Algorithms and Data Structures

1st Lecture: Getting Started

Yutaka Watanobe, Jie Huang, Yan Pei, Wenxi Chen,
S. Semba, Deepika Saxena, Yinghu Zhou, Akila Siriweera

University of Aizu

Last Updated: 2023/12/05

Outline

- Algorithms
- Data Structures
- Pseudocode
- Insertion Sort

Algorithms

- Informally, an algorithm is any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output.
- An algorithm is thus a sequence of computational steps that transform the input into the output.
- Example: Sorting problem
  - Input: A sequence of n numbers $(a_1, a_2, ..., a_n)$.
  - Output: A permutation (i.e., reordering) $(a'_1, a'_2, ..., a'_n)$ of the input sequence such that $a'_1 \leq a'_2 \leq ... \leq a'_n$.

  Such an input sequence is called an instance of the sorting problem.

Correct Algorithms

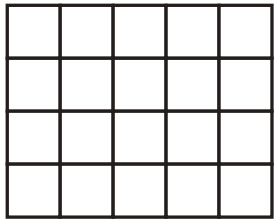
- An algorithm is said to be correct if, for every input instance, it halts with the correct output. We also say that a correct algorithm **solves** the given computational problem.
- An algorithm can be specified in English, as a computer program, or even as a hardware design. The only requirement is that the specification must provide a **precise description** of the computational procedure to be followed.

Data Structures

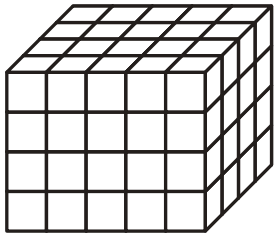
- A data structure is a way to store and organize data in order to facilitate **access** and **modifications**.
- No single data structure works well for all purposes, and so it is important to know the strengths and limitations of several of them.

Data Structures: Examples

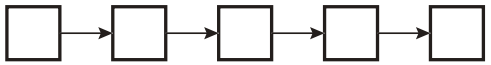
Array



2D-Array



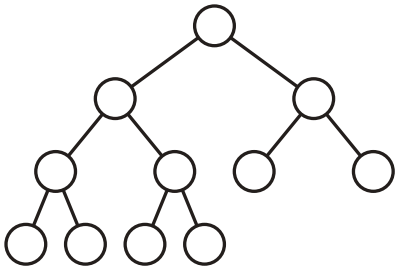
ND-Array



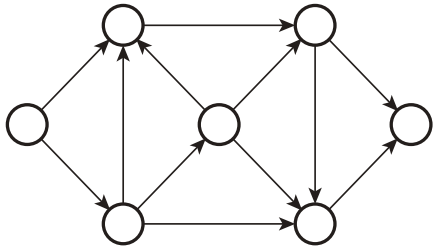
List

Queue

Stack



Tree



Graph

Pseudocode Convention (1)

- We typically describe algorithms as programs written in a pseudocode that is similar in many respects to C, C++ or Java.
- In pseudocde, we employ whatever method is most clear and concise to specify a given algorithm.
- The symbols /* ... */ indicate that the statement is a comment.
- The symbols // also indicate that the statement located on the right side of them is a commnet.
- Indentation indicates block structure.
- The symbol $\leftarrow$ or = indicates variable assignment.
- Parameters are passed to a procedure by value.
- The Boolean operators are short-circuiting.

Pseudocode Convention (2)

- A[i] indicates the *i*-th element of an array A.
- A.length indicates the length of the array A.
- A[1, ... , j] indicates a contiguous subsequence of A including A[1], A[2], ... A[j]
- We use both 0-origin and 1-origin depending on the situation.

