CMP302: Algorithms

Lecture 01: Insertion Sort

Mohamed Alaa El-Dien Aly
Computer Engineering Department
Cairo University
Fall 2013

Agenda

- Analysis of Algorithms
- Insertion Sort
- Asymptotic Analysis

Acknowledgment
A lot of slides adapted from the slides of Erik Demaine and Charles Leiserson

Algorithms

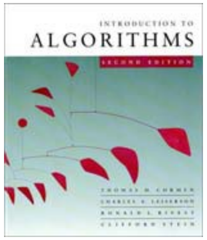
Algorithm

A computational procedure that takes some values as input and produces some values as output

Algorithm Analysis

Determining the *resources* required by the algorithm as a function of the input size.

Resources include *space* and *time*.



Why study algorithms and performance?

- Algorithms help us to understand ***scalability***.
- Performance often draws the line between what is feasible and what is impossible.
- Algorithmic mathematics provides a ***language*** for talking about program behavior.
- Performance is the ***currency*** of computing.
- The lessons of program performance generalize to other computing resources.
- Speed is fun!

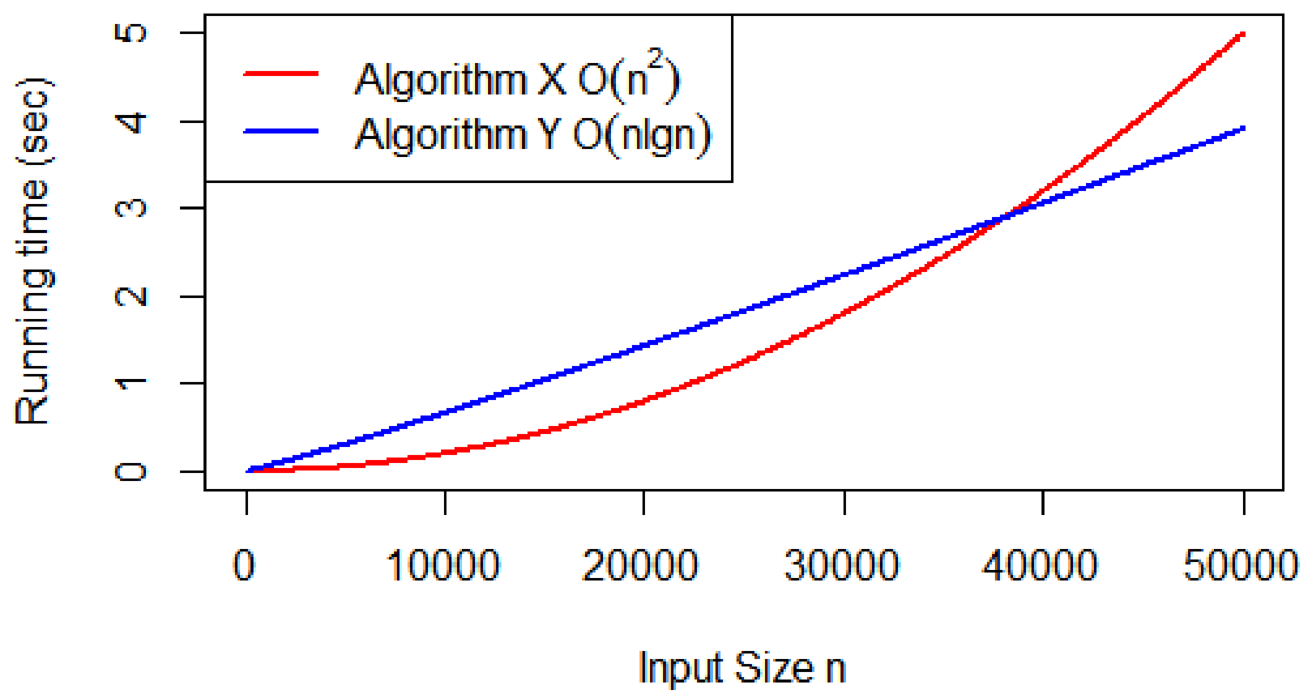
Running time

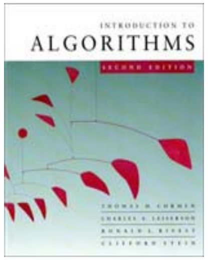
- Assume:
  - Algorithm X takes time $2n^2$ (written by best programmer) running on machine with 1000 MIPS
  - Algorithm Y takes time $50n \lg n$ (written by worst programmer) running on machine with 10 MIPS
- Running time for 10^6 numbers
  - Algorithm X takes 2000 seconds
  - Algorithm Y takes ~100 seconds

Complexity makes a huge difference!

