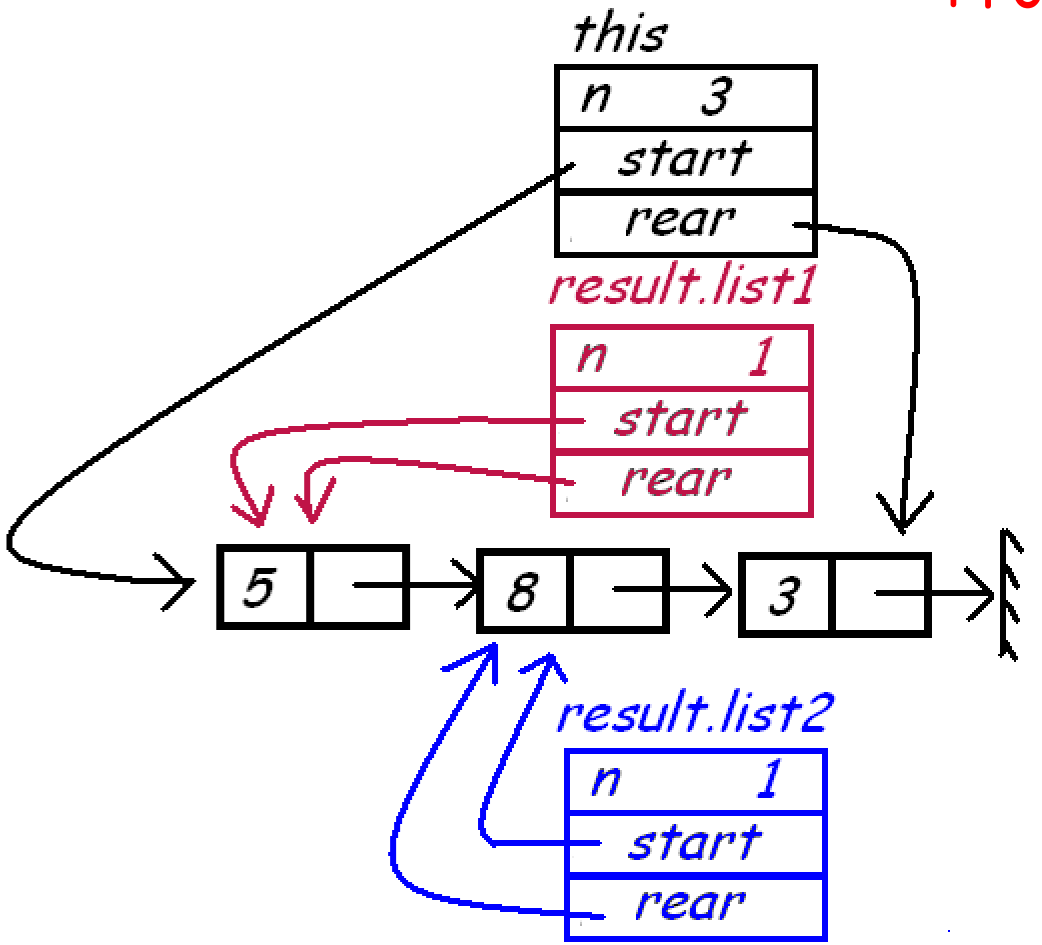
Problem of the day:



Execute the code on the next slide starting with this and determine what happens on this example.

```
while (result.list2.rear.next!= null )
{
  result.list1.rear.next= result.list2.rear.next;
  result.list1.rear= result.list2.rear.next;
  result.list1.n++;
  if (result.list1.rear.next != null)
  {
    result.list2.rear.next= result.list1.rear.next;
    result.list2.rear= result.list1.rear.next;
    result.list2.n++;
   }
}
```

```
result.list1.rear.next= null;
result.list2.rear.next= null;
return(result);
```

```
public SplitList evenOddSplit()
{   SplitList result;
    boolean done;
    result= new SplitList();
    if (start== null) return(result);
    result.list1.start= start;
    result.list1.rear = start;
    result.list1.n++;

    if (start.next == null) return(result);
    result.list2.start= start.next;
    result.list2.rear = start.next;
    result.list2.n++;
    done= false;
```

Corrected code

```
while (result.list2.rear.next!= null && ! done)
{
  result.list1.rear.next= result.list2.rear.next;
  result.list1.rear= result.list2.rear.next;
  result.list1.n++;
  if (result.list1.rear.next != null)
  {
     result.list2.rear.next= result.list1.rear.next;
     result.list2.rear= result.list1.rear.next;
     result.list2.n++;
   }
   else done= true;
}
```

```
result.list1.rear.next= null;
result.list2.rear.next= null;
return(result);
```

Assignment #1 A is due Fri. at the beginning of class, part B is due Tues. at midnight.