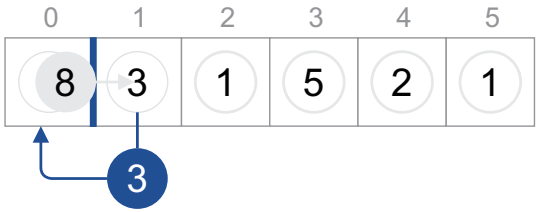
Insertion Sort

- Consider the previous sorting problem.
  - Input: A sequence of n numbers $(a_1, a_2, ..., a_n)$.
  - Output: A permutation (i.e., reordering) $(a'_1, a'_2, ..., a'_n)$ of the input sequence such that $a'_1 \leq a'_2 \leq ... \leq a'_n$.
- Let us consider the Insertion Sort algorithm.

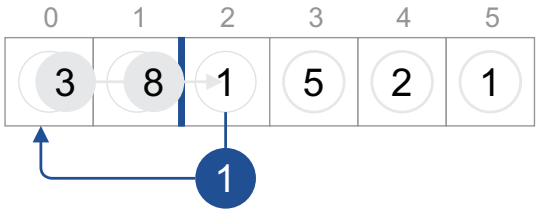
Insertion Sort (into nondecreasing order)

0.

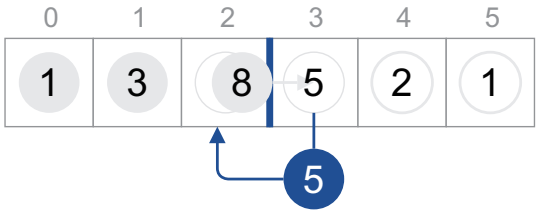
1.



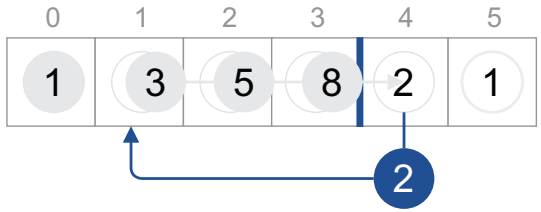
2.



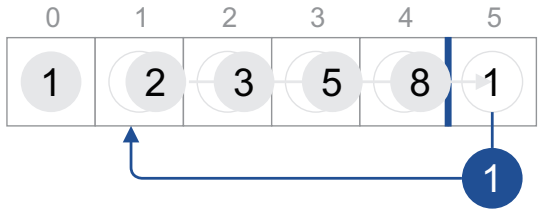
3.



4.



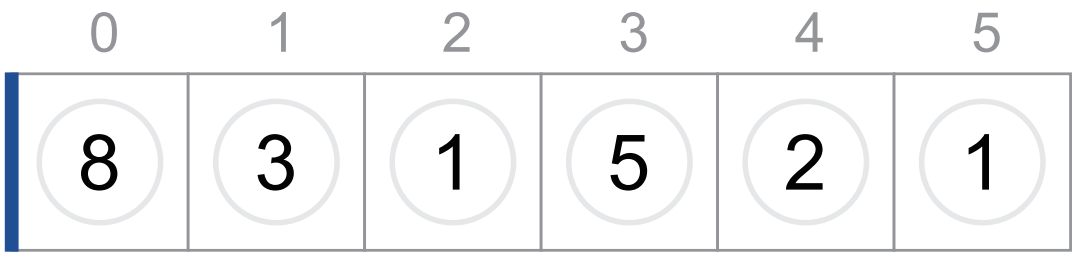
5.