Running time

- Assume:
  - Algorithm X takes time $2n^2$ (written by best programmer) running on machine with 1000 MIPS
  - Algorithm Y takes time $50n \lg n$ (written by worst programmer) running on machine with 10 MIPS





The problem of sorting

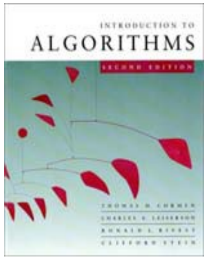
Input: sequence $\langle a_1, a_2, \ldots, a_n \rangle$ of numbers.

Output: permutation $\langle a'_1, a'_2, \ldots, a'_n \rangle$ such that $a'_1 \le a'_2 \le \cdots \le a'_n$.

Example:

Input: 8 2 4 9 3 6

Output: 2 3 4 6 8 9



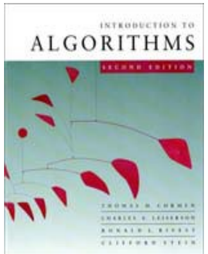
Insertion sort

"pseudocode"

```
INSERTION-SORT (A, n)        ▷ A[1 . . n]
    for j ← 2 to n
        do key ← A[ j]
           i ← j – 1
           while i > 0 and A[i] > key
               do A[i+1] ← A[i]
                  i ← i – 1
           A[i+1] = key
```

Insertion Sort

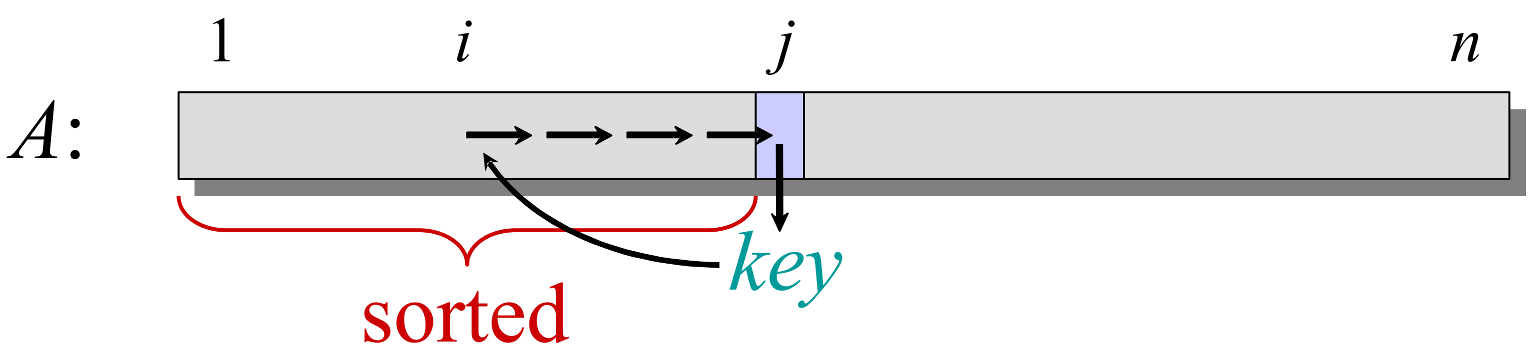
```
void insertion_sort(vector<int>& A) {
  for (int j = 1; j < A.size(); ++j) {
    int key = A[j];
    int i = j - 1;
   for (; i >= 0 && A[i] > key;) {
      A[i+1] = A[i--];
    }
    A[i+1] = key;
  }
}
```

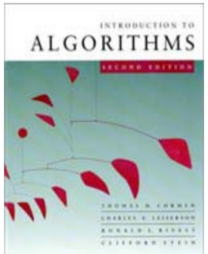


Insertion sort

"pseudocode"