Recall that you need a 50% assignment average. It is better to hand in your best effort than to hand in nothing.

Any questions about the assignment?

Office hours this week:

Tues. 12:30 or 2:30
Wed. 12:30 or 1:30
Fri. 12:30 or 2:30

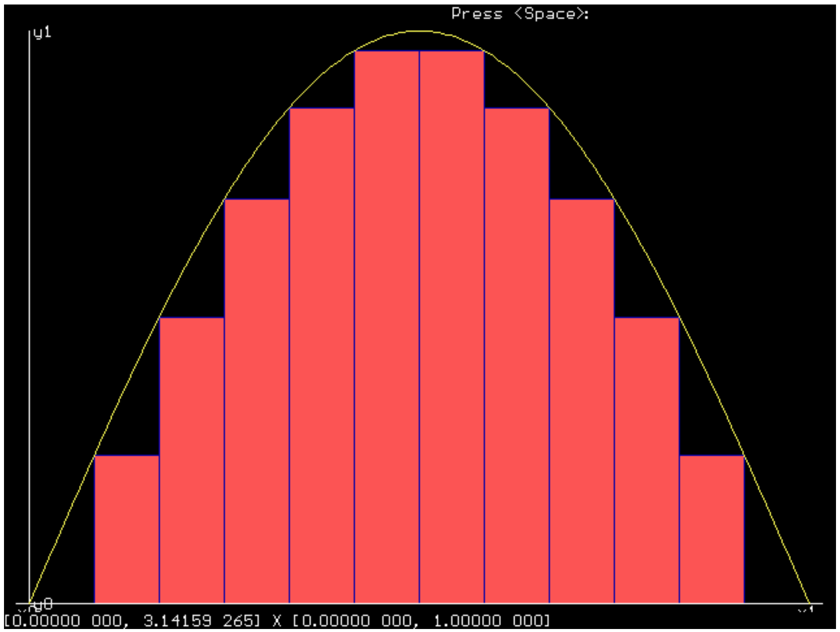
Appointments are possible on Thursday but only if you have classes at the other times.

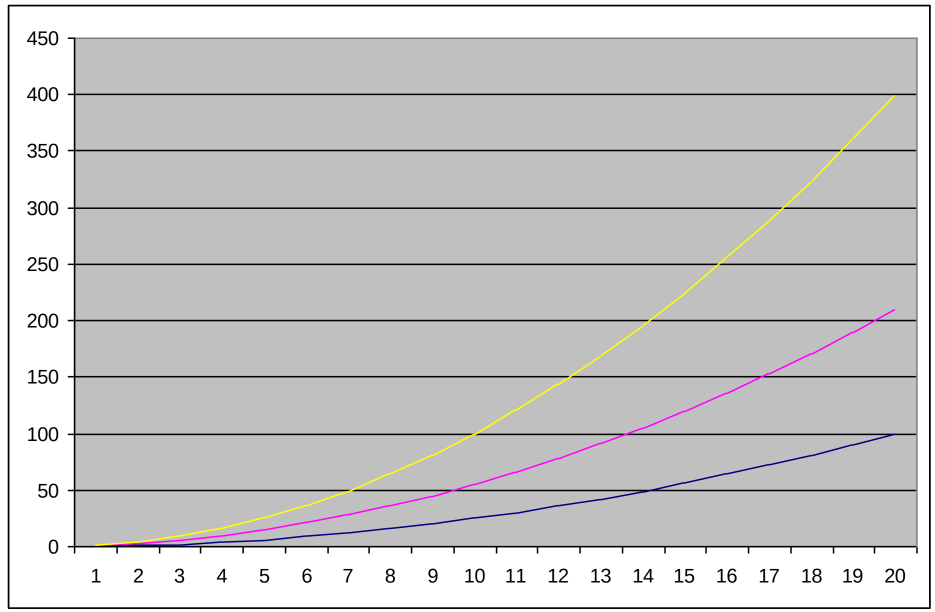
Please let me know if you plan to come by.