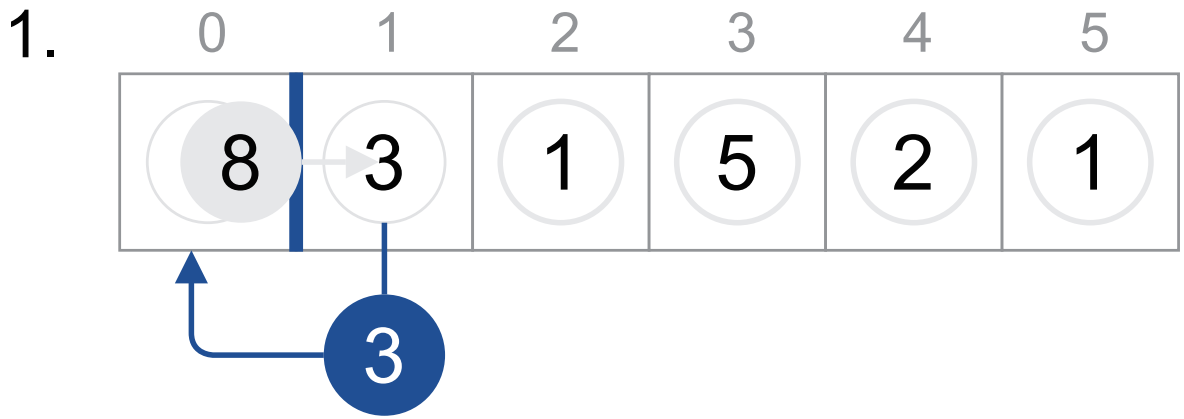
6.

Insertion Sort (0)

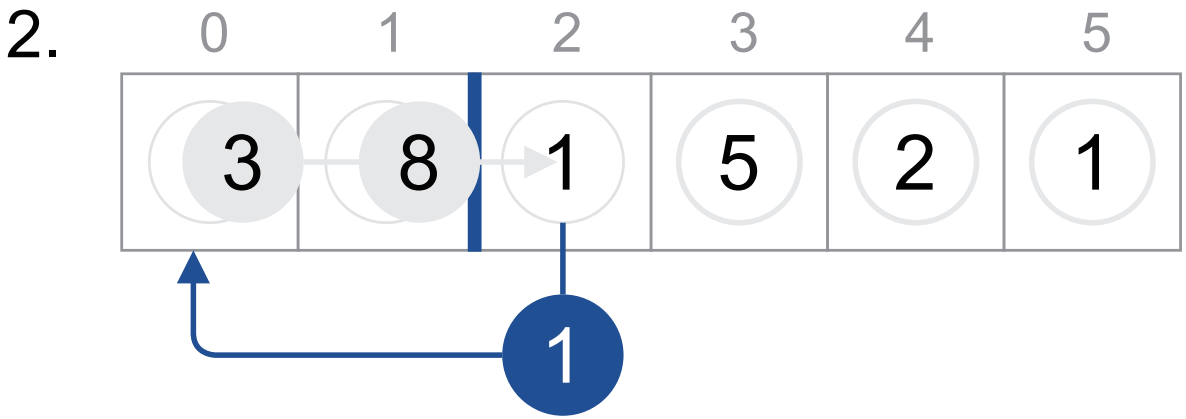
0.



Insertion Sort (1)

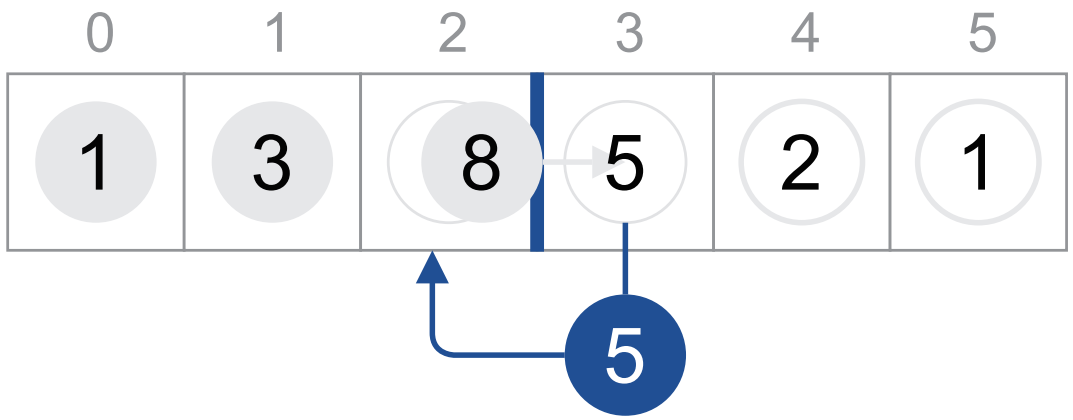


Insertion Sort (2)



