

## Analyzing Algorithms

Algorithms are the only important, durable, and original part of computer science because they can be studied in a machine and language independent way.

How can we study the time complexity of an algorithm if we don't choose a specific machine to measure it on?

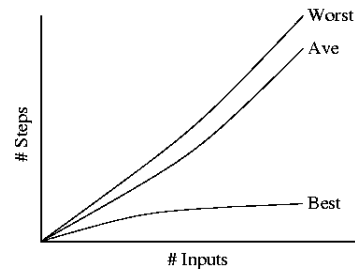
## The RAM Model

- Each "simple" operation (+, -, =, if, call) takes exactly 1 step.
- Loops and subroutine calls are not simple operations, but depend upon the size of the data and the contents of a subroutine. We do not want "sort" to be a single step operation.
- Each memory access takes exactly 1 step.

## Measuring Complexity

- The *worst case complexity* of the algorithm is the function defined by the maximum number of steps taken on any instance of size  $n$ .
- The *best case complexity* of the algorithm is the function defined by the minimum number of steps taken on any instance of size  $n$ .
- The *average-case complexity* of the algorithm is the function defined by an average number of steps taken on any instance of size  $n$ .

## Best, Worst, and Average-Case



## Insertion Sort

One way to sort an array of  $n$  elements is to start with empty list, then successively insert new elements in the proper position:

$$a_1 \leq a_2 \leq \dots \leq a_k \mid a_{k+1} \leq \dots \leq a_n$$

At each stage, the inserted element leaves a sorted list, and after  $n$  insertions contains exactly the right elements. Thus the algorithm must be correct.

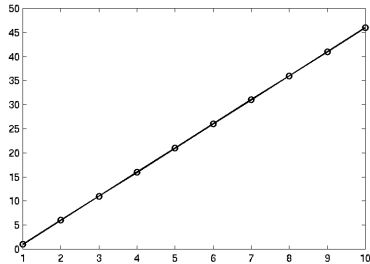
But how *efficient* is it?

## Exact Analysis

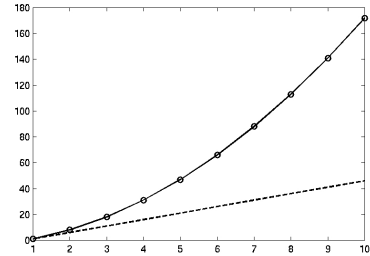
Count the number of times each line will be executed:

	Num Exec.
for $i = 2$ to $n$	$(n-1) + 1$
$key = A[i]$	$n-1$
$j = i - 1$	$n-1$
while $j > 0$ AND $A[j] > key$	?
$A[j+1] = A[j]$	?
$j = j - 1$	?
$A[j+1] = key$	$n-1$

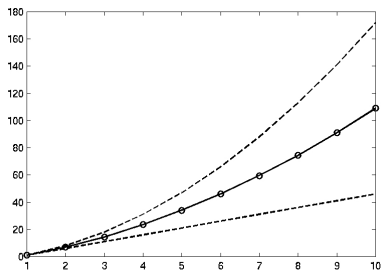
### Best Case



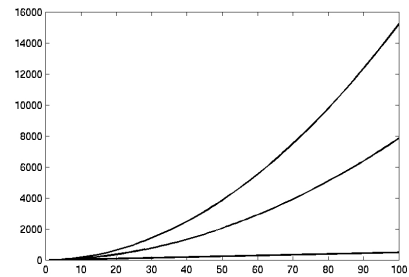
### Worst Case



### Average Case



### Average Case (Zoom Out)



```

1: for i = 2 to n
2:   key = A[i]
3:   j = i - 1
4:   while j > 0 AND A[j] > key
5:     A[j+1] = A[j]
6:     j = j - 1
7:   A[j+1] = key

[3] [7] [4] [9] [4]
1: i = 2
2: key = 7
3: j = 1
4: ( while false! )
7: A[2] = 7
[3] [7] [4] [9] [4]

[3] [7] [4] [9] [4]
1: i = 3
2: key = 4
3: j = 2
4: (while true!)
5: A[3] = A[2]
[3] [7] [7] [9] [4]
6: j = 1
4: (while false!)
7: A[2] = 4
[3] [4] [7] [9] [4]
1: i = 4
2: key = 9
3: j = 3
4: (while false!)
7: A[4] = 9

[3] [4] [7] [9] [4]
1: i = 5
2: key = 4
3: j = 4
4: (while true!)
5: A[5] = A[4]
[3] [4] [7] [9] [9]
6: j = 3
4: (while true!)
5: A[4] = A[3]
[3] [4] [7] [7] [9]
6: j = 2
4: (while false!)
7: A[3] = 4

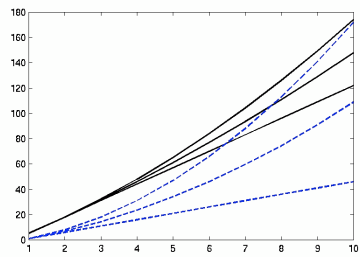
[3] [4] [4] [7] [9]
1: i = 4
2: key = 9
3: j = 3
4: (while false!)
7: A[4] = 9
    
```

