Analysis of Algorithm
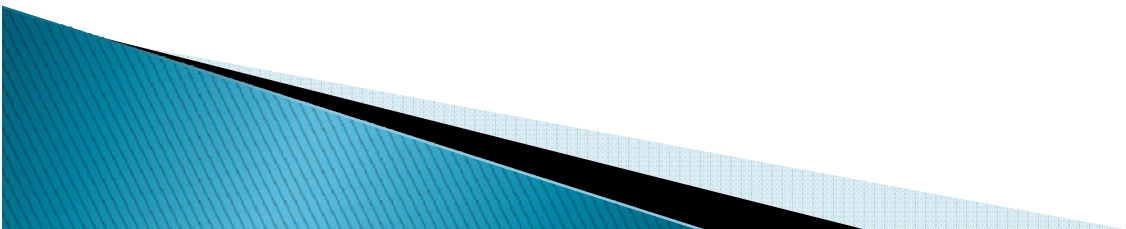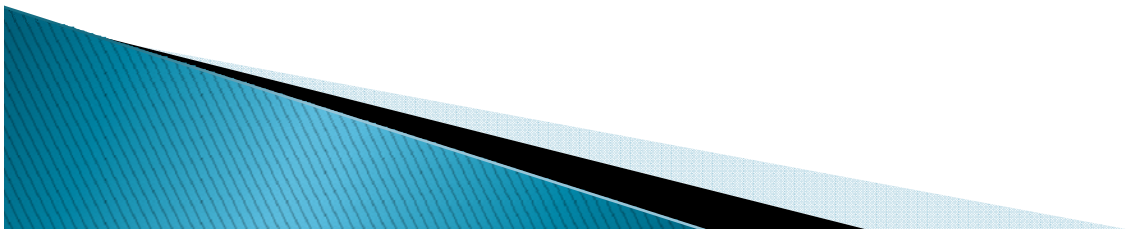
- Time

- Space

Running Time (1)

- How to analyze the running time of an algorithm?
  - We cannot compare the running time of an algorithm on a machine with the running time of another algorithm on another machine.
  - So, we have to measure the running time of every algorithm on a specific machine, which is impossible.

  - Hence, we count the **number of instructions** used by the algorithm.

Running Time (2)

- The running time of an algorithm grows with the input size.

  ➢ Sorting 10 keys is faster than sorting 1000 keys

- Input size: depends on the problem being studied.

  ➢ Sorting: Number of items in the input

     Multiplication : Number of bits

     Graph: Number of vertices and edges

- The time also depends on the input itself.

  ➢ Best, Average and Worst case

- So the running time of an algorithm is described as a function of input size n, for example, $T(n)$.

Kinds of Analysis

▶ Worst-case (Usually)

T(n)=max time on any input of size n

▶ Average-case (Sometimes)

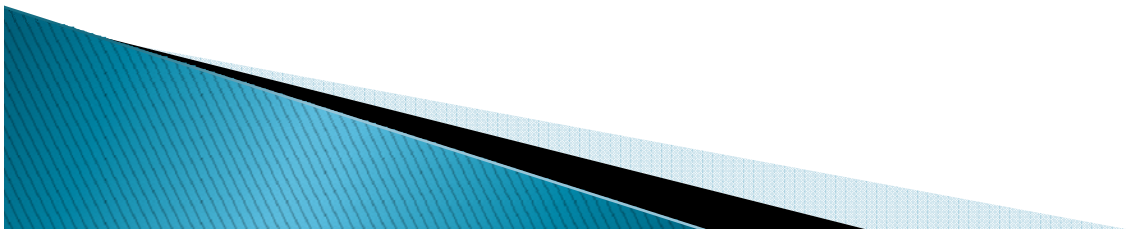T(n)=expected time over all input of size n

How do we know the probability of every particular input is?

I do not know. Make assumption of statistical distribution of inputs (what is the common assumption?)

▶ Best-Case (bogus)

