

Solve this recurrence relation by repeated substitution. I am going to ask you to hand in your work so do it on a clean piece of paper.

You will get credit for participation as long as you hand something in.

Assume that  $n = 2^k$  for some integer  $k \geq 0$ .

The recurrence relation has

$$T(1) = 14$$

$$\text{For } n \geq 2, T(n) = n + T(n/2)$$

Make sure you number your steps starting at step 0.

# Announcements

Assignment #1:

Part 1A Programming Questions: Upload to connex by **Saturday Sept. 23 at 11:55pm.**

Double check the assignment specifications before submitting your assignment to ensure you followed the instructions. Put your name inside your program (in the comments at the top).

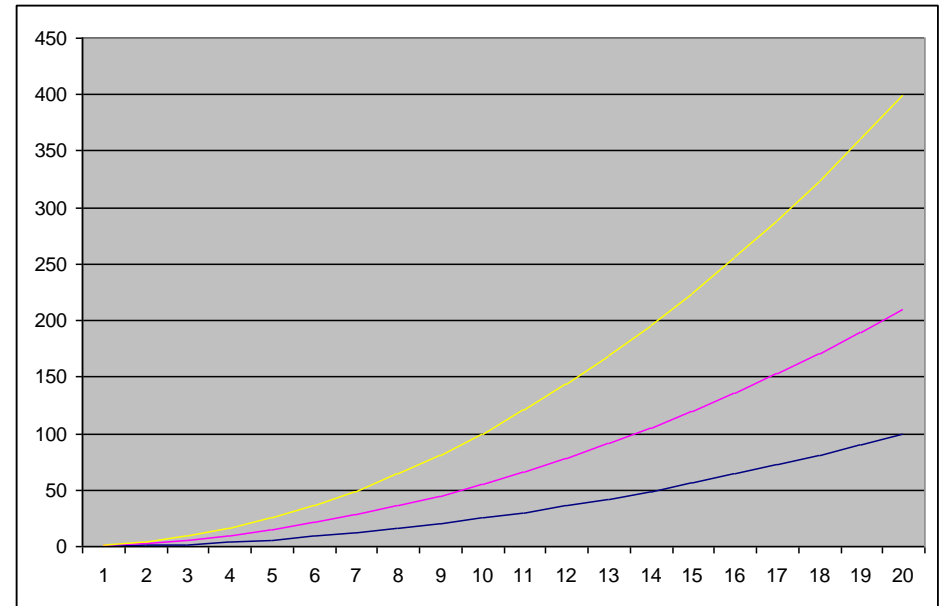
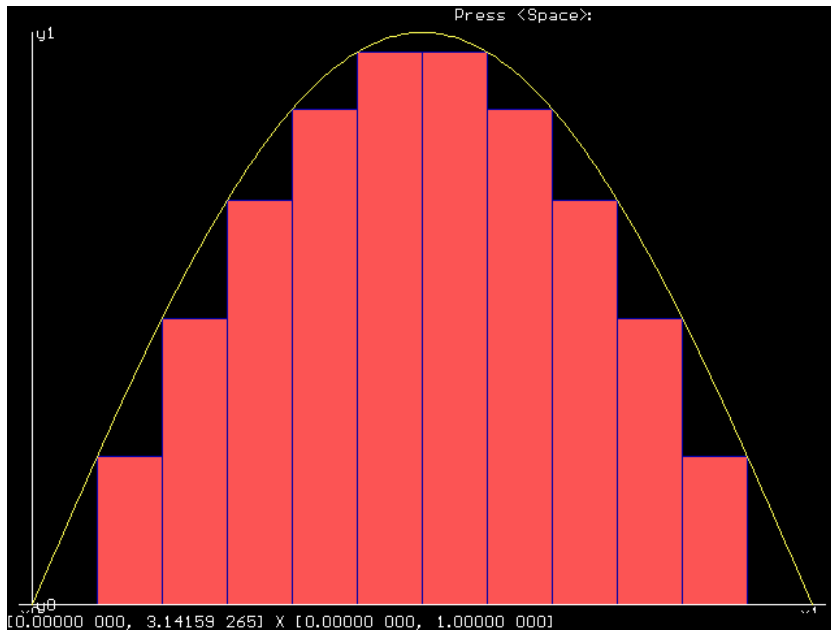
Connex is set up to accept an unlimited number of resubmissions up until the deadline. It's a good idea to upload what you have early- you can revise it later if you improve the code.