## Divide & Conquer

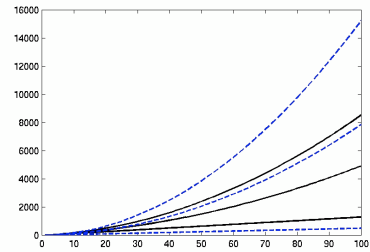
- **Divide** the problem into smaller problems, often even if they are all the same.
- **Conquer** the individual pieces, recursively if they are just smaller versions of the main problem.
- **Combine** the results into a solution for the main problem.

### Divided Insertion Sort

Sort first half, sort second half, and then merge!

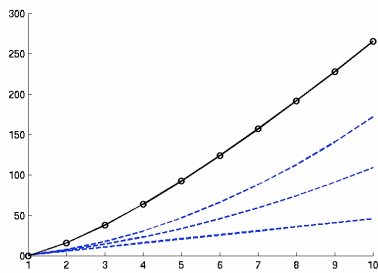


### Divided Insertion Sort

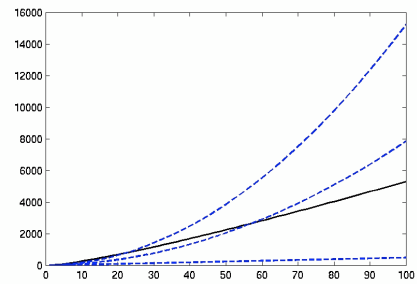


Can we do better?

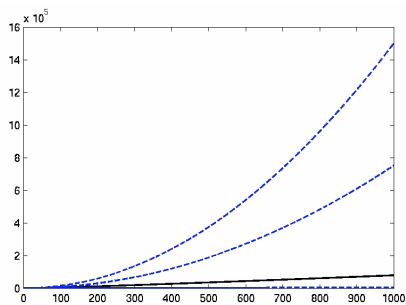
### Merge Sort



### Merge Sort



### Merge Sort



### Measuring Complexity

What is the best way to measure the time complexity of an algorithm?

- Best-case run time?
- Worst-case run time?
- Average run time?

Which should we try to optimize?

## Best-Case Measures

*How can we modify almost any algorithm to have a good best-case running time?*

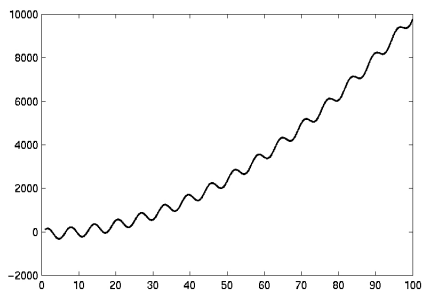
Solve **one** instance of it efficiently.

## Worst Case

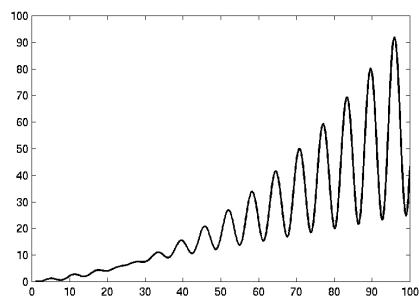
*What inputs will cause our algorithms the most difficulty?*

Worst case is relatively easy to find, and gives us a guarantee on the upper bound of how much time our algorithm will require.

## Exact Analysis is Hard!



## Even Harder Exact Analysis



## Bounding Functions

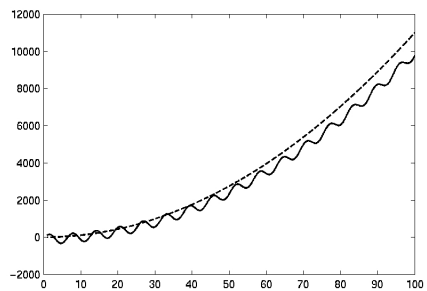
$g(n) = O(f(n))$  means  $C \times f(n)$  is an *Upper Bound* on  $g(n)$