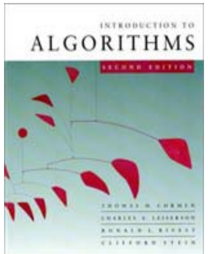
```
INSERTION-SORT (A, n)      ▷ A[1 . . n]
   for j ← 2 to n
        do key ← A[ j]
           i ← j – 1
           while i > 0 and A[i] > key
                do A[i+1] ← A[i]
                   i ← i – 1
           A[i+1] = key
```





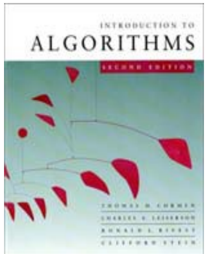
Example of insertion sort

8 2 4 9 3 6



Example of insertion sort

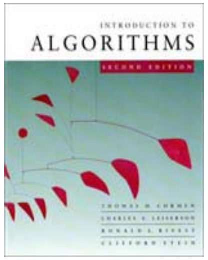
8 2 4 9 3 6



Example of insertion sort

8 2 4 9 3 6

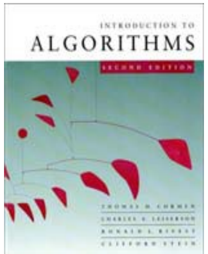
2 8 4 9 3 6



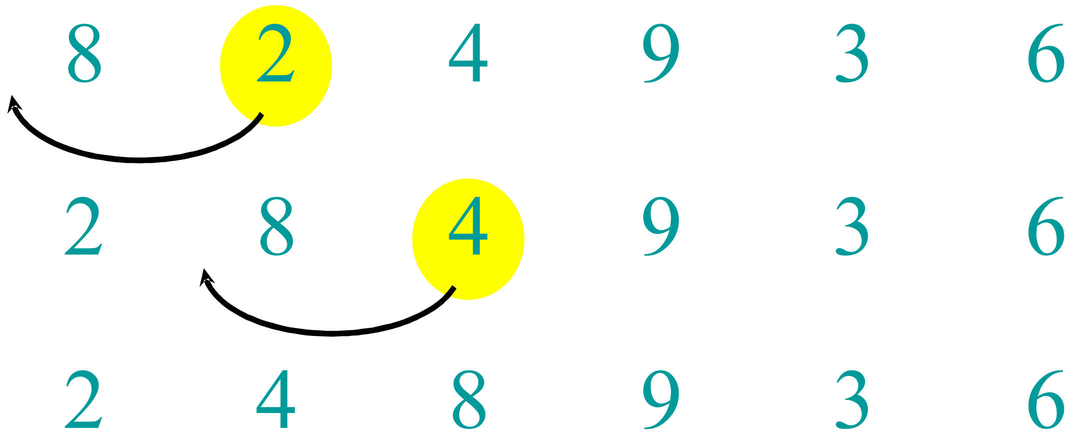
Example of insertion sort

