

CSE515 Advanced Algorithms

Lecture 1: Insertion Sort

Antoine Vigneron
antoine@unist.ac.kr

Ulsan National Institute of Science and Technology

March 2, 2021

- 1 Introduction
- 2 Algorithm
- 3 Proof of correctness
- 4 Analysis

Introduction

Problem (Sorting)

Given an input sequence of n numbers, the *sorting problem* is to find a permutation of the input sequence sorted in nondecreasing order.

- The sorting problem can also be stated as follows:
 - ▶ **Input:** a sequence of n numbers (a_1, a_2, \dots, a_n)
 - ▶ **Output:** a permutation of the input sequence $(a'_1, a'_2, \dots, a'_n)$ such that $a'_1 \leq a'_2 \leq \dots \leq a'_n$

Example

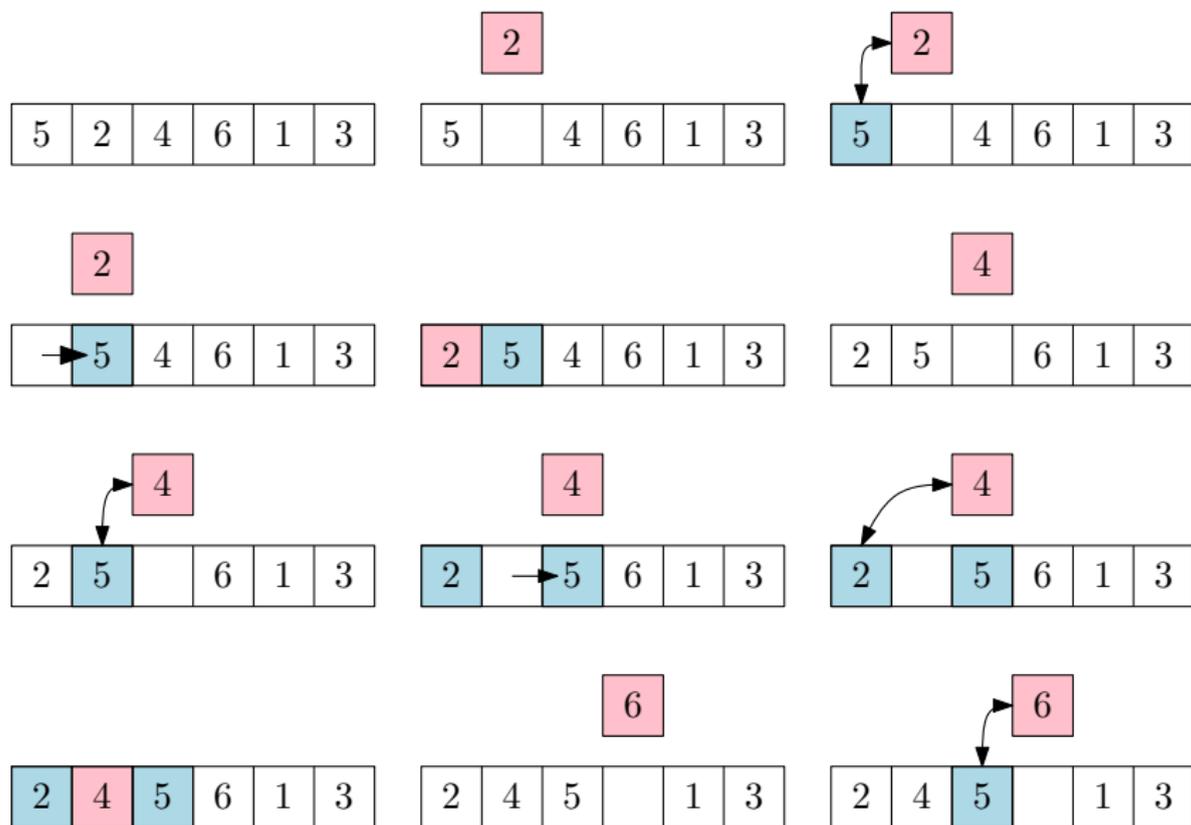
- **Input:** (6, 1, 7, 6, 4)
- **Output:** (1, 4, 6, 6, 7)

- The numbers a_i that we wish to sort are also called the *keys*.