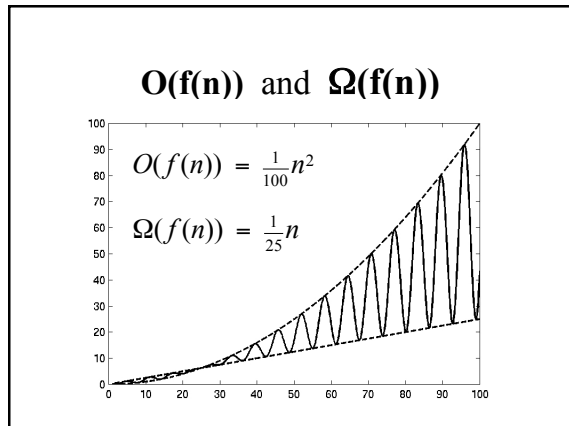
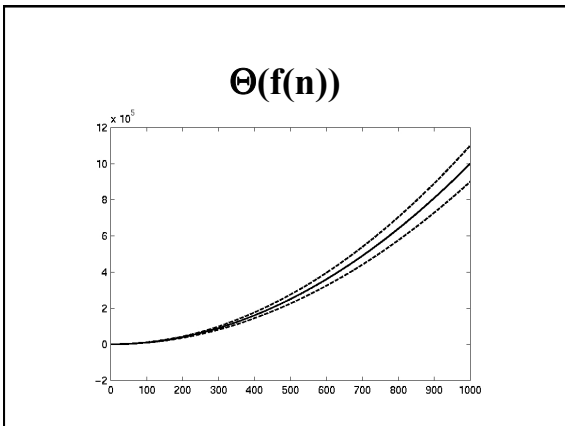
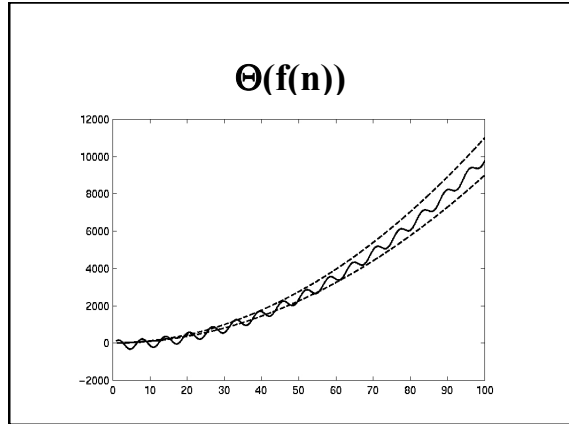
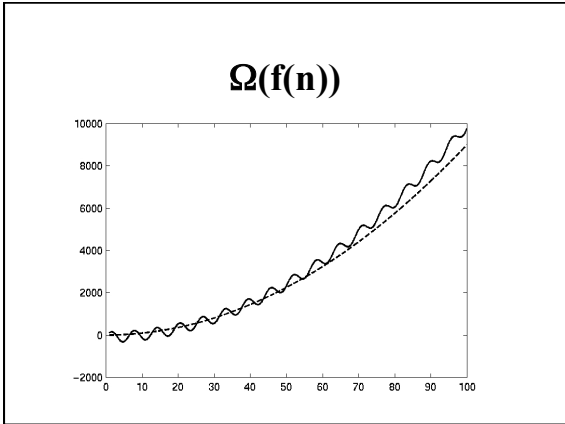
$g(n) = \Omega(f(n))$  means  $C \times f(n)$  is a *Lower Bound* on  $g(n)$

$g(n) = \Theta(f(n))$  means  $C_1 \times f(n)$  is an *Upper Bound* on  $g(n)$   
and  $C_2 \times f(n)$  is a *Lower Bound* on  $g(n)$

These bounds hold for all inputs beyond some threshold  $n_0$ .

## $O(f(n))$





### Example Function

$$f(n) = 3n^2 - 100n + 6$$

### What does all this mean?

$3n^2 - 100n + 6 = O(n^2)$  because  $3n^2 > 3n^2 - 100n + 6$   
 $3n^2 - 100n + 6 = O(n^3)$  because  $0.0001n^3 > 3n^2 - 100n + 6$   
 $3n^2 - 100n + 6 \neq O(n)$  because  $c \times n < 3n^2$  when  $n > c$

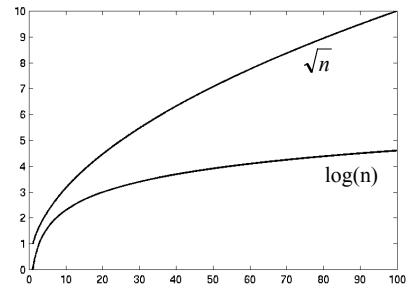
$3n^2 - 100n + 6 = \Omega(n^2)$  because  $2.99n^2 < 3n^2 - 100n + 6$   
 $3n^2 - 100n + 6 \neq \Omega(n^3)$  because  $3n^2 - 100n + 6 < n^3$   
 $3n^2 - 100n + 6 = \Omega(n)$  because  $10^{10}! n < 3n^2 - 100n + 6$