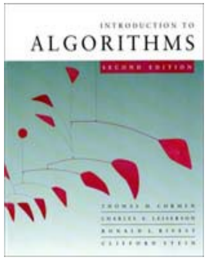
8 2 4 9 3 6

2 8 4 9 3 6

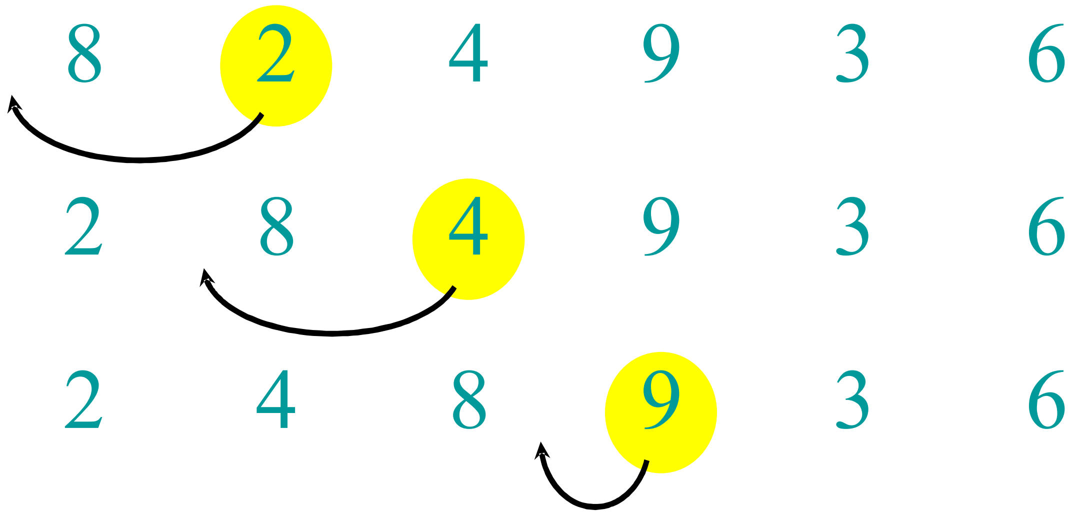


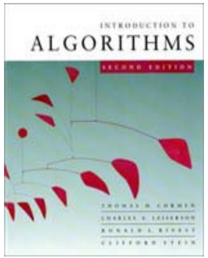
Example of insertion sort





Example of insertion sort





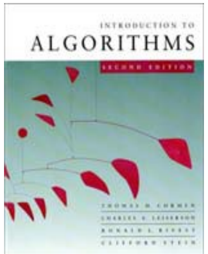
Example of insertion sort

8 2 4 9 3 6

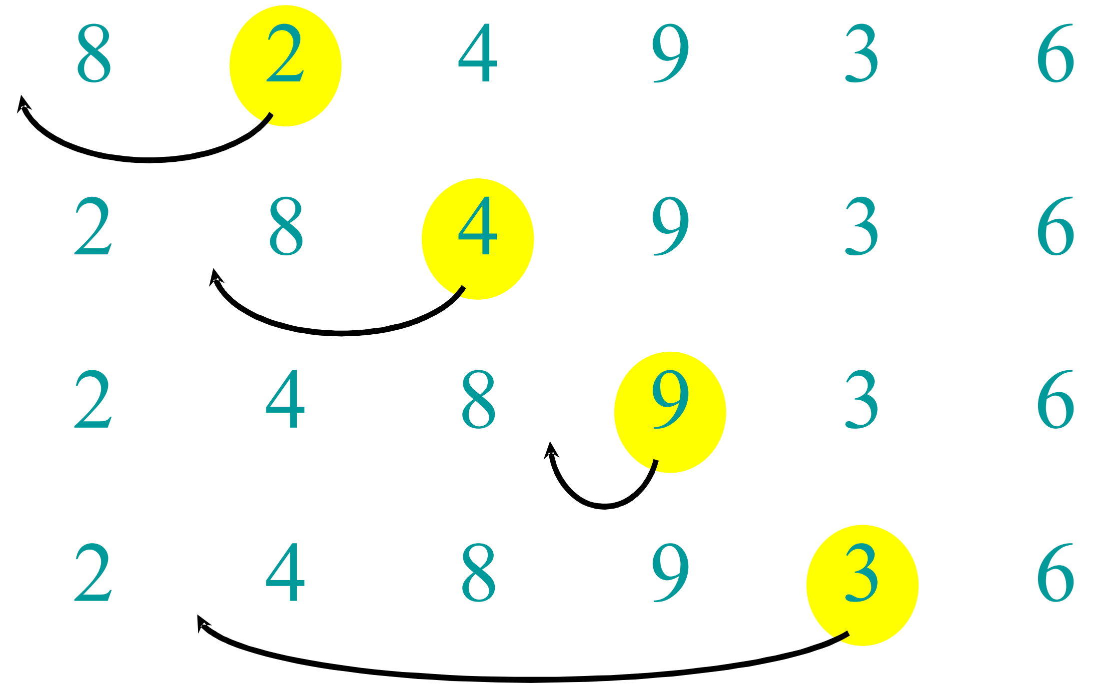
2 8 4 9 3 6

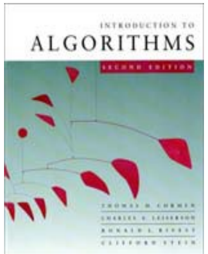
2 4 8 9 3 6

2 4 8 9 3 6

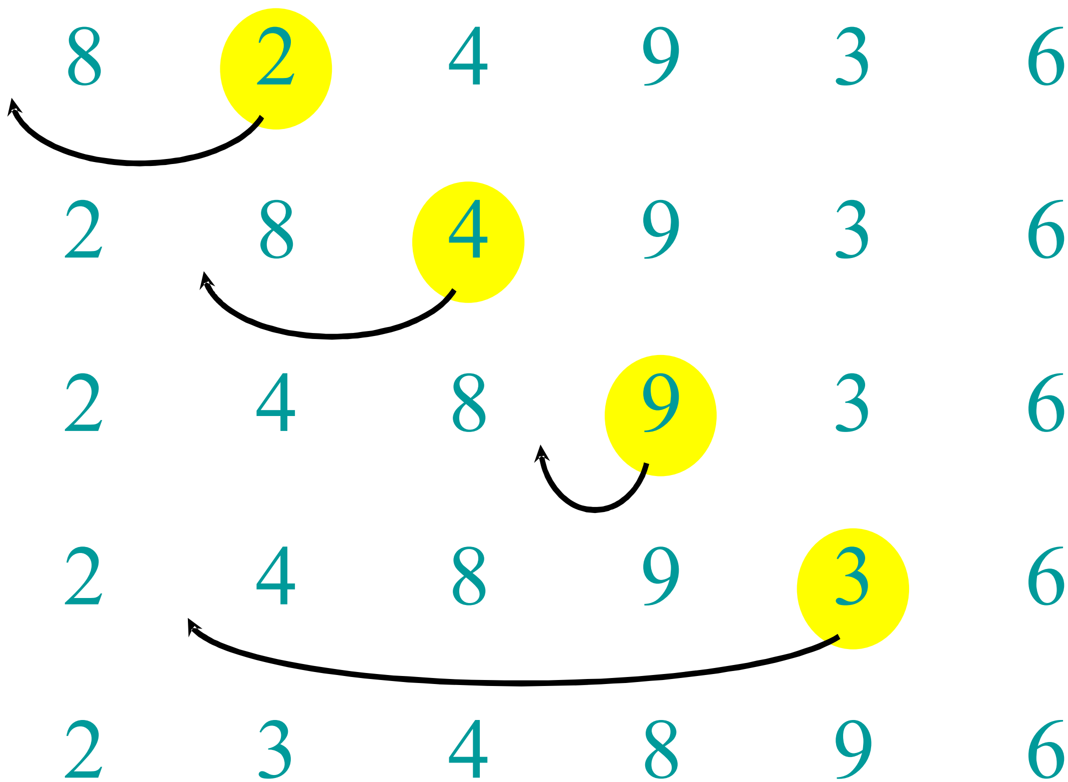


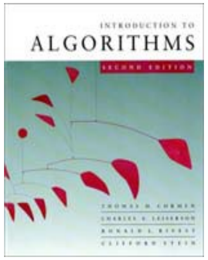
Example of insertion sort



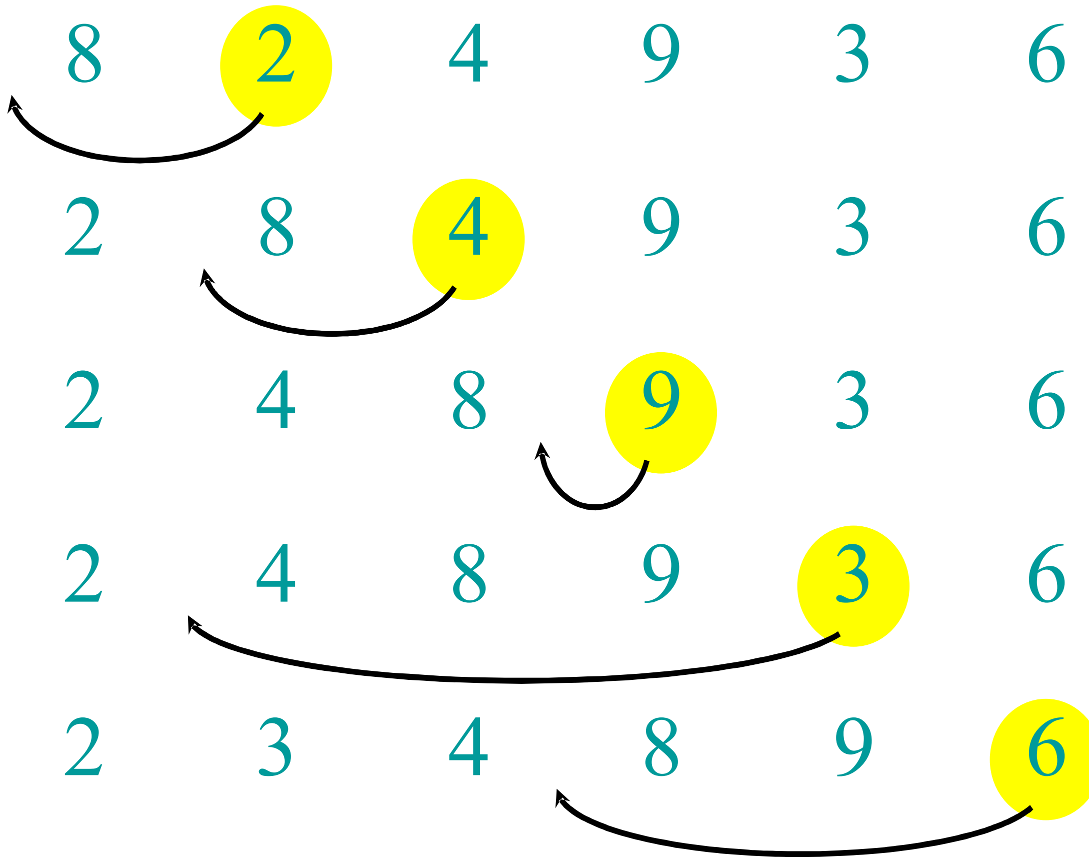


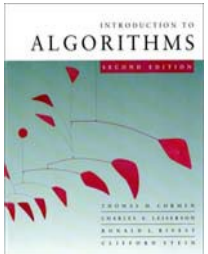