Part 1B Written Questions: Hand in on paper at the beginning of class on **Thursday September 28.**

It's better to hand in a partially completed assignment than to hand in nothing at all. You must pass the assignments to pass the course.

**For help: stay after class today or send e-mail. Attach your program.**

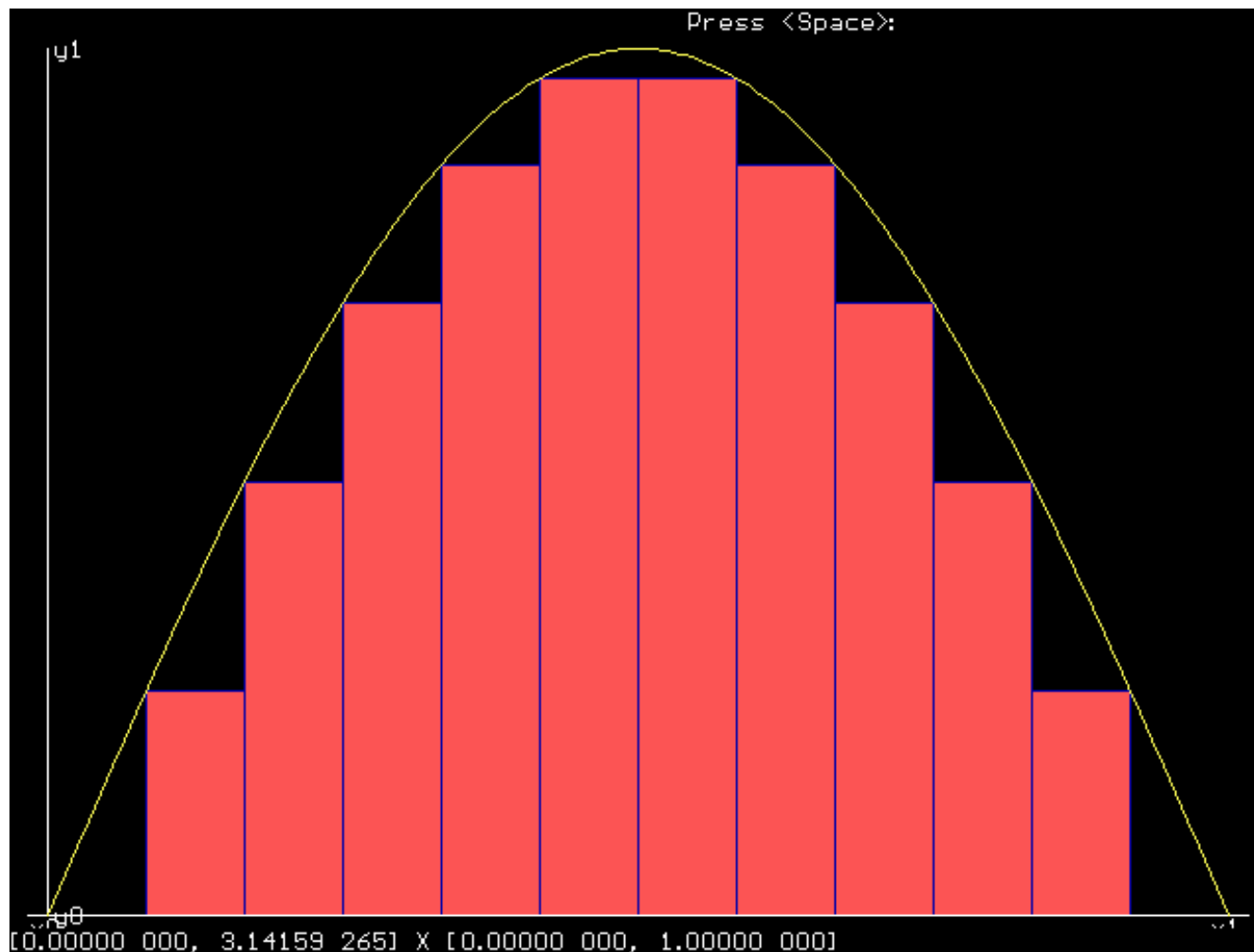
# Mathematics of Algorithm Analysis



# Outline

- Lower bounds and upper bounds on functions.
- Terminology for talking about the amount of time or space that an algorithm uses.