$3n^2 - 100n + 6 = \Theta(n^2)$  because both  $O$  and  $\Omega$   
 $3n^2 - 100n + 6 \neq \Theta(n^3)$  because  $O$  only  
 $3n^2 - 100n + 6 \neq \Theta(n)$  because  $\Omega$  only

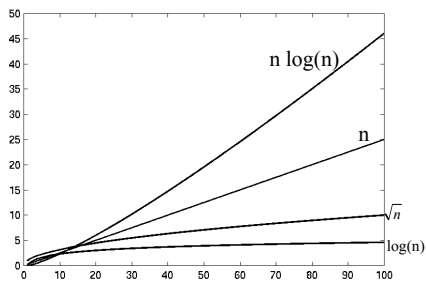
## Common Complexities

Complexity	10	20	30	40	50	60
$n$	$1 \times 10^{-5}$ sec	$2 \times 10^{-5}$ sec	$3 \times 10^{-5}$ sec	$4 \times 10^{-5}$ sec	$5 \times 10^{-5}$ sec	$6 \times 10^{-5}$ sec
$n^2$	0.0001 sec	0.0004 sec	0.0009 sec	0.016 sec	0.025 sec	0.036 sec
$n^3$	0.001 sec	0.008 sec	0.027 sec	0.064 sec	0.125 sec	0.216 sec
$n^5$	0.1 sec	3.2 sec	24.3 sec	1.7 min	5.2 min	13.0 min
$2^n$	0.001sec	1.0 sec	17.9 min	12.7 days	35.7 years	366 cent
$3^n$	0.59sec	58 min	6.5 years	3855 cent	$2 \times 10^8$ cent	$1.3 \times 10^{13}$ cent
$\log_2 n$	$3 \times 10^{-6}$ sec	$4 \times 10^{-6}$ sec	$5 \times 10^{-6}$ sec	$5 \times 10^{-6}$ sec	$6 \times 10^{-6}$ sec	$6 \times 10^{-6}$ sec
$n \log_2 n$	$3 \times 10^{-5}$ sec	$9 \times 10^{-5}$ sec	0.0001 sec	0.0002 sec	0.0003 sec	0.0004 sec

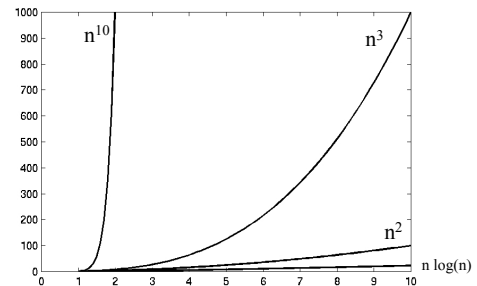
## Complexity Graphs



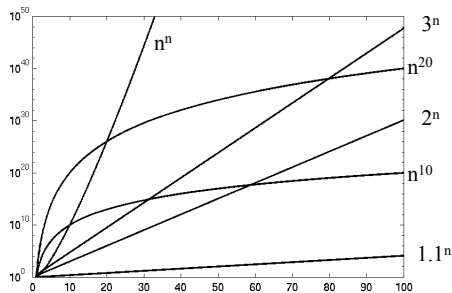
## Complexity Graphs



## Complexity Graphs



## Complexity Graphs (log scale)



## Analyzing Insertion Sort

for i = 2 to n	Num Exec.
key = A[i]	(n-1) + 1
j = i - 1	n-1
while j > 0 AND A[j] > key	n-1
A[j+1] = A[j]	n-1 to n(n-1)/2+n-1
j = j - 1	0 to n(n-1)/2
A[j+1] = key	0 to n(n-1)/2
	n-1

Worst Case Complexity:  $\frac{3}{2}n^2 + \frac{7}{2}n - 4$

## Logarithms

Properties:

$$b^x = y \equiv x = \log_b y$$

$$b^{\log_b x} = x$$

$$\log_a b = b \log a$$

$$\log_a x = c \log_b x \quad (\text{where } c = 1/\log_b a)$$

Questions:

- \* How do  $\log_a n$  and  $\log_b n$  compare?
- \* How can we compare  $n \log n$  with  $n^2$ ?

## Example Problems

1. What does it mean if:  
 $f(n) \neq O(g(n))$  and  $g(n) \neq O(f(n))$  ???
2. Is  $2^{n+1} = O(2^n)$  ?  
Is  $2^{2n} = O(2^n)$  ?
3. Does  $f(n) = O(f(n))$  ?
4. If  $f(n) = O(g(n))$  and  $g(n) = O(h(n))$ ,  
can we say  $f(n) = O(h(n))$  ?

## Order-of-Magnitude Problems

**Easy:** What is the total mass of food an average person eats in their lifetime?

**Harder:** How many bricks are there on the MSU campus?

**Very Hard:** What is the probability of Sentient life elsewhere in the universe?