Example of insertion sort





Example of insertion sort





Example of insertion sort

8 2 4 9 3 6

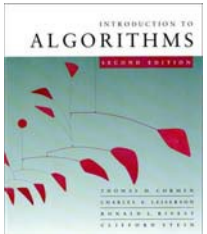
2 8 4 9 3 6

2 4 8 9 3 6

2 4 8 9 3 6

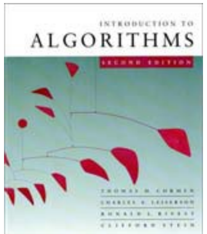
2 3 4 8 9 6

2 3 4 6 8 9 *done*



Running time

- The running time depends on the input: an already sorted sequence is easier to sort.
- Parameterize the running time by the size of the input, since short sequences are easier to sort than long ones.
- Generally, we seek upper bounds on the running time, because everybody likes a guarantee.



Kinds of analyses

Worst-case: (usually)

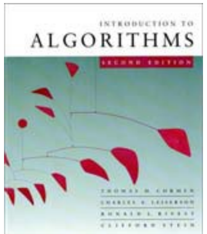
- $T(n)$ = maximum time of algorithm on any input of size n.

Average-case: (sometimes)

- $T(n)$ = expected time of algorithm over all inputs of size n.
- Need assumption of statistical distribution of inputs.

Best-case: (bogus)

- Cheat with a slow algorithm that works fast on *some* input.



Machine-independent time

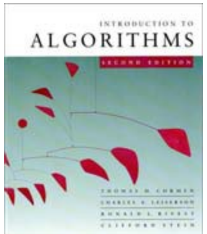
What is insertion sort's worst-case time?

- It depends on the speed of our computer:
  - relative speed (on the same machine),
  - absolute speed (on different machines).