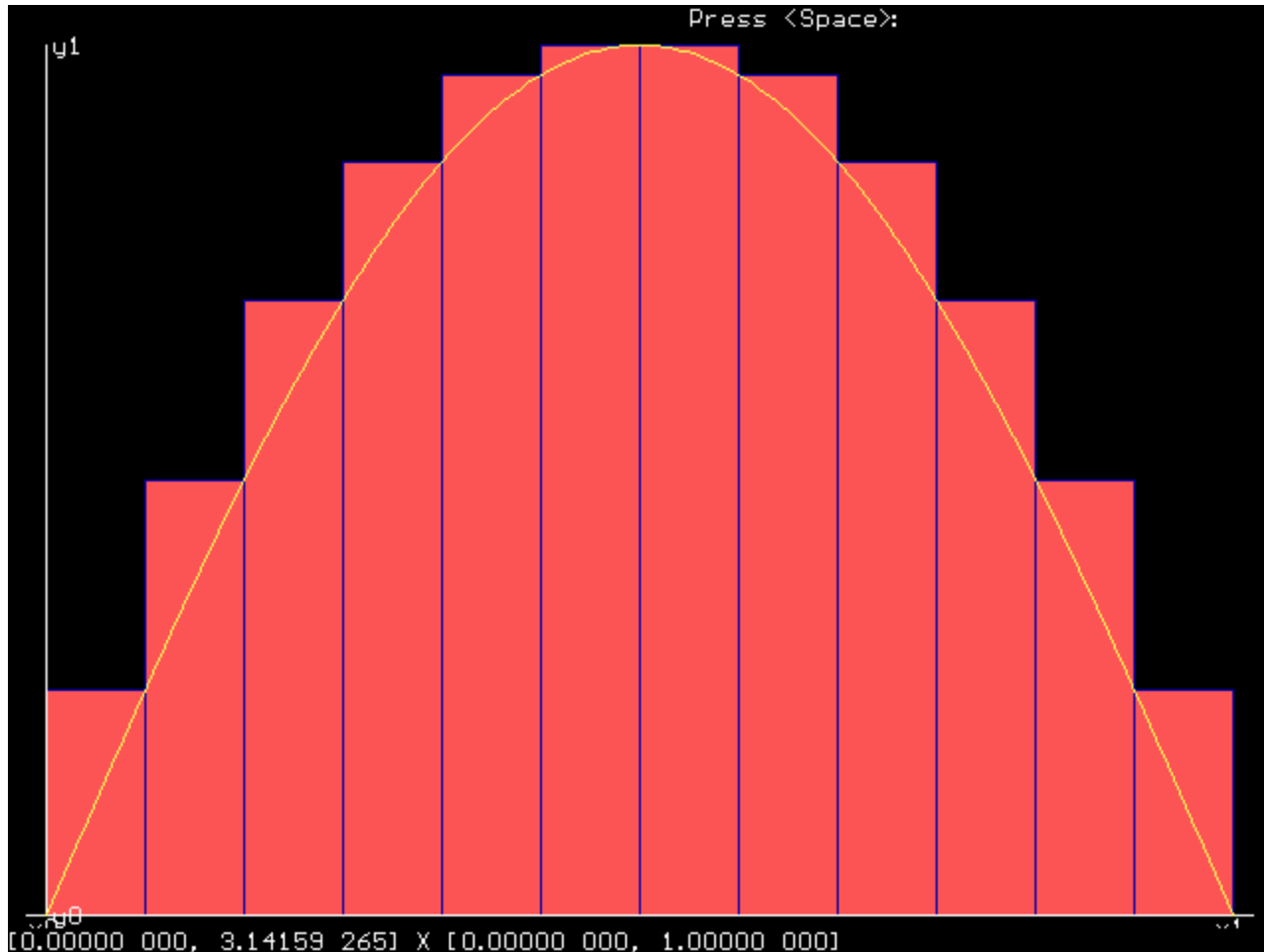
The area in the red boxes is a **lower bound** for the area under the yellow curve.