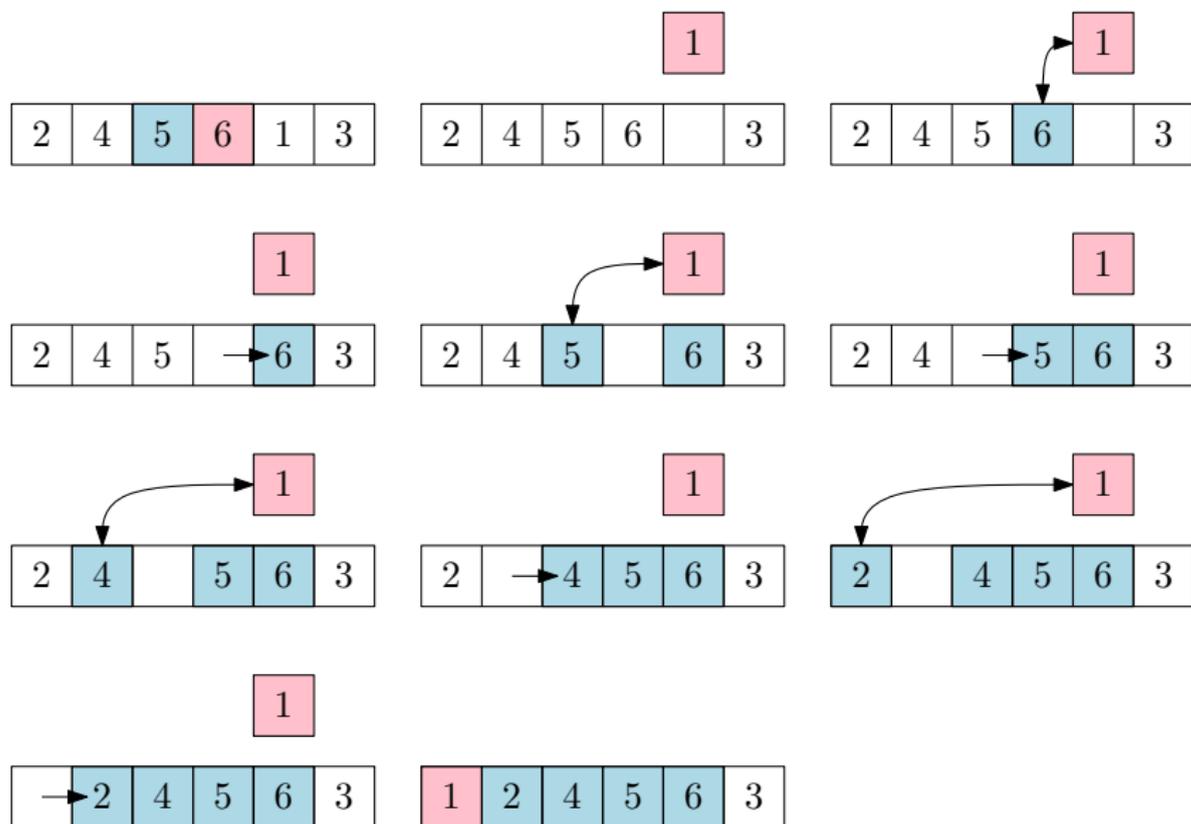
Introduction

- This lecture and the next one are reviews of undergraduate algorithms topics.
- In this lecture, we present a first sorting algorithm.
 - ▶ **Reference:** Chapter 2 of the textbook [Introduction to Algorithms](#) by Cormen, Leiserson, Rivest and Stein.
 - ▶ Available online from the UNIST library website.
- The main goal is to introduce the framework of this course.
- Sorting is an important problem.
 - ▶ In the 60's, 25% of computing time was spent on sorting.
 - ▶ It allows us to illustrate several algorithmic techniques.