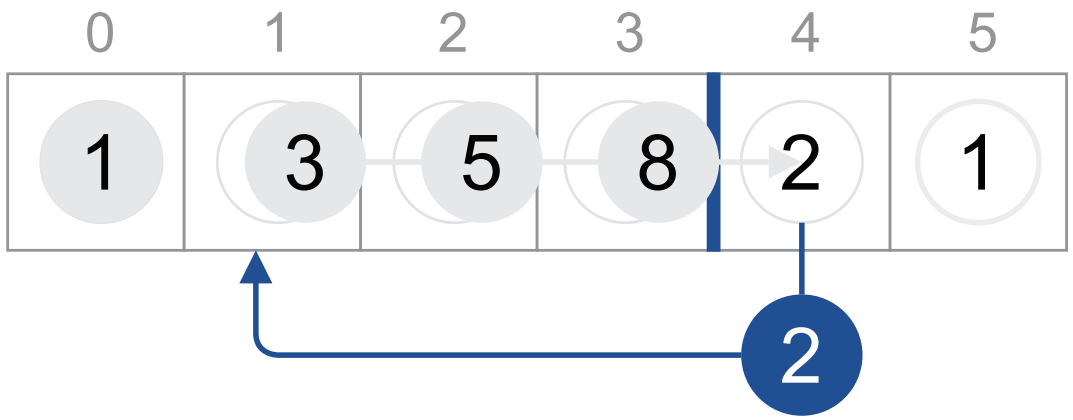
Insertion Sort (3)

3.



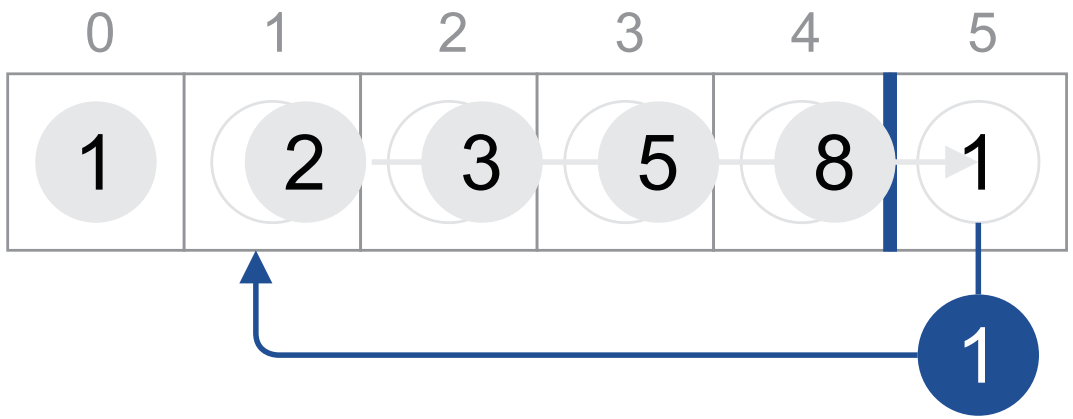
Insertion Sort (4)

4.



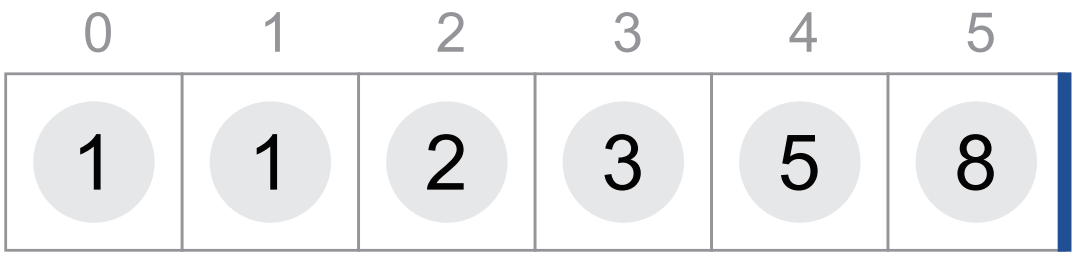
Insertion Sort (5)

5.