Picture from

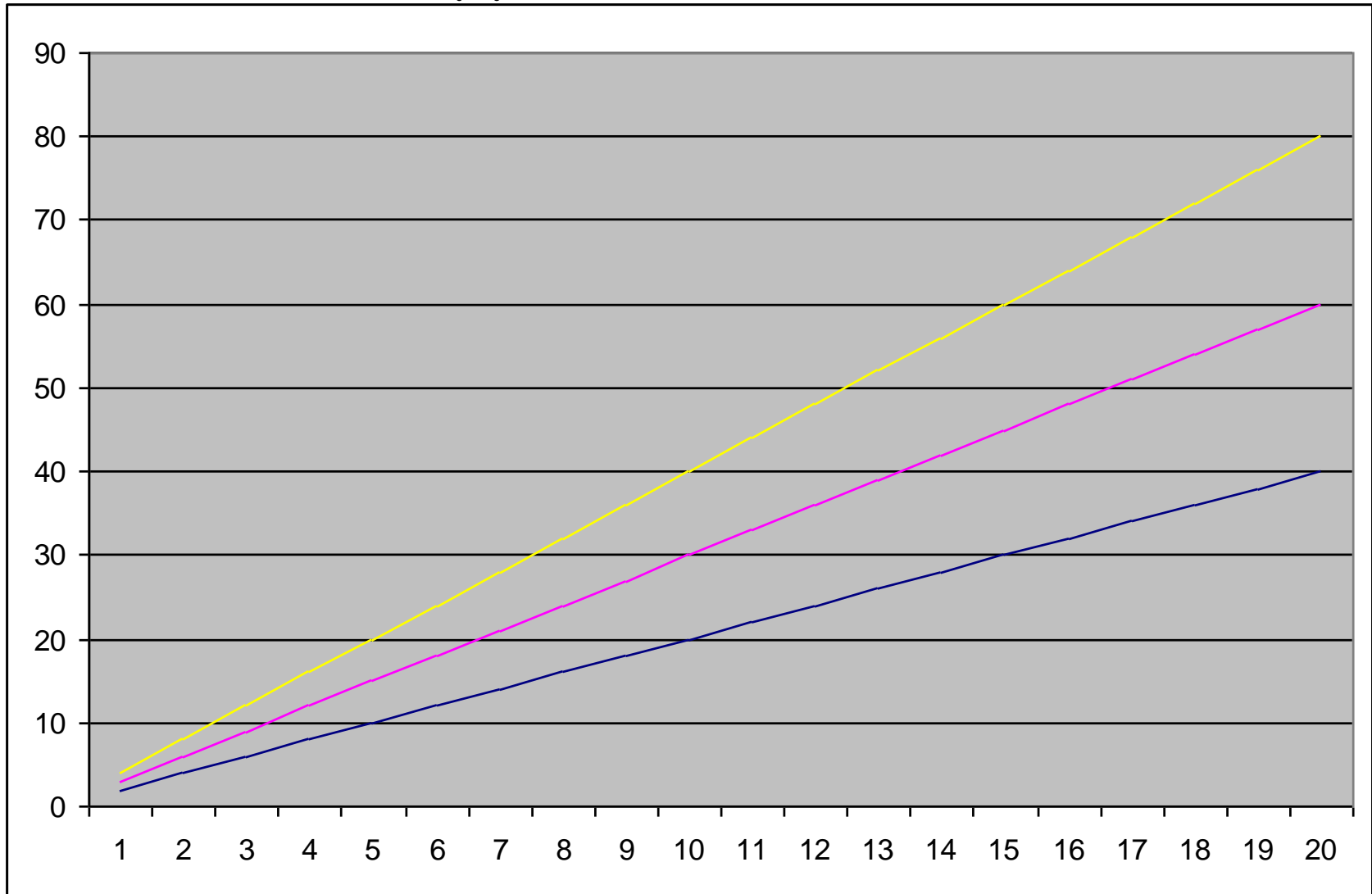
[http://archives.math.utk.edu/visual.calculus/4/riemann\\_sums.3/microcalc.html](http://archives.math.utk.edu/visual.calculus/4/riemann_sums.3/microcalc.html)

The area in the red boxes is an **upper bound** for the area under the yellow curve.