Some slow algorithms work well on some input , cheating.
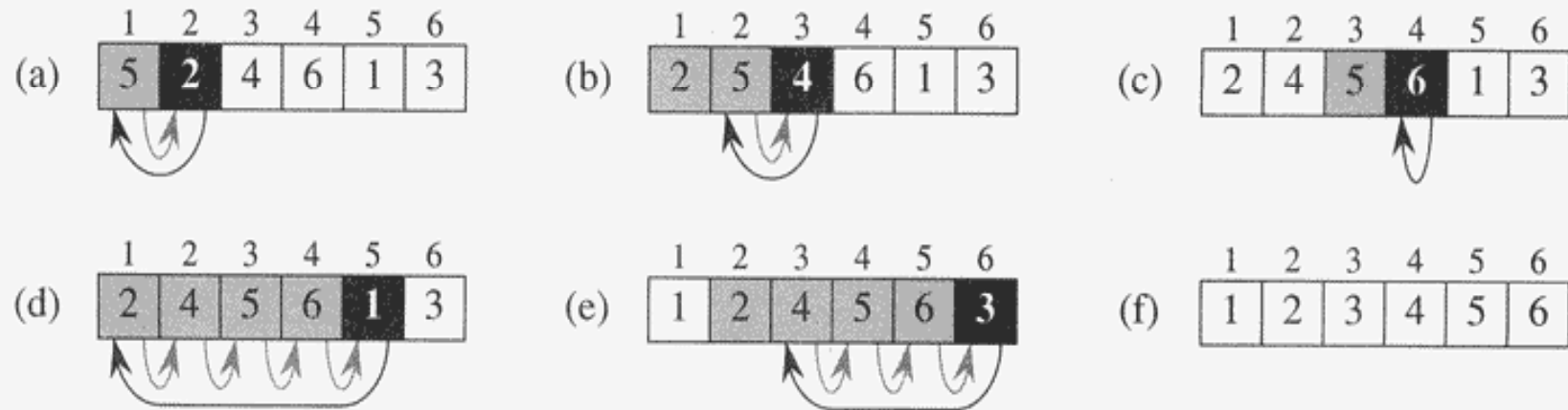
Insertion Sort Scenario

Figure 2.2 The operation of INSERTION-SORT on the array $A = \langle 5, 2, 4, 6, 1, 3 \rangle$. Array indices appear above the rectangles, and values stored in the array positions appear within the rectangles. (a)–(e) The iterations of the **for** loop of lines 1–8. In each iteration, the black rectangle holds the key taken from $A[j]$, which is compared with the values in shaded rectangles to its left in the test of line 5. Shaded arrows show array values moved one position to the right in line 6, and black arrows indicate where the key is moved to in line 8. (f) The final sorted array.

Insertion Sort Algorithm

INSERTION-SORT(A)
1   **for** $j \leftarrow 2$ **to** $length[A]$
2   **do** $key \leftarrow A[j]$
3           $\triangleright$ Insert $A[j]$ into the sorted sequence $A[1 .. j-1]$.
4           $i \leftarrow j - 1$
5   **while** $i > 0$ and $A[i] > key$
6   **do** $A[i+1] \leftarrow A[i]$
7                   $i \leftarrow i - 1$
8           $A[i+1] \leftarrow key$

Comparing and exchange in reverse order

Running time of Insertion Sort (1)

INSERTION-SORT(A)	cost	times
1 **for** $j \leftarrow 2$ **to** $length[A]$	c_1	n
2 **do** $key \leftarrow A[j]$	c_2	$n-1$
3 $\triangleright$ Insert $A[j]$ into the sorted		
sequence $A[1..j-1]$.	0	$n-1$
4 $i \leftarrow j-1$	c_4	$n-1$
5 **while** $i > 0$ **and** $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6 **do** $A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^{n}(t_j - 1)$
7 $i \leftarrow i-1$	c_7	$\sum_{j=2}^{n}(t_j - 1)$
8 $A[i+1] \leftarrow key$	c_8	$n-1$

○ **$T(n)$**: The sum of product of *cost* and *times* of each line.

Running time of Insertion Sort (2)

$$T(n) = c_1 n + c_2 (n-1) + c_4 (n-1)$$

$$+ c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1)$$

$$+ c_7 \sum_{j=2}^{n} (t_j - 1) + c_8 (n-1)$$

cost	times
c_1	n
c_2	$n - 1$
0	$n - 1$
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	$n - 1$

- *T(n)*: The sum of product of *cost* and *times* of each line.
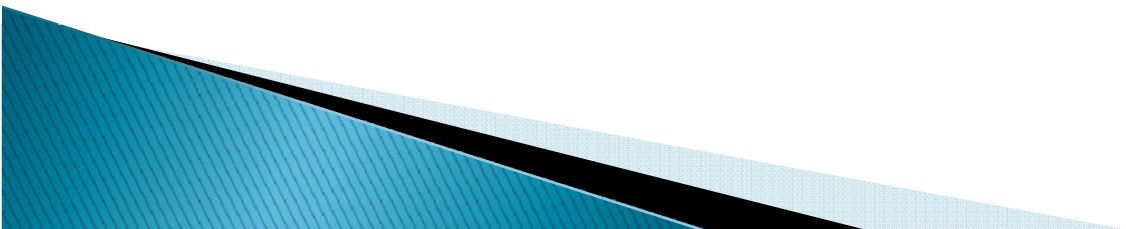
Running time of Insertion Sort (3)

- t_i: The number of times the **while** loop test is executed for j.