Small typo on 1A:

Question 4 refers to question 3 not question 2.

Mathematics of Algorithm Analysis

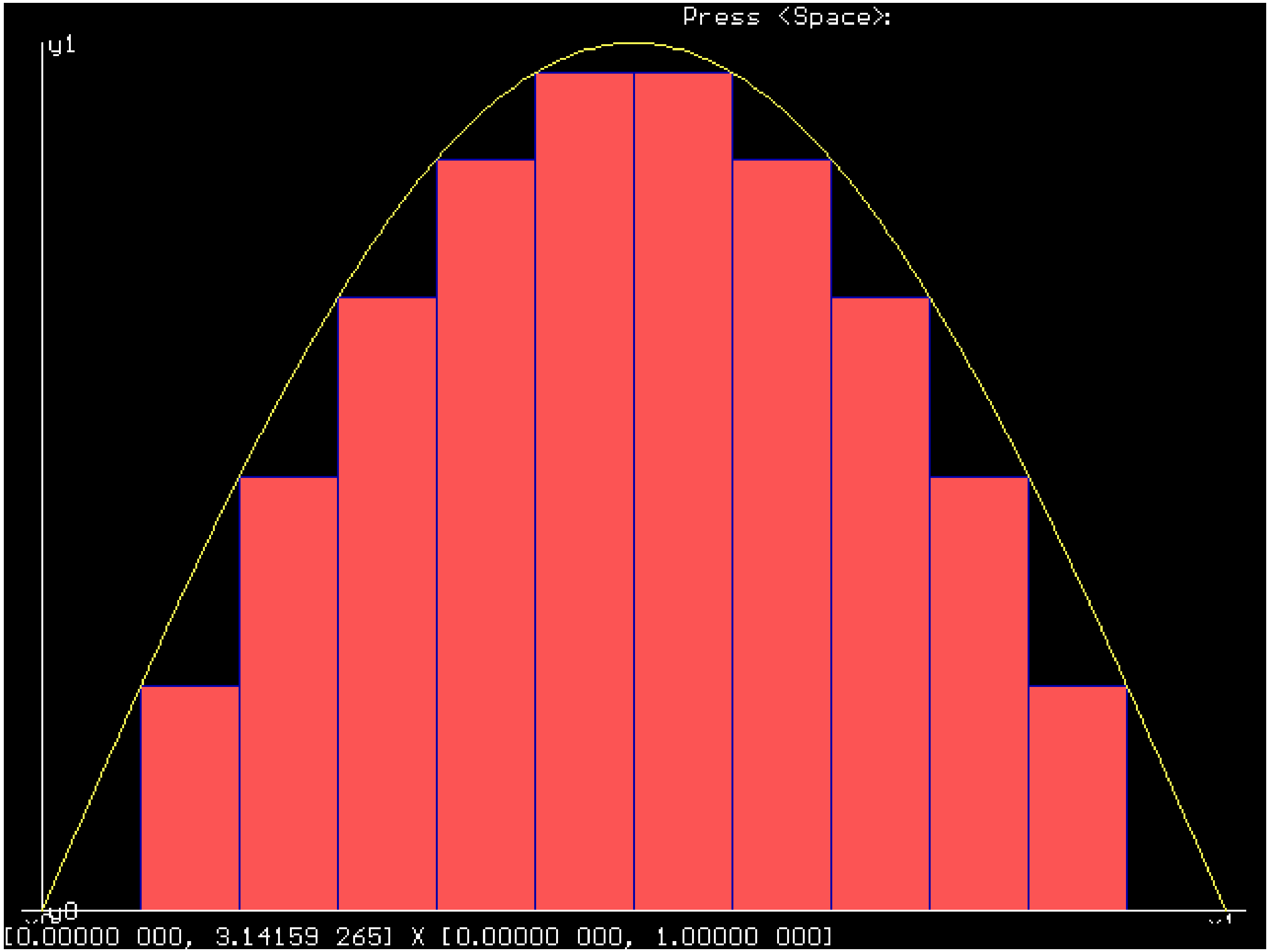




Outline