Picture from  
[http://archives.math.utk.edu/visual.calculus/4/riemann\\_sums.3/microcalc.html](http://archives.math.utk.edu/visual.calculus/4/riemann_sums.3/microcalc.html) 6

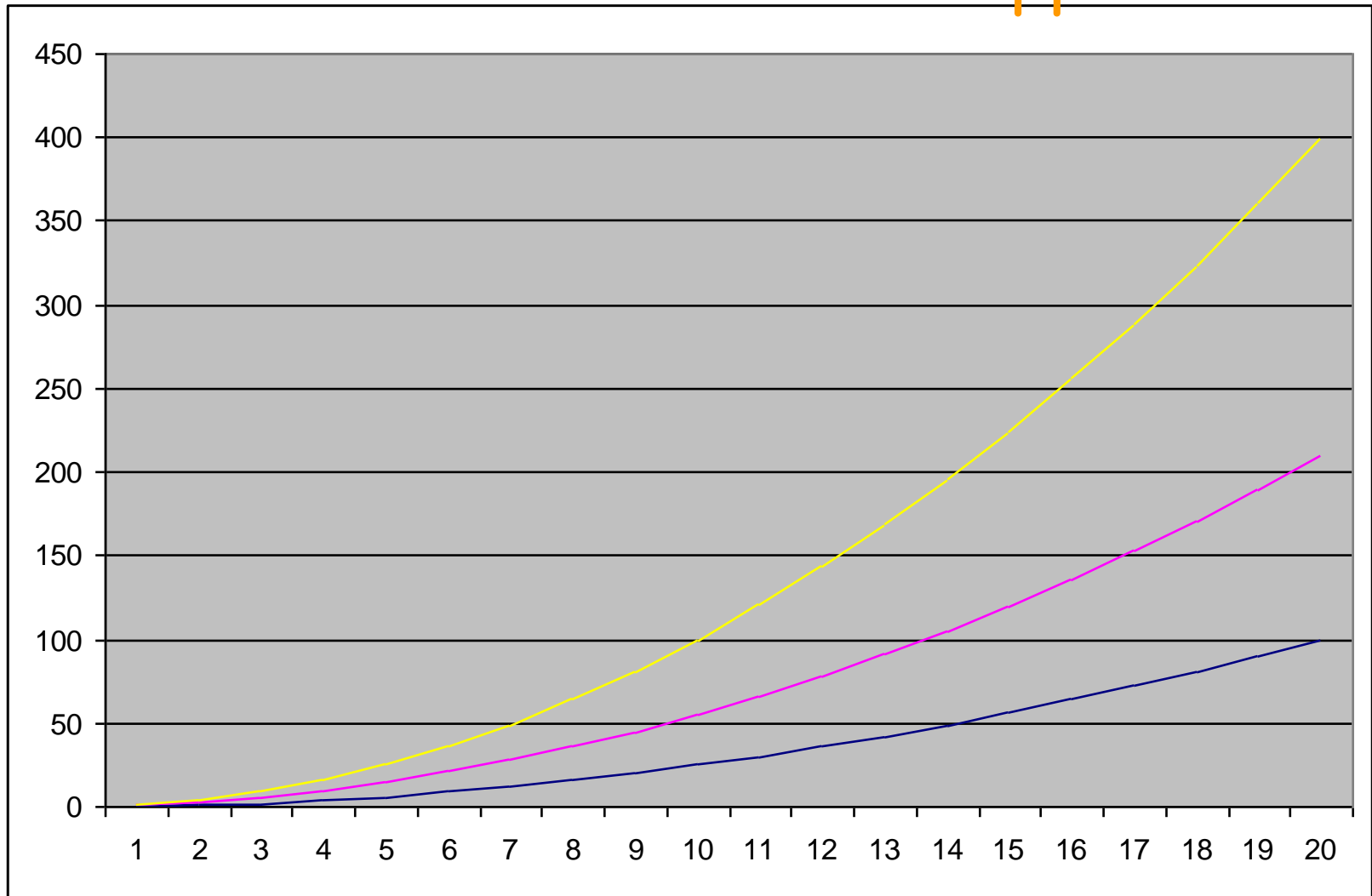
The function  $2n$  is a lower bound for  $3n$ , and  $4n$  is an upper bound ( $n \geq 0$ ).



$$n^2 / 4 \leq n(n+1)/2 \leq n^2 \text{ for } n \geq 0.$$

lower bound

upper bound





**Definition: Lower bound.**

A function  $f(x)$  is a *lower bound* for  $g(x)$  over a range  $R$  if for all  $x$  in  $R$ ,  $f(x) \leq g(x)$ .