- Note that **for**, **while** loop test is executed one time more than the loop body.
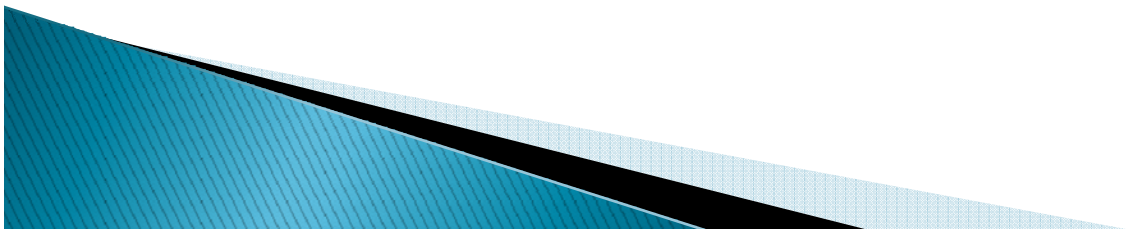
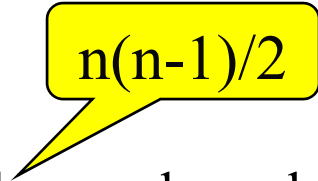# Running time of Insertion Sort (4)

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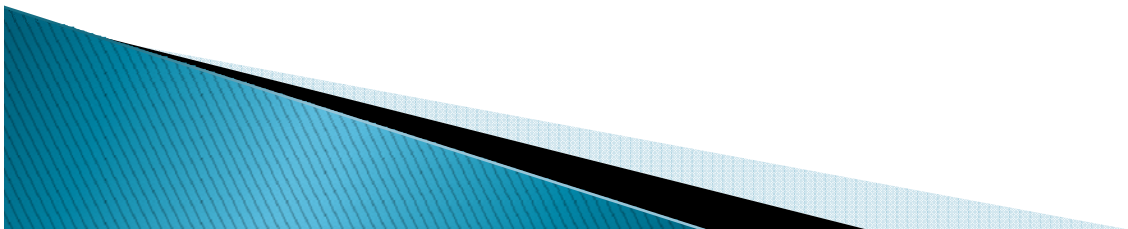
$$+ c_7 \sum_{j=2}^{n} (t_j - 1) + c_8 (n-1)$$

▸ Although the size of the input is the same, we have
  ◦ best case
  ◦ average case, and
  ◦ worst case.

Running time of Insertion Sort (5)

- Recall, $t_j = $ # of times that while loop test is executed for that value of j.

- **Best** case: the array is already sorted (all $t_j = 1$)

- **Worst** case: n(n-1)/2
  - the array is in reverse order ($t_j = j$).
  - The worst case running time gives a guaranteed upper bound on the running time for any input.

- **Average** case:
  - On average, the key in $A[j]$ is less than half the elements in $A[1 .. j-1]$ and it is greater than the other half. ($t_j = j /2$).

Running time of Insertion Sort (6)

- Best case

  If $A[1..n]$ is already sorted, $t_j = 1$ for $j = 2, 3, \ldots, n$.

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

  $$+ c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1)$$

  $$= c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1)$$

  $$= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)$$

- This running time can be expressed as *an+b* for constants *a* and *b*; it is thus a ***linear function*** of *n.*

Running time of Insertion Sort (7)

- Worst case

If $A[1..n]$ is sorted in reverse order, $t_j = j$ for $j = 2, 3, \ldots, n$.

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

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5(\frac{n(n+1)}{2} - 1)$$

$$+ c_6(\frac{n(n-1)}{2}) + c_7(\frac{n(n-1)}{2}) + c_8(n-1)$$

$$= (\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2})n^2 + (c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8)n - (c_2 + c_4 + c_5 + c_8)$$

- This running time can be expressed as $an^2 + bn + c$ for constants a, b, and c; it is thus a **quadratic function** of n.