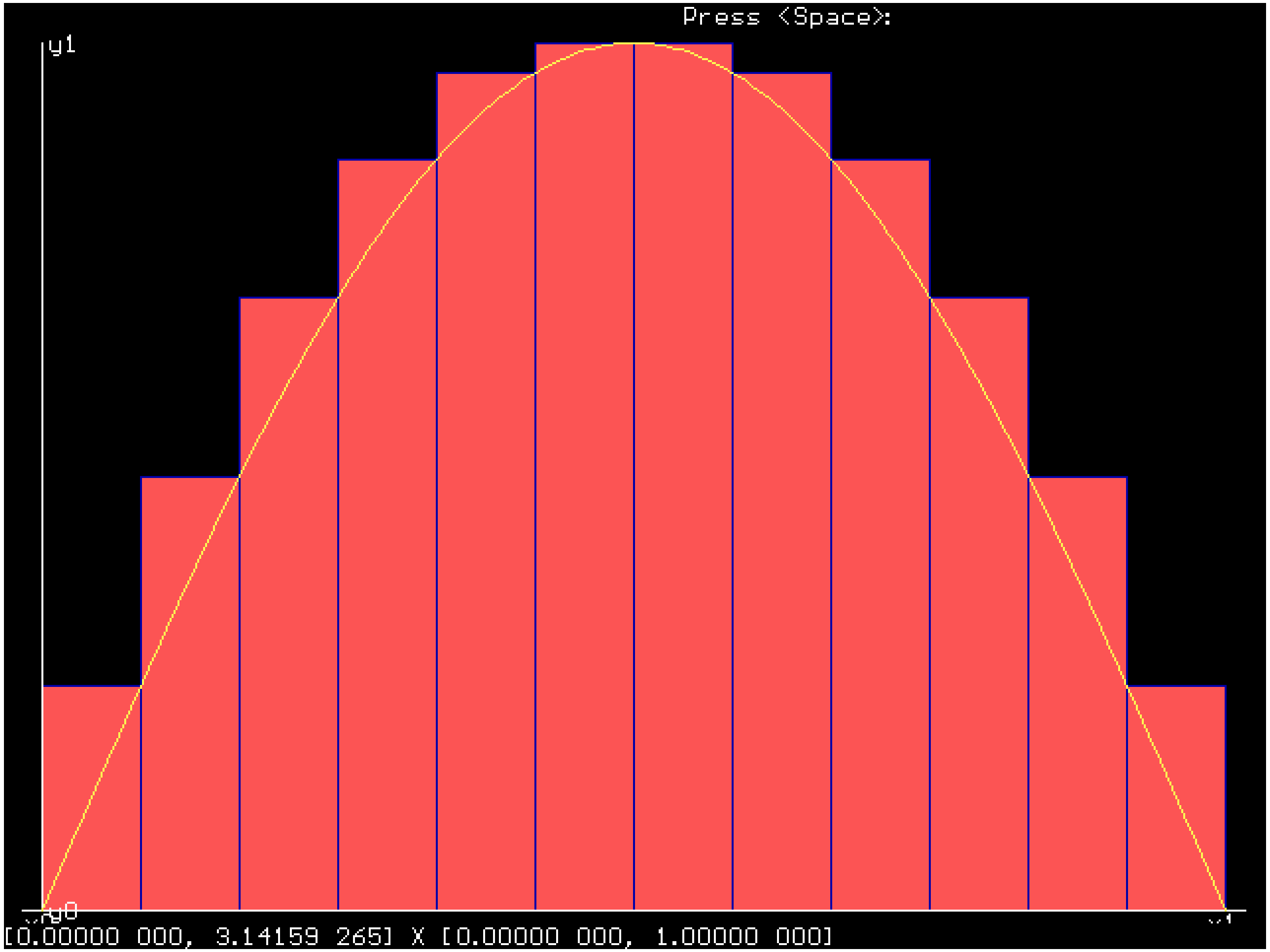
- Lower bounds and upper bounds on functions.
- Terminology for talking about the amount of time or space that an algorithm uses.