BIG IDEA:

- Ignore machine-dependent constants.
- Look at ***growth*** of $T(n)$ as $n \rightarrow \infty$.

"Asymptotic Analysis"



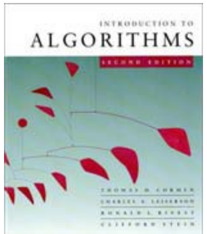
Θ-notation

Math:

$\Theta(g(n)) = \{ f(n) :$ there exist positive constants c_1, c_2, and n_0 such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0$ $\}$

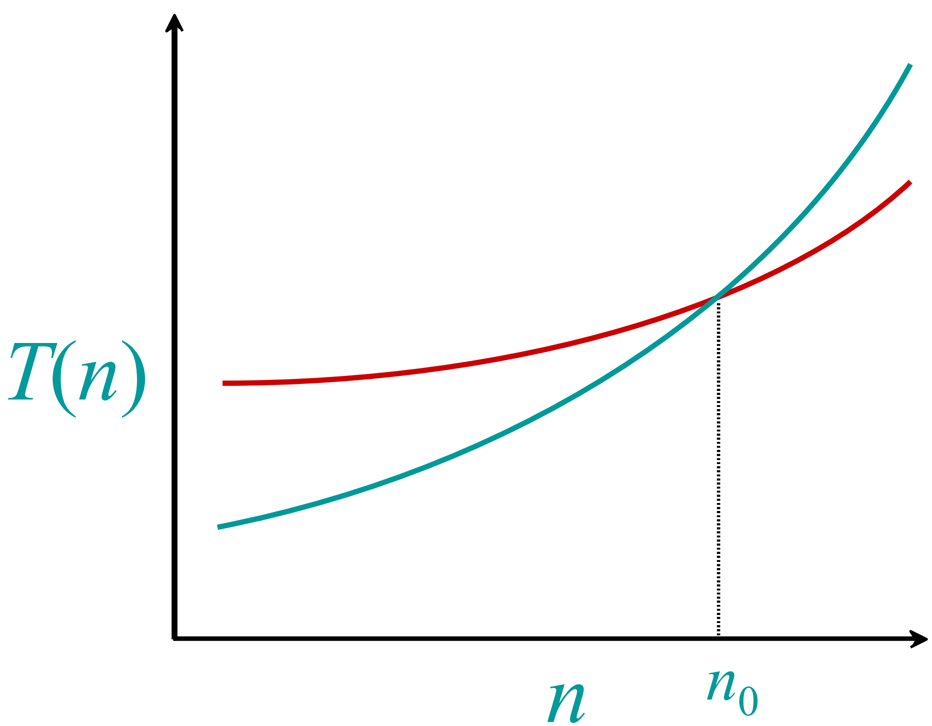
Engineering:

- Drop low-order terms; ignore leading constants.
- Example: $3n^3 + 90n^2 - 5n + 6046 = \Theta(n^3)$



Asymptotic performance

When n gets large enough, a $\Theta(n^2)$ algorithm *always* beats a $\Theta(n^3)$ algorithm.



- We shouldn't ignore asymptotically slower algorithms, however.
- Real-world design situations often call for a careful balancing of engineering objectives.
- Asymptotic analysis is a useful tool to help to structure our thinking.

Insertion Sort Analysis

INSERTION-SORT(A)	*cost*	*times*
for $j \leftarrow 2$ **to** *length*$[A]$	c_1	n
do *key* $\leftarrow A[j]$	c_2	$n-1$
▷ Insert $A[j]$ into the sorted sequence $A[1\,..\,j-1]$.	0	$n-1$
$i \leftarrow j-1$	c_4	$n-1$
while $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
do $A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
$i \leftarrow i-1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
$A[i+1] \leftarrow key$	c_8	$n-1$

Sum of all terms

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1) .$$

Insertion Sort Analysis

INSERTION-SORT(*A*)	*cost*	*times*
for $j \leftarrow 2$ **to** *length*[*A*]	c_1	n
do $key \leftarrow A[j]$	c_2	$n-1$
▷ Insert $A[j]$ into the sorted sequence $A[1 \,..\, j-1]$.	0	$n-1$
$i \leftarrow j-1$	c_4	$n-1$
while $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
do $A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
$i \leftarrow i-1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
$A[i+1] \leftarrow key$	c_8	$n-1$