13

Running time of Insertion Sort (8)
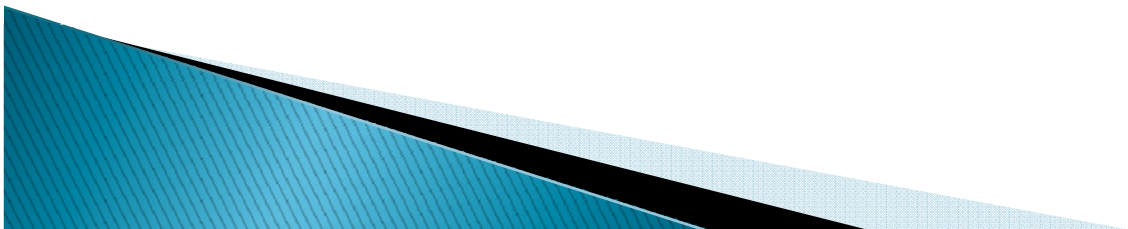
- Average case

  If the key in A[j] is less than half the elements in A[1 .. j-1] and it is greater than the other half, $t_j = j /2$ for $j = 2, 3,…, n$.

$$\sum_{j=2}^{n} \frac{j}{2} = \frac{1}{2}\left(\frac{n(n+1)}{2} - 1\right) \quad \text{and} \quad \sum_{j=2}^{n}\left(\frac{j}{2} - 1\right) = HomeWork$$
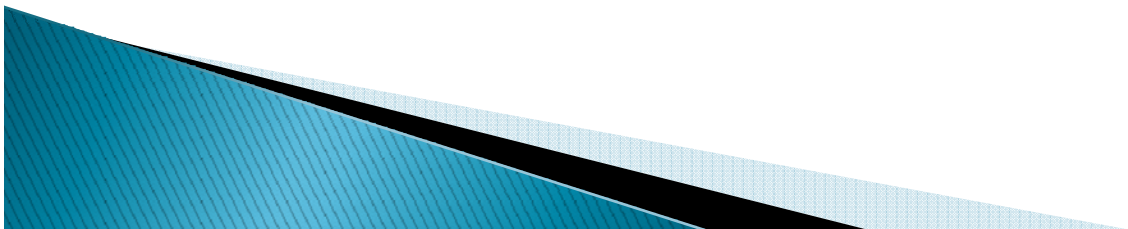
$$T(n) = HomeWork$$

- This running time can be expressed as *an² + bn + c* for constants *a*, *b*, and *c*; it is thus a **quadratic function** of *n*.

Concentrate on Worst Case running time

Reasons

➢Gives an upper bound on the running time.

➢Average case is often roughly as bad as worst case
   Example: Insertion Sort

➢Frequently occur in the database

Order of growth (1)

◆ **Look only at the leading term of the formula for running time.**
  - Drop lower-order terms.
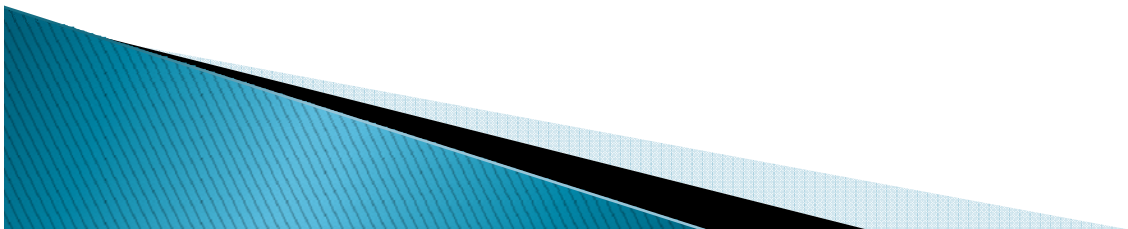 - Ignore the constant coefficient in the leading term.

◆ **Example:** $an^2 + bn + c = \Theta(n^2)$
  - Drop lower-order terms $\Rightarrow an^2$
  - Ignore constant coefficient $\Rightarrow n^2$
  - The running time $T(n)$ grows like n^2; it does **not equal to** n^2.
  - Running time is $\Theta(n^2)$ to capture the notion that **the order of growth is n^2.**
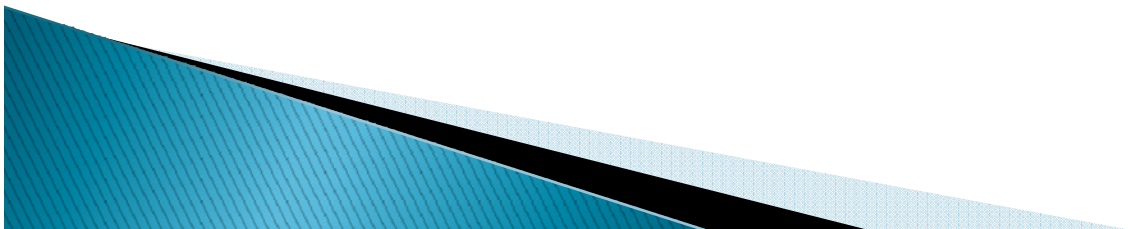
Order of growth (2)

- We usually consider one algorithm to be more efficient than another if its worst-case running time has a lower order of growth.

- Due to constant factors and lower-order terms, this evaluation may be error for small inputs but for large enough inputs, it is true.

Stay Safe