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



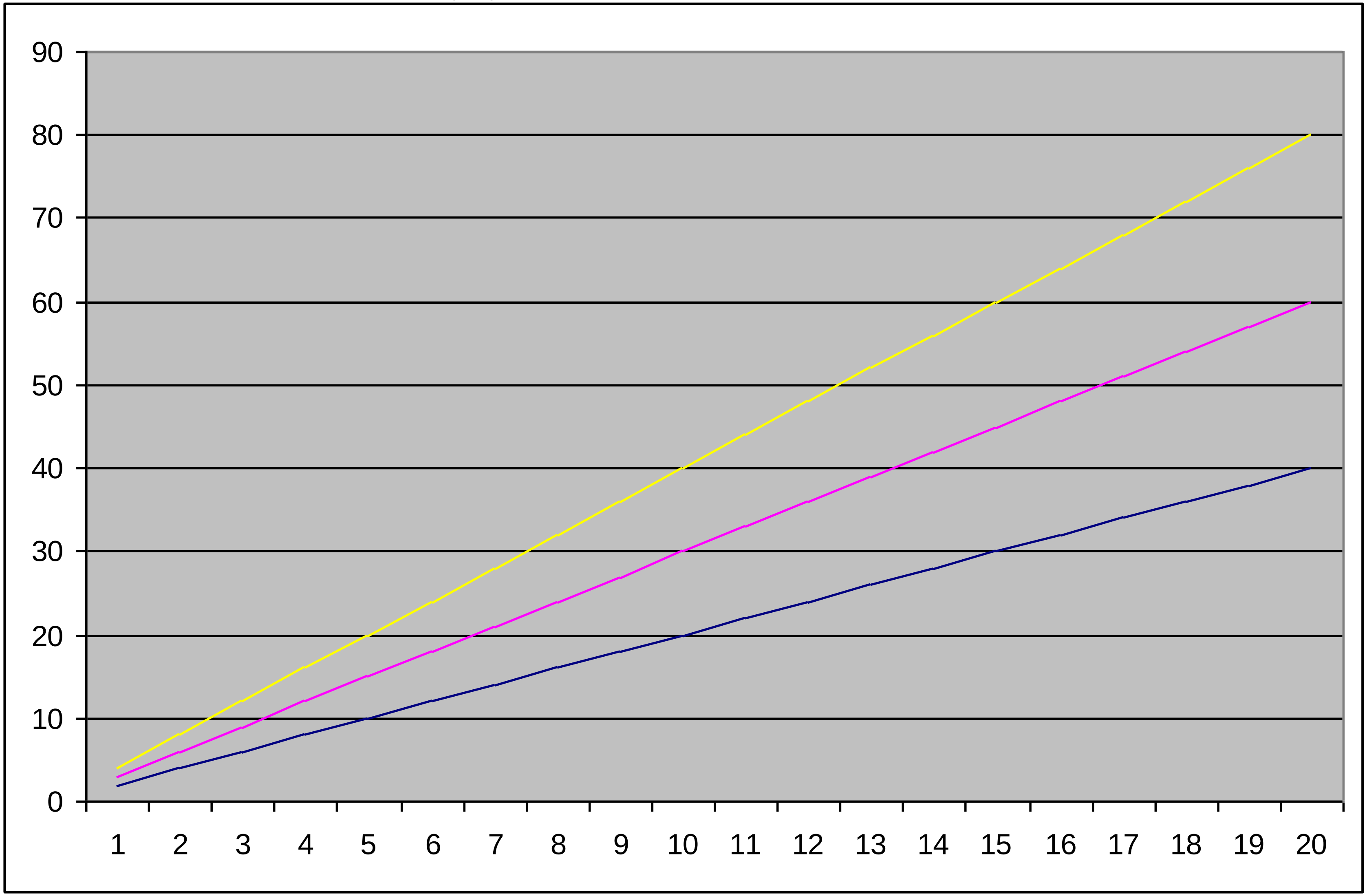
Picture from
http://archives.math.utk.edu/visual.calculus/4/riemann_sums.3/microcalc.html

