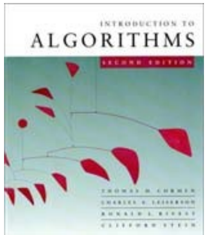
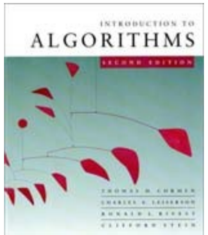
# CS60020: Foundations of Algorithm Design and Machine Learning

Sourangshu Bhattacharya




# 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!




# The problem of sorting

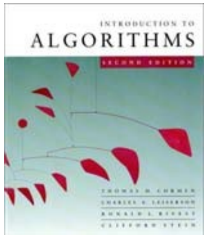
***Input:*** sequence $\langle a_1, a_2, \ldots, a_n \rangle$ of numbers.

***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 $(A, n)$ $\quad\triangleleft A[1 \,.\,.\, n]$

**for** $j \leftarrow 2$ **to** $n$

  **do** $key \leftarrow A[\,j]$

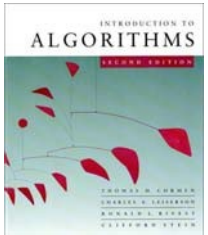
    $i \leftarrow j - 1$

    **while** $i > 0$ and $A[i] > key$

      **do** $A[i+1] \leftarrow A[i]$

        $i \leftarrow i - 1$

    $A[i+1] = key$




# Insertion sort

“pseudocode”

INSERTION-SORT $(A, n)$ $\triangleleft A[1\,.\,.\,n]$

**for** $j \leftarrow 2$ **to** $n$

  **do** $key \leftarrow A[j]$

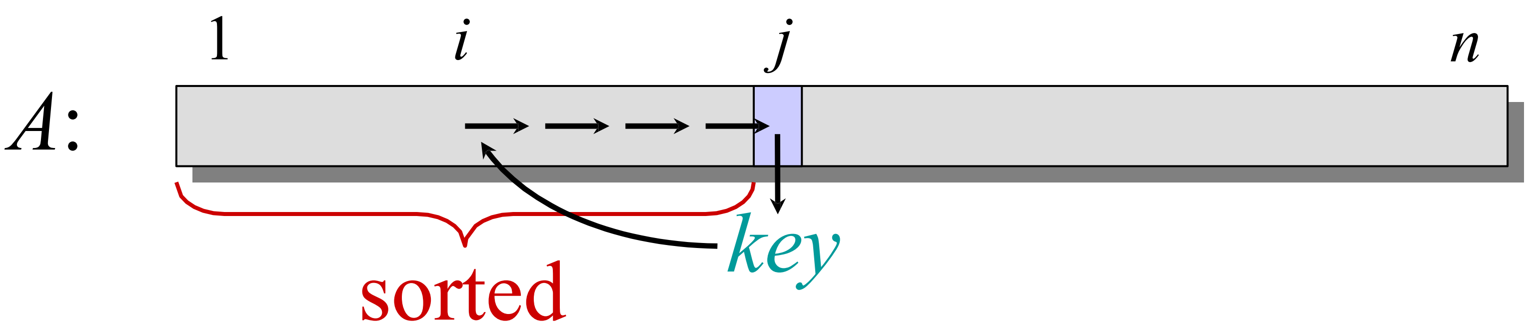
    $i \leftarrow j - 1$

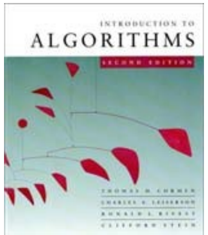
    **while** $i > 0$ and $A[i] > key$

      **do** $A[i+1] \leftarrow A[i]$

        $i \leftarrow i - 1$

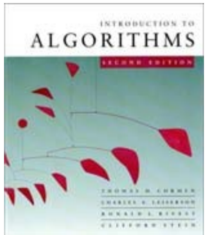
    $A[i+1] = key$





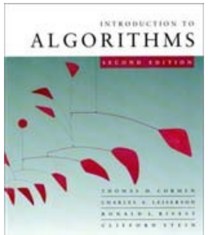

# Example of insertion sort

8 2 4 9 3 6



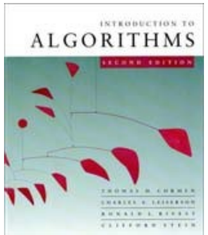

# Example of insertion sort

8 2 4 9 3 6




# Example of insertion sort

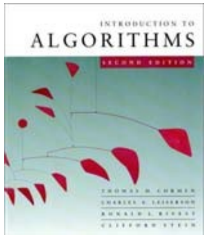
| | | | | | |
|---|---|---|---|---|---|
| 8 | 2 | 4 | 9 | 3 | 6 |
| 2 | 8 | 4 | 9 | 3 | 6 |




# Example of insertion sort

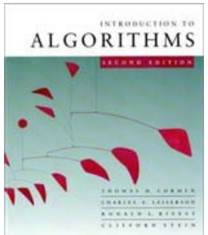
8 2 4 9 3 6

2 8 4 9 3 6



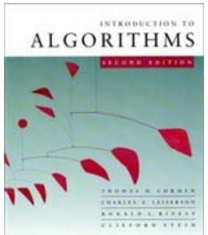

# Example of insertion sort

| | | | | | |
|---|---|---|---|---|---|
| 8 | 2 | 4 | 9 | 3 | 6 |
| 2 | 8 | 4 | 9 | 3 | 6 |
| 2 | 4 | 8 | 9 | 3 | 6 |




