Introduction to Algorithms

Date: 3/10 (Tuesday)

Instructor: 유준수

Assignment

- Read 2.2, 2.3.1
- Problems:
  - 2.2절 – 1, 2, 3

Summary

1.1 : Alg $\Rrightarrow$ pseudocode

(precise description)

input size n

1.2 : hardware vs algorithm

2.1 : Insertion sort

I : $\langle a_1, \cdots, a_n \rangle$

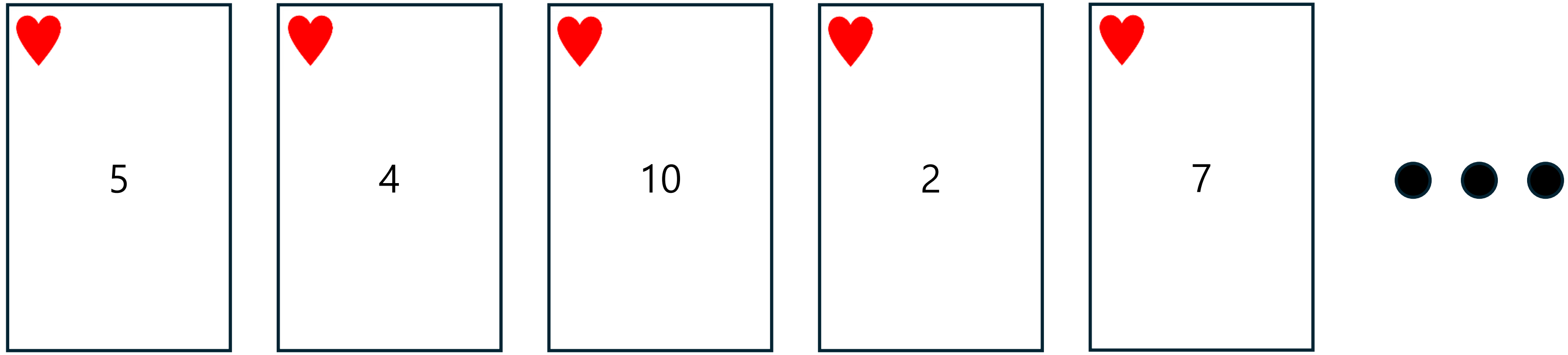
such that

O : $\langle a_1', \cdots, a_n' \rangle$ s.t.

$a_1' \le \cdots \le a_n'$

Insertion Sort

A:



i j

$A[j] = key$

Pseudocode

```
for i=2 to n
    key = A[i]
    j = i-1
    while A[j] > key , j>0
        A[j+1] = A[j]
        j = j-1
    A[j+1] = key
```

Review of 1.1, 1.2, 2.1

Correctness

Induction

- Loop Invariant
- Init/Main/Ter

```
INSERTION-SORT(A, n)
for i = 2 to n
    key = A[i]
    // Insert A[i] into the sorted subarray A[1 : i − 1].
    j = i − 1
    while j > 0 and A[j] > key
        A[j + 1] = A[j]
        j = j − 1
    A[j + 1] = key
```

→ $i = 2;\ i \le n;\ i{+}{+}$

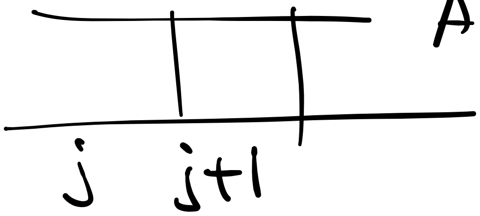
"at the start of the loop", property holds

Loop Inv : $A[1 : i-1]$ is sorted.

Init $A[1]$ is sorted, trivial

Main $A[1 : i-1]$ is sorted.

$\Rightarrow A[i]$ finds position s.t.



$A[j] \le A[i] \le A[j+1]$

$\Rightarrow A[1:i]$ is sorted

Termination $A[1:n]$ is sorted

$\Rightarrow$ Alg. correctly outputs sorted array

Problem 2.1-4

2.1-4

Consider the ***searching problem***:

Input: A sequence of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$ stored in array $A[1:n]$ and a value x.

Output: An index i such that x equals $A[i]$ or the special value NIL if x does not appear in A.

Write pseudocode for ***linear search***, which scans through the array from beginning to end, looking for x. Using a loop invariant, prove that your algorithm is correct. Make sure that your loop invariant fulfills the three necessary properties.

Think about 2 min…

Searching Problem

$I: \langle a_1, \dots, a_n \rangle, x$

$\downarrow$

O: return index i s.t. $x = A[i]$ or NIL if $x \neq A[i]$ for $i \in [1, n]$

1. Pseudocode
2. Correctness

Pseudocode

```
lin_search(A, n, x)
  for i = 1 to n
      if A[i] = x
          return i
  return NIL
```

loop invariant: $A[1:i-1]$ doesn't contain x at the start of ith iter.

Proof of Correctness

Init ϕ, trivial

Main $A[1:i-1]$ doesn't contain x

$\Rightarrow$ check $A[i] = x$

$\Rightarrow$ If match, returns index i

$\Rightarrow$ If match X, $A[1:i]$ doesn't contain x

Termination: ① match occurs,

returns the index

② If no match occurs, A[1:n]

⇒ NIL

Chapter 2. The Role of Algorithms in Computing

- 2.1 Insertion sort
- 2.2 Analyzing algorithms
- 2.3 Designing algorithms

Model Assumption: RAM

$A_1, \dots, A_k$

$\Downarrow$

efficiency

$\uparrow$

RAM model

Ground basis : ① Instruction runs one after another

② Instruction takes constant

time

Calculate Time Complexity (hard way): Insertion Sort

```
INSERTION-SORT(A, n)
for i = 2 to n
    key = A[i]
    // Insert A[i] into the sorted subarray A[1 : i – 1].
    j = i – 1
    while j > 0 and A[j] > key
        A[j + 1] = A[j]
        j = j – 1
    A[j + 1] = key
```

cost	times
c_1	n
c_2	$n-1$
c_3	
c_4	$n-1$
c_5	$\sum_{i=2}^{n} t_i$
c_6	$\sum (t_i - 1)$
c_7	$\sum (t_i - 1)$
c_8	$n-1$

$$T(n) = c_1(n-1) + c_2(n-2) + \cdots$$

Best case analysis

worst
average

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \underbrace{\sum_{i=2}^{n} t_i}_{(n-1)} + c_6 \cancel{\sum_{i=2}^{n}(t_i - 1)}$$
$$+ \cancel{c_7 \sum_{i=2}^{n}}(t_i - 1) + c_8(n-1) .$$

$t_i = 1$

$$T(n) = (c_1 + c_2 + c_3 + c_4 + c_n)\, n$$
$$- c_2 - c_3 - c_4 - c_n$$
$$= an + b$$

linear in n $= \Theta(n)$

Worst-case analysis

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

$$t_i = i$$

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

$$T(n) = \left(\frac{c_5}{2} + \cdots\right) n^2 + (c_1 \ldots) n - (c_2 + c_4 + \cdots)$$

quadratic in n

Average-case analysis

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

$t_i = i/2$

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

$= \boxed{\text{O}\, n^2} + \text{O}n + \text{O}$

$= \Theta(n^2)$

Summary of time complexity table (insertion sort)

Case	t_i	Time Complexity
Best-case	1	$\Theta(n)$
Worst-case	i	$\Theta(n^2)$
Avg-case	$i/2$	$\Theta(n^2)$

Order of Growth (Rate of growth)

$$T(n) = c_1 n^k + c_2 n^{k-1} + \cdots + c_{k+1}$$

$$= \boxed{\Theta}(n^k)$$

"roughly proportional to"

Why do we ignore the constant term?

$$T(n) = \frac{1}{100} n^2 + 100n + \cdots$$

$n = 10^6$

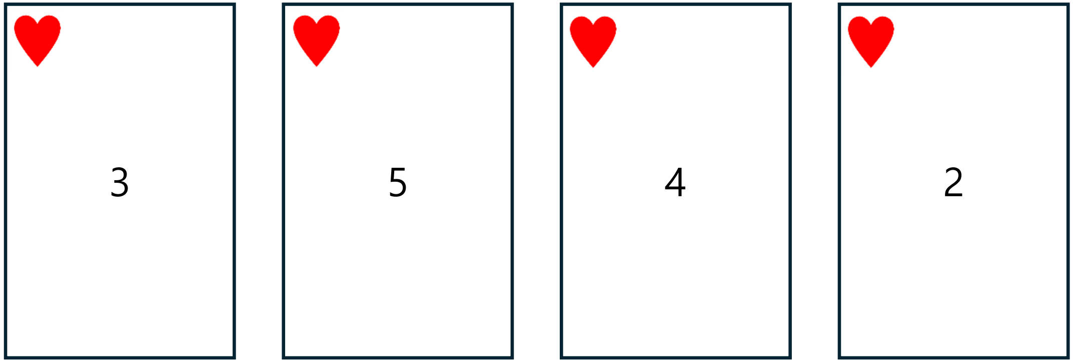
$\downarrow \quad 10^{10}$

$\downarrow \quad 10^{8}$

Chapter 2. The Role of Algorithms in Computing