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



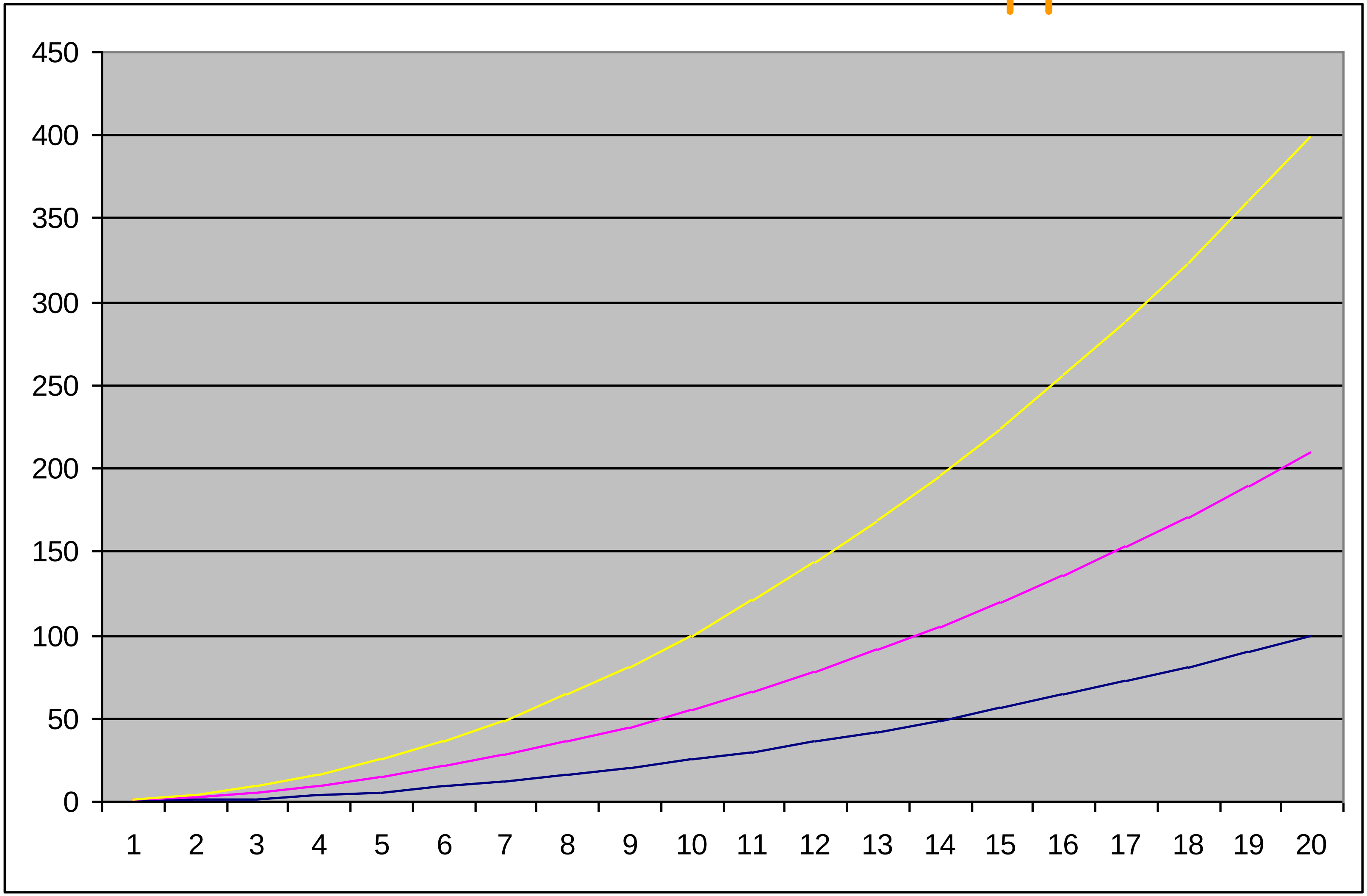
Picture from
http://archives.math.utk.edu/visual.calculus/4/riemann_sums.3/microcalc.html

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



$n^2/4 \le n(n+1)/2 \le n^2$ for $n \ge 0$.