**Definition: Upper bound.**

A function  $f(x)$  is an *upper bound* for  $g(x)$  over a range  $R$  if for all  $x$  in  $R$ ,  $f(x) \geq g(x)$ .

**Definition: Optimal.** A solution is *optimal* if it is impossible to do better. What "better" means depends on the problem situation.

Why do we care about lower and upper bounds?

When analyzing algorithms, it is often easier to bound the amount of work done than to compute it exactly.

One example:

$$1 + 2 + 3 + 4 + \dots + (n-2) + (n-1) + n.$$

We know a closed formula for this:  $n(n+1)/2$ .

But assume for a minute we do not and let's work out some bounds.

General Technique for bounding a sum:

Assume  $a_i$ ,  $b_i$ , and  $c_i \geq 0$  for  $i = 1, 2, 3, \dots, n$ .

$$A = \sum_{i=1}^n a_i$$

If  $a_i \leq b_i \leq c_i$  for  $i = 1, 2, 3, \dots, n$   
then  $A \leq B \leq C$ .

$$B = \sum_{i=1}^n b_i$$

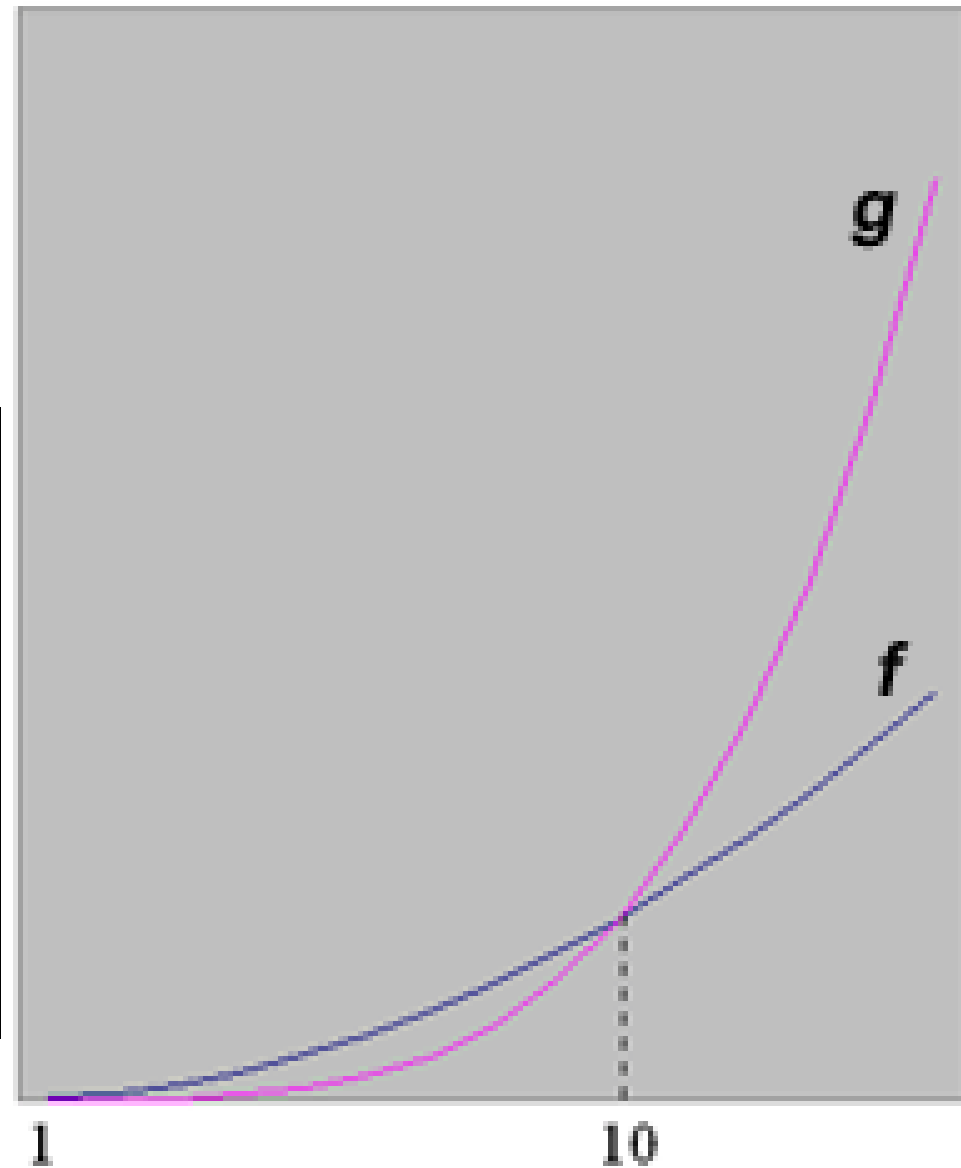
That is,  $A$  is a lower bound for  $B$   
and  $C$  is an upper bound for  $B$