Insertion Sort (6)

6.



Implementation

Pseudocode Insertion Sort (Note that array indices are based on 0-origin, and the number of elements is $n =$ A.length)

```
01. for j ← 1 to n-1
02.     key ← A[j]
03.     // insert A[j] into the sorted sequence A[0,...,j-1]
04.     i ← j - 1
05.     while i ≥ 0 and A[i] > key
06.         A[i+1] ← A[i]
07.         i ← i-1
08.     A[i+1] ← key
```

Analyzing Algorithms

- Analyzing an algorithm has come to mean predicting the resources that the algorithm requires.
- Occasionally, resources such as memory, communication bandwidth, or computer hardware are of primary concern, but most often it is computational time that we want to measure.

Input Size and Running Time

- The best notion for input size depends on the problem being studied.
- For many problems, the most natural measure is the number of items in the input. For many other problems, the best measure is the total number of bits needed to represent the input in ordinary binary notation.
- The running time of an algorithm on a particular input is the number of primitive operations or ”steps” executed.

Analysis of Insertion Sort

InsertionSort(A)	cost	times
01. for j ← 1 to n-1	c_1	n
02. key ← A[j]	c_2	$n-1$
03. // comment line	c_3	$n-1$
04. i ← j - 1	c_4	$n-1$
05. while i ≥ 0 and A[i] > key	c_5	$\sum_{j=1}^{n-1} t_j$
06. A[i+1] ← A[i]	c_6	$\sum_{j=1}^{n-1} (t_j - 1)$
07. i ← i-1	c_7	$\sum_{j=1}^{n-1} (t_j - 1)$
08. A[i+1] ← key	c_8	$n-1$

Let t_j be the number of times the while loop test in line 5 is executed for the value of j.

Calculation of Running Time

- The running time of the algorithm is the sum of running times for each statement executed.
- We sum the product of the ”cost” and ”times” columns.
- Let $T(n)$ be the running time of InsertionSort(A) on an input sequence of size n.

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

Calculation (Best Case)

- The best case occurs if the array is already sorted.
- For each $j = 1, 2, ..., n-1$, we then find that A[i] $\leq$ key in line 5 when i has its initial value of $j - 1$.
- Thus, $t_j = 1$ for $j = 1, 2, ..., n-1$.

The best-case running time is given as follows:

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) \\ &= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8) \end{aligned}$$

it is thus a **linear function** of n.

Calculation (Worst Case)

- The worst case occurs if the array is in reverse sorted order.
- We must compare each element A[j] with each element in the entire sorted subarray A[0..j-1].
- Thus, $t_j = j$ for $j = 1, 2, ..., n-1$.

The worst-case running time is given as follows:

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) \\ &+ c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1) \\ &= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2}\right. \\ &- \left.\frac{c_7}{2} + c_8\right)n - (c_2 + c_4 + c_5 + c_8) \end{aligned}$$

it is thus a **quadratic function** of n.

Worst-Case and Average-Case Analysis

- We usually concentrate on finding only the worst-case running time, that is, the longest running time for any input of size n.
- In some particular cases, we shall be interested in the average-case (or expected) running time of an algorithm.

Worst-Case Analysis

Three reasons for using the worst case:

- The worst-case running time of an algorithm is an upper bound on the running time for any input. Knowing it gives us a guarantee that the algorithm will never take any longer.
- For some algorithms, the worst case occurs fairly often. For example, in searching a database for a particular piece of information, the searching algorithm's worst case will often occur when the information is not present in the database.
- The average case is often roughly as bad as the worst case.

Reference

1 Introduction to Algorithms (third edition), Thomas H.Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. The MIT Press, 2012.