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, $f(x) \le g(x)$.

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

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:

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

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, … , n.

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

If $a_i \leq b_i \leq c_i$ for i= 1, 2, 3, … n

then $A \leq B \leq C$.

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

That is, A is a lower bound for B

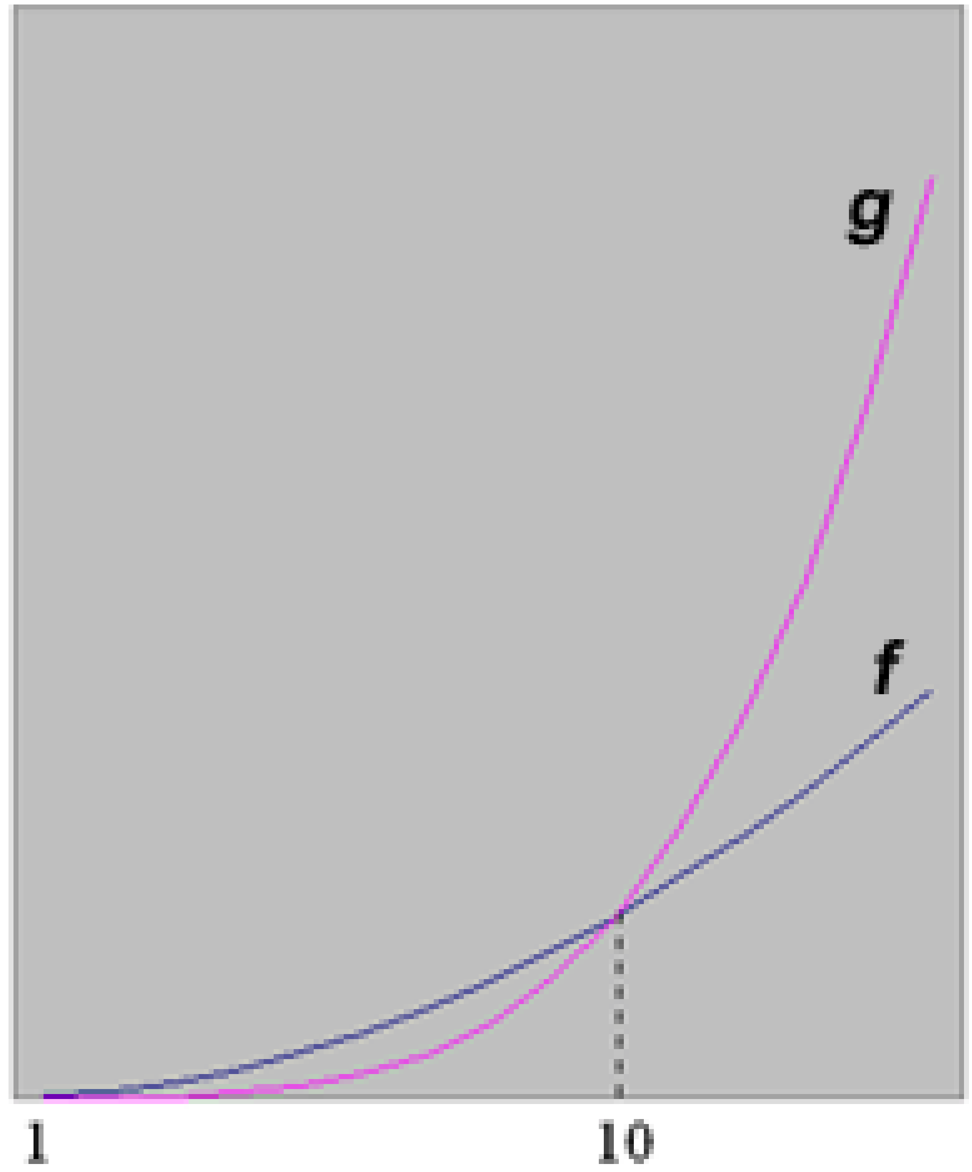
and C is an upper bound for B

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

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

$g(n) = n^4$

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:

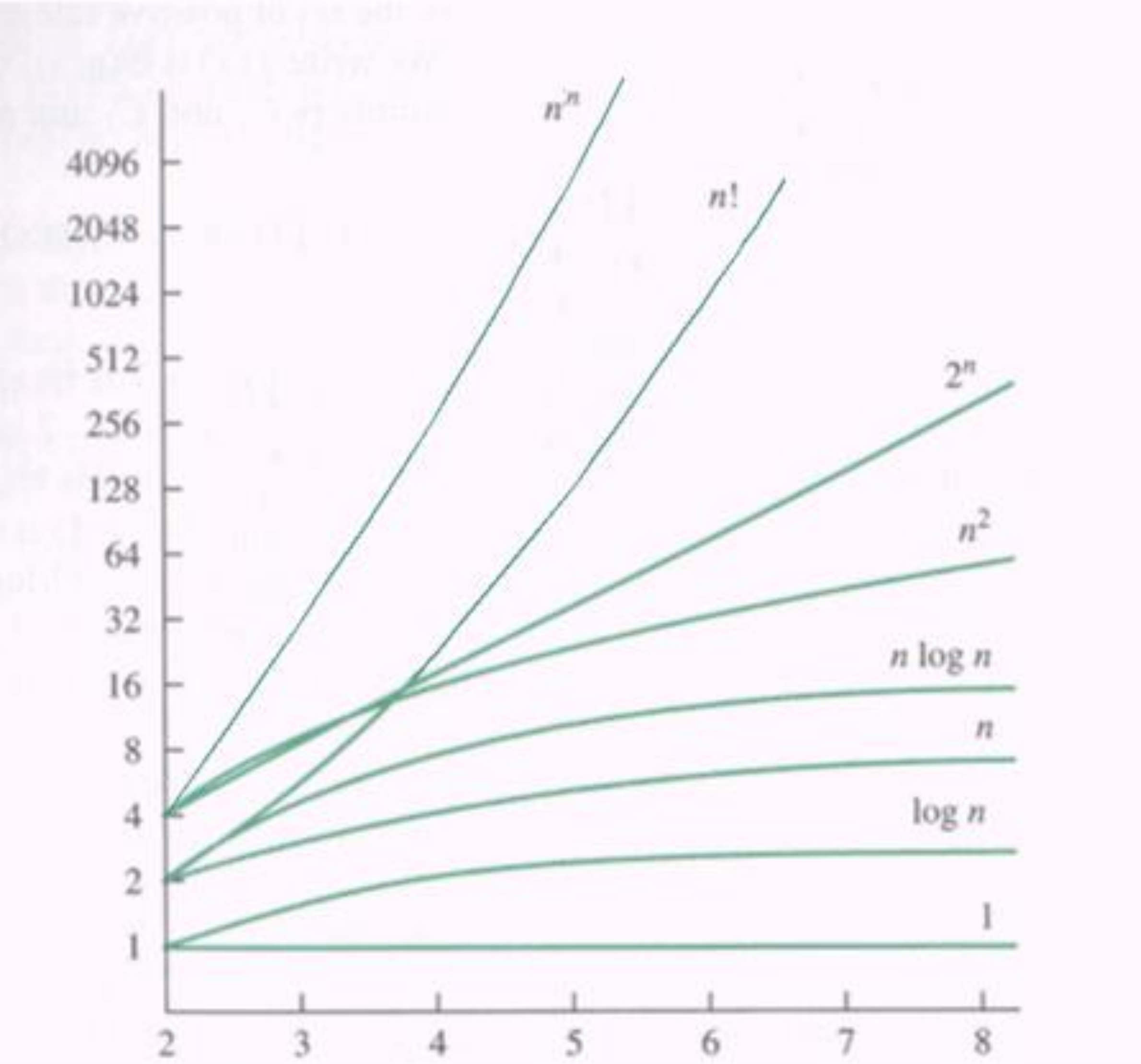
http://www.cs.odu.edu/~toida/nerzic/content/function/growth.html

Assume that T, f are functions mapping the natural numbers {0, 1, 2, 3, ...} 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



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

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, ...} 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

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

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 runnning 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

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

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.