$$C = \sum_{i=1}^n c_i$$

$$f(n) = 100 n^2$$

$$g(n) = n^4$$

$n$	$f(n)$	$g(n)$
10	10,000	10,000
50	250,000	6,250,000
100	1,000,000	100,000,000
150	2,250,000	506,250,000



Example from:

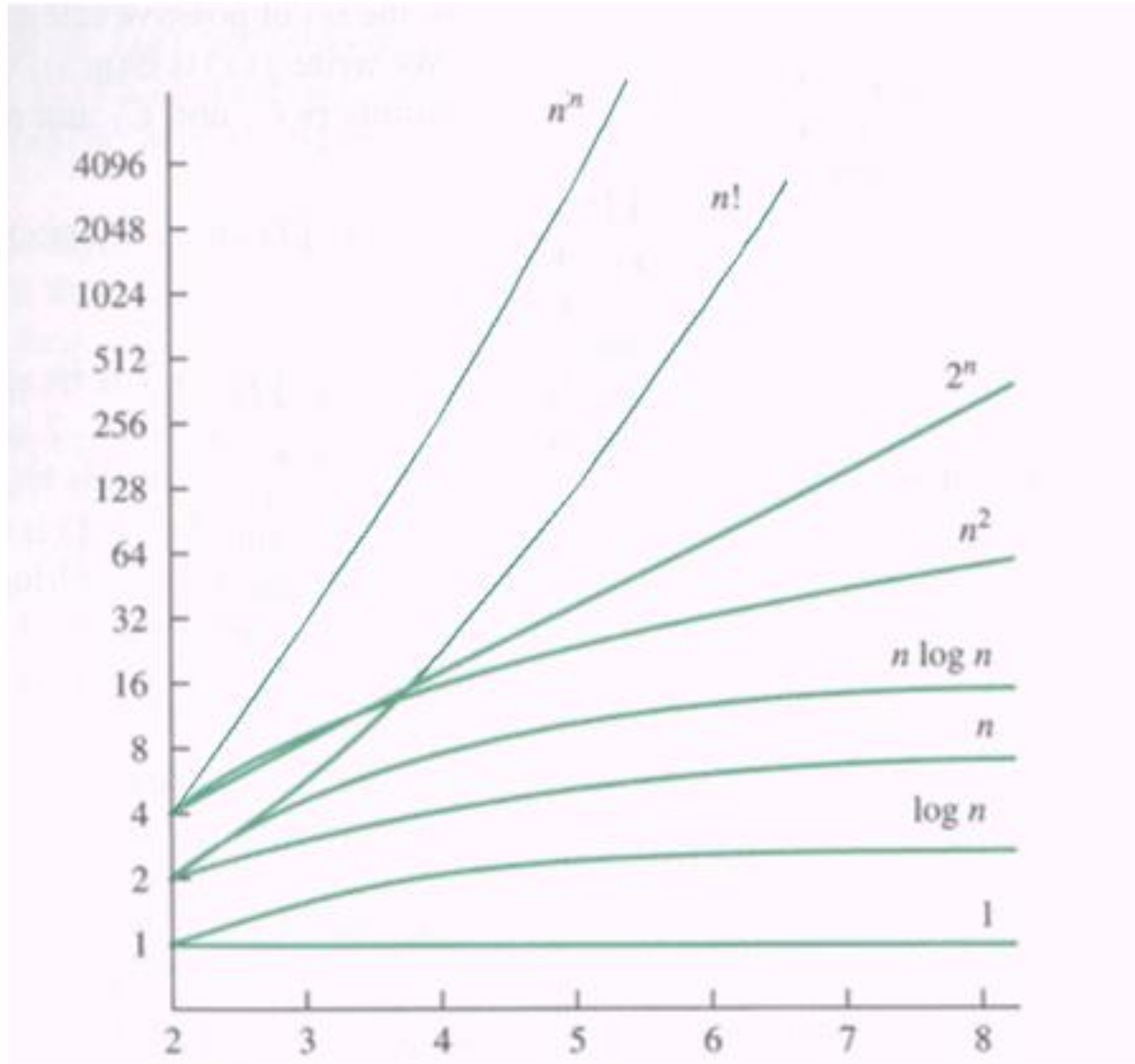
<http://www.cs.odu.edu/~toida/nerzic/content/function/growth.html>