# Example of insertion sort

| | | | | | |
|---|---|---|---|---|---|
| 8 | 2 | 4 | 9 | 3 | 6 |
| 2 | 8 | 4 | 9 | 3 | 6 |
| 2 | 4 | 8 | 9 | 3 | 6 |



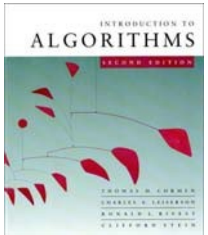

# Example of insertion sort

8 2 4 9 3 6

2 8 4 9 3 6

2 4 8 9 3 6

2 4 8 9 3 6



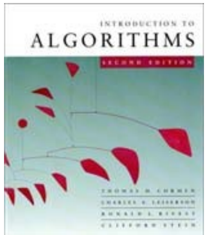

# Example of insertion sort

8 2 4 9 3 6

2 8 4 9 3 6

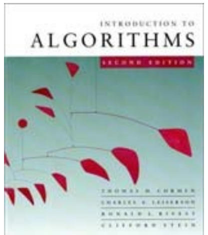
2 4 8 9 3 6

2 4 8 9 3 6



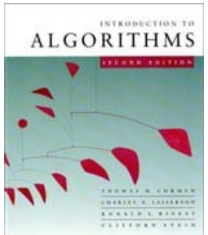

# Example of insertion sort

| | | | | | |
|---|---|---|---|---|---|
| 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 |




# 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 |




# Example of insertion sort

8 2 4 9 3 6

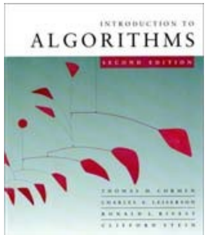
2 8 4 9 3 6

2 4 8 9 3 6

2 4 8 9 3 6

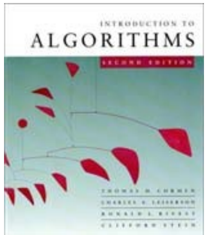
2 3 4 8 9 6

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)

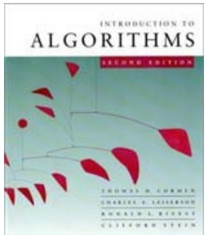
- $T(n)$ = maximum time of algorithm on any input of size $n$.

**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

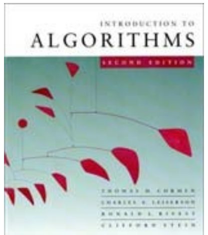
*What is insertion sort's worst-case time?*

- 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 $n \to \infty$ .

**"Asymptotic Analysis"**



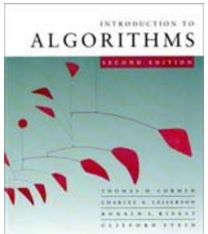

# Θ-notation

***Math:***

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

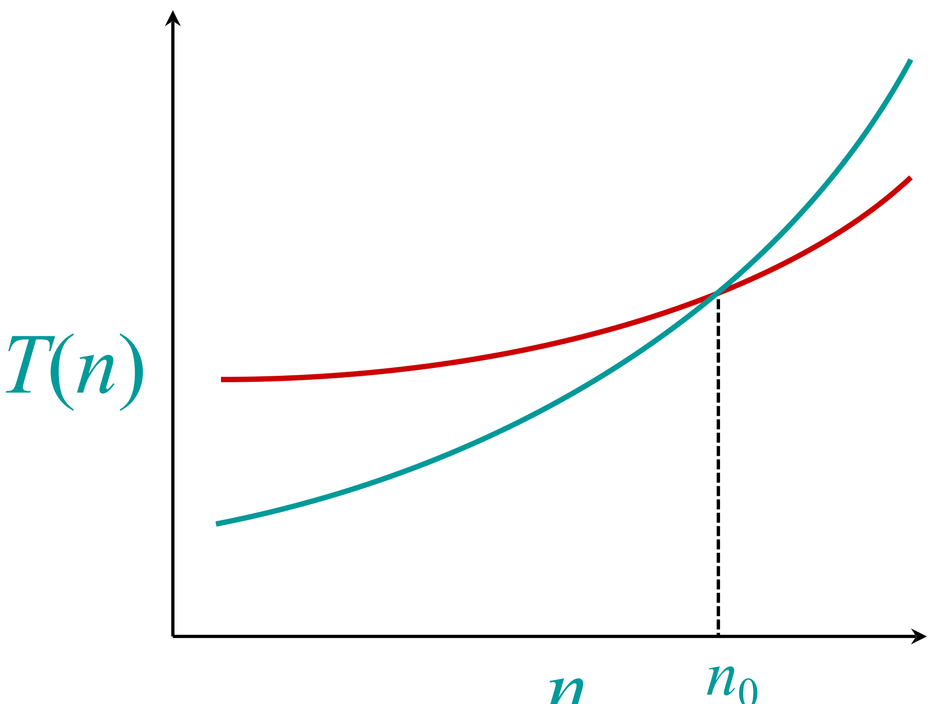
***Engineering:***

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

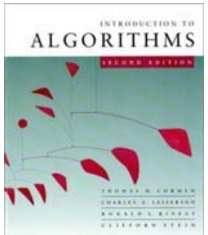



# Asymptotic performance

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

***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 $n$.

# Analysis

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