Best case: sorted array

$$\sum_{j=2}^{n} t_j = \sum_{j=2}^{n} 1 = n-1$$

$$T(n) = an + b = \Theta(n)$$

Insertion Sort Analysis

INSERTION-SORT(A)	*cost*	*times*
for $j \leftarrow 2$ **to** $length[A]$	c_1	n
do $key \leftarrow A[j]$	c_2	$n-1$
▷ Insert $A[j]$ into the sorted sequence $A[1\,..\,j-1]$.	0	$n-1$
$i \leftarrow j-1$	c_4	$n-1$
while $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
do $A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^{n}(t_j-1)$
$i \leftarrow i-1$	c_7	$\sum_{j=2}^{n}(t_j-1)$
$A[i+1] \leftarrow key$	c_8	$n-1$

Worst case: reversed array

$$\sum_{j=2}^{n} t_j = \sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$$

Why?

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

Why?

Insertion Sort Analysis

$$\sum_{i=1}^{n} i = 1 + \quad 2 + \quad \cdots + \quad n-1 + \quad n$$

$$\sum_{i=1}^{n} i = n + \quad n-1 + \quad \cdots + \quad 2 + \quad 1$$

$$2\sum_{i=1}^{n} i = (n+1) + \quad (n+1) + \quad \cdots + \quad (n+1) + \quad (n+1)$$

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

Review in Appendix A

Insertion Sort Analysis

INSERTION-SORT(*A*)	*cost*	*times*
for $j \leftarrow 2$ **to** *length*[*A*]	c_1	n
do $key \leftarrow A[j]$	c_2	$n-1$
▷ Insert $A[j]$ into the sorted sequence $A[1\,..\,j-1]$.	0	$n-1$
$i \leftarrow j-1$	c_4	$n-1$
while $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
do $A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^{n}(t_j-1)$
$i \leftarrow i-1$	c_7	$\sum_{j=2}^{n}(t_j-1)$
$A[i+1] \leftarrow key$	c_8	$n-1$

Worst case: reversed array

$$\sum_{j=2}^{n} t_j = \sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$$

$$T(n) = a\,n^2 + bn + c = \Theta(n^2)$$

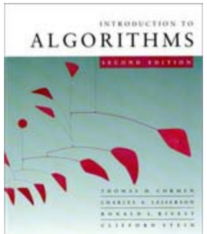
Insertion Sort Analysis

INSERTION-SORT(A)	*cost*	*times*
for $j \leftarrow 2$ **to** $length[A]$	c_1	n
do $key \leftarrow A[j]$	c_2	$n-1$
▷ Insert $A[j]$ into the sorted sequence $A[1 \,.\,.\, j-1]$.	0	$n-1$
$i \leftarrow j-1$	c_4	$n-1$
while $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
do $A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
$i \leftarrow i-1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
$A[i+1] \leftarrow key$	c_8	$n-1$

Average case: all permutations equally likely

$$\sum_{j=2}^{n} t_j = \sum_{j=2}^{n} \frac{j}{2}$$

$$T(n) = a\,n^2 + bn + c = \Theta(n^2)$$



Insertion sort analysis

Worst case: Input reverse sorted.

$$T(n)=\sum_{j=2}^{n}\Theta(j)=\Theta\left(n^2\right) \qquad \text{[arithmetic series]}$$

Average case: All permutations equally likely.

$$T(n)=\sum_{j=2}^{n}\Theta(j/2)=\Theta\left(n^2\right)$$

Is insertion sort a fast sorting algorithm?

- Moderately so, for small n.
- Not at all, for large n.

Insertion Sort Analysis

- What about space?
- Insertion sorts "in place" as it does not copy the array anywhere
- It only takes a constant amount of extra storage, independent of n
- Therefore $S(n) = \Theta(1)$

Recap

- Analysis of Algorithms
- Insertion Sort
- Asymptotic Analysis