Assume that  $T, f$  are functions mapping the natural numbers  $\{0, 1, 2, 3, \dots\}$  into the reals.

**Definition: "Big Oh"** A function  $T(n)$  is in  $O(f(n))$  if there exist constants  $n_0 \geq 0$ , and  $c > 0$ , such that for all  $n \geq n_0$ ,  $T(n) \leq c * f(n)$ .

Important: here I differ from older usage in defining  $O(f(n))$  to be a *set* of functions. This will prove useful later.

# Growth rates of functions



Big-Oh	Informal name
$O(1)$	constant
$O(\log n)$	logarithmic
$O(n)$	linear
$O(n \log n)$	$n \log n$
$O(n^2)$	quadratic
$O(n^3)$	cubic
$O(2^n)$	exponential
$O(n^c)$ for constant $c$ ,	polynomial time

Assume that  $T$ ,  $f$  and  $g$  are functions mapping the natural numbers  $\{0, 1, 2, 3, \dots\}$  into the reals.

**Definition: "Omega"** A function  $T(n)$  is in  $\Omega(f(n))$  if there exist constants  $n_0 \geq 0$ , and  $c > 0$ , such that for all  $n \geq n_0$ ,  $T(n) \geq c * f(n)$ .

**Definition: "Theta"** The set  $\Theta(g(n))$  of functions consists of  $\Omega(g(n)) \cap O(g(n))$ .



1. Prove that

$f(n) = 2 + n + 3n^2 + 5n^3$  is in

(a)  $O(n^3)$ ,

(b)  $\Omega(n^3)$ , and

(c)  $\Theta(n^3)$ .

2. Prove that  $-10 + 6n$  is in  $\Omega(n)$ .

Prove that

$k$

$$\sum_{i=0}^k 2^i$$

$$i=0$$

is in  $\theta(2^k)$ .

Getting a *tight (optimal)* estimate for the running time  $T(n)$  of an algorithm in the "Big Oh sense" means finding  $g(n)$  so that  $T(n)$  is in  $\Theta(g(n))$ .

To prove that an algorithm *for a problem* is *optimal* with respect to Big Oh analysis, you need to show:

1. The running time  $T(n)$  of the algorithm is in  $O(g(n))$  for some function  $g(n)$ , and
2. the running times for all algorithms under the given computational model must be in  $\Omega(g(n))$  for at least one input of size  $n$ .

# Logs

Logs arise often in CSC 225 as an artifact of divide and conquer algorithms.

## Definitions: Logarithms

For  $n = 2^k$ ,  $\log_2(n) = k$ .

For  $n = 10^k$ ,  $\log_{10}(n) = k$ .

In general: For  $n = c^k$ ,  $\log_c(n) = k$ .

Calculus- log conversion formula:

$$\log_b(x) = \log_c(x) / \log_c(b)$$

Theorem:  $\log_2(n) \in \Theta(\log_{10}(n))$

In CSC 225, the logs are generally  $\log_2$  but this shows in a Big Oh sense it does not matter what base it is for an expression like " $O(n \log n)$ ".

Theorem: The function

$$f(n) = 1 + 4n + 2n^2 + n^3$$

is not in the set  $O(n^2)$ .

How do we prove that  $f(n)$  is not in  $O(g(n))$ ?

Tactic: Proof by contradiction

To show that a statement  $S(n)$  is not true:

1. Assume that  $S(n)$  is true.
2. Apply valid mathematical operations.
3. Reach a conclusion that is obviously false.

Since the only thing done which is possibly mathematically invalid is to assume that  $S(n)$  is true,  $S(n)$  must be false.