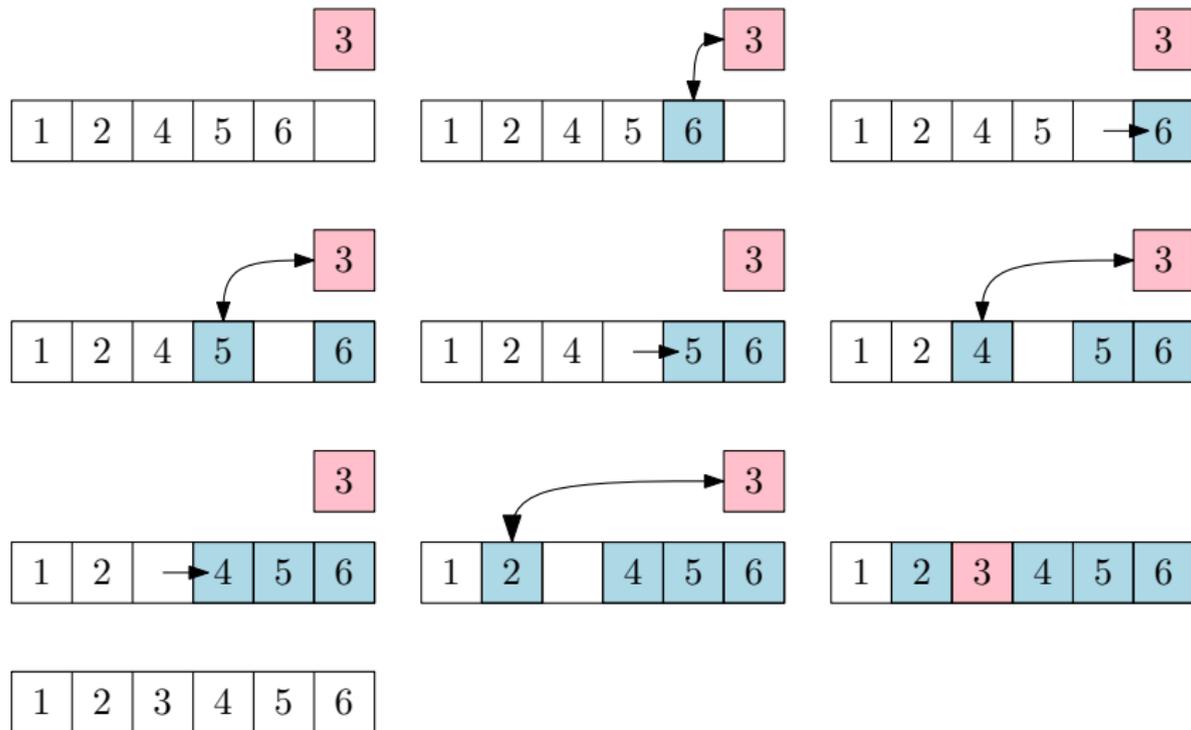
Algorithm



Algorithm



Algorithm



Algorithm

- INSERTION SORT proceeds from left to right. The current element $A[j]$ (red) is inserted into $A[1 \dots j - 1]$.
- $A[j]$ is compared with all the blue keys.
- INSERTION SORT is a very natural algorithm.
 - ▶ People use it to sort a deck of cards.

Algorithm

- *Pseudocode* of INSERTION SORT:

INSERTION SORT

```
1: procedure INSERTION-SORT( $A[1 \dots n]$ )
2:   for  $j \leftarrow 2, n$  do
3:      $\text{key} \leftarrow A[j]$ 
4:      $i \leftarrow j - 1$ 
5:     while  $i > 0$  and  $A[i] > \text{key}$  do
6:        $A[i + 1] \leftarrow A[i]$ 
7:        $i \leftarrow i - 1$ 
8:      $A[i + 1] \leftarrow \text{key}$ 
```

- We will present algorithms in pseudocode in this course.
 - ▶ Sometimes resembles C, Java, Python...
 - ▶ Sometimes uses plain English.
 - ▶ No strict rule.
 - ▶ Should be clear and concise.

Proof of Correctness

- We now want to prove that INSERTION SORT outputs a correct result.
 - ▶ i.e. at the end of the execution, A is sorted.
- Strategy: We use a *loop invariant*.

Loop invariant for INSERTION SORT

At the start of each iteration of the for loop, the subarray $A[1 \dots j - 1]$ consists of the elements originally in $A[1 \dots j - 1]$ in sorted order.

- A proof by loop invariant requires to prove 3 properties:
 - ▶ **Initialization.** It is true prior to the first iteration of the loop.
 - ▶ **Maintenance.** If it is true before an iteration of the loop, it remains true before the next iteration.
 - ▶ **Termination.** When the loop terminates, the invariant gives us a useful property that helps show that the algorithm is correct.
- Proofs done in class. See textbook page 19.

Remarks on Loop Invariants

- A loop invariant is a property that is true at the beginning (or at the end) of each iteration of a loop.
- You should always state the loop invariant precisely.
- A proof by loop invariant is essentially a proof by induction.
- It is useful when a detailed and rigorous proof is needed.

Analysis

- *Analyzing* an algorithm means predicting the amount of resources it uses.
 - ▶ Usually: estimate the *running time*, i.e. the time needed for the algorithm to complete.
 - ▶ It requires a model of computation.
- Our model of computation: The *Random Access Machine (RAM)*.
- RAM can perform in constant time simple instructions such as:
 - ▶ Arithmetic operations $+$, $-$, \times , $/$, remainder, floor, ceiling
 - ▶ Branching instructions (IF THEN ELSE,)
 - ▶ Copying a single variable (not a whole array)
 - ▶ Accessing an element of an array
- The *input size* n is the number of bits, or the number of words needed to encode the problem. We will specify it for each problem.
 - ▶ Here n is the size of the input array $A[1 \dots n]$.
- Data types:
 - ▶ Word size $c \log n$ for an input of size n , where c is a constant.
 - ▶ For instance, $c \log n$ -bit integers.

Analysis

INSERTION SORT

	line	cost	times
1: procedure INSERTION-SORT($A[1 \dots n]$)	2	c_2	n
2: for $j \leftarrow 2, n$ do	3	c_3	$n - 1$
3: $\text{key} \leftarrow A[j]$	4	c_4	$n - 1$
4: $i \leftarrow j - 1$	5	c_5	$\sum_{j=2}^n t_j$
5: while $i > 0$ and $A[i] > \text{key}$ do	6	c_6	$\sum_{j=2}^n (t_j - 1)$
6: $A[i + 1] \leftarrow A[i]$	7	c_7	$\sum_{j=2}^n (t_j - 1)$
7: $i \leftarrow i - 1$	8	c_8	$n - 1$
8: $A[i + 1] \leftarrow \text{key}$			

- t_j : # of times the while loop test is performed
- c_k , $k = 2 \dots 8$ is the time taken to execute line k once
 - ▶ Unknown constant, depends on your hardware/OS/compiler ...
- Line 2 takes time $c_2 n$ because we count the n th iteration where $j = n + 1$ and we check whether $j > n$, before exiting the loop.

Analysis

- So the running time is

$$T(n) = c_2 n + c_3(n - 1) + c_4(n - 1) + c_5 \sum_{j=2}^n t_j \\ + c_6 \sum_{j=2}^n (t_j - 1) + c_7 \sum_{j=2}^n (t_j - 1) + c_8(n - 1).$$

Analysis

- If the input is already sorted, then $t_j = 1$ for all j .
- So the running time on sorted input is

$$T(n) = (c_2 + c_3 + c_4 + c_5 + c_8)n - (c_3 + c_4 + c_5 + c_8)$$

- $T(n)$ cannot be smaller for any input of size n , as we have $t_j = 1$ for all j .
- It is the *best-case running time*.
- As $T(n) = an + b$ for two constants a, b , we say that it is a *linear function*.

Analysis

- Suppose that the input A is sorted in decreasing order:
 $A[1] > A[2] > \dots > A[n]$.
- Then $t_j = j$ for all j .
- As $\sum_{j=2}^n j = \frac{n(n+1)}{2} - 1$ and $\sum_{j=2}^n j - 1 = \frac{n(n-1)}{2}$, we get:

$$T(n) = \left(\frac{c_5 + c_6 + c_7}{2} \right) n^2 + \left(c_2 + c_3 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8 \right) n - (c_2 + c_4 + c_5 + c_8).$$

- As t_j cannot be larger, this is the *worst-case running time*.
- Since $T(n)$ can be written $an^2 + bn + c$ for some constants a, b, c , we say that it is a *quadratic function*.

Analysis

- We usually perform a worst-case analysis rather than best case.
- Reasons:
 - ▶ It gives a *guarantee* on the running time.
 - ▶ It often happens in practice.
 - ▶ The average case is often roughly as bad.
 - ★ Example: Apply INSERTION SORT to a set of random numbers.
 - ★ Then t_j is about $j/2$ on average.
 - ★ So the average running time is still quadratic.
- When the running time is linear, we will write $T(n) = \Theta(n)$, and when it is quadratic, we will write $T(n) = \Theta(n^2)$.
 - ▶ We will study this in details in two weeks.
 - ▶ Intuition: Keep the dominant term, remove constant factors.