  - $\Theta(f(n))$ sets tight bounds

Square roots

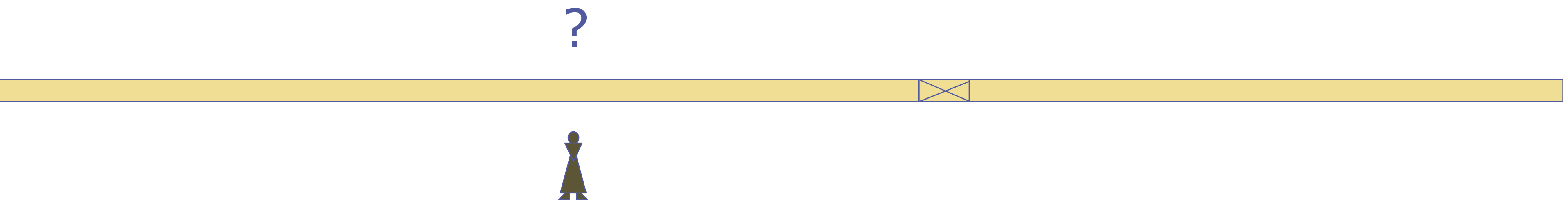
- Write pseudocode for an algorithm that computes $\lfloor \sqrt{n} \rfloor$ for any positive integer n. Besides assignment and comparison, your algorithm may use only the four basic arithmetic operations.

Euclid’s Euclid

- Euclid’s algorithm, as presented in Euclid’s treatise, uses subtractions rather than **mod**. Write pseudocode for this version of Euclid’s algorithm.

Door in a Wall

- You are facing a wall that stretches infinitely in both directions. There is a door in the wall, but you know neither how far away, nor in which direction. You can see the door only when you are right next to it.
- Design an algorithm that enables you to reach the door.
- Write an expression for the number of steps that your algorithm will take. Your expression should be in terms of n, the (unknown to you) number of steps between your initial position and the door.



Which way do you walk?

A. To the right
B. To the left
C. It doesn't matter

How far do you go?

A. 1 step

B. 2 steps

C. Until you are in front of the door

D. k steps, for some fixed k

E. The lesser of C and D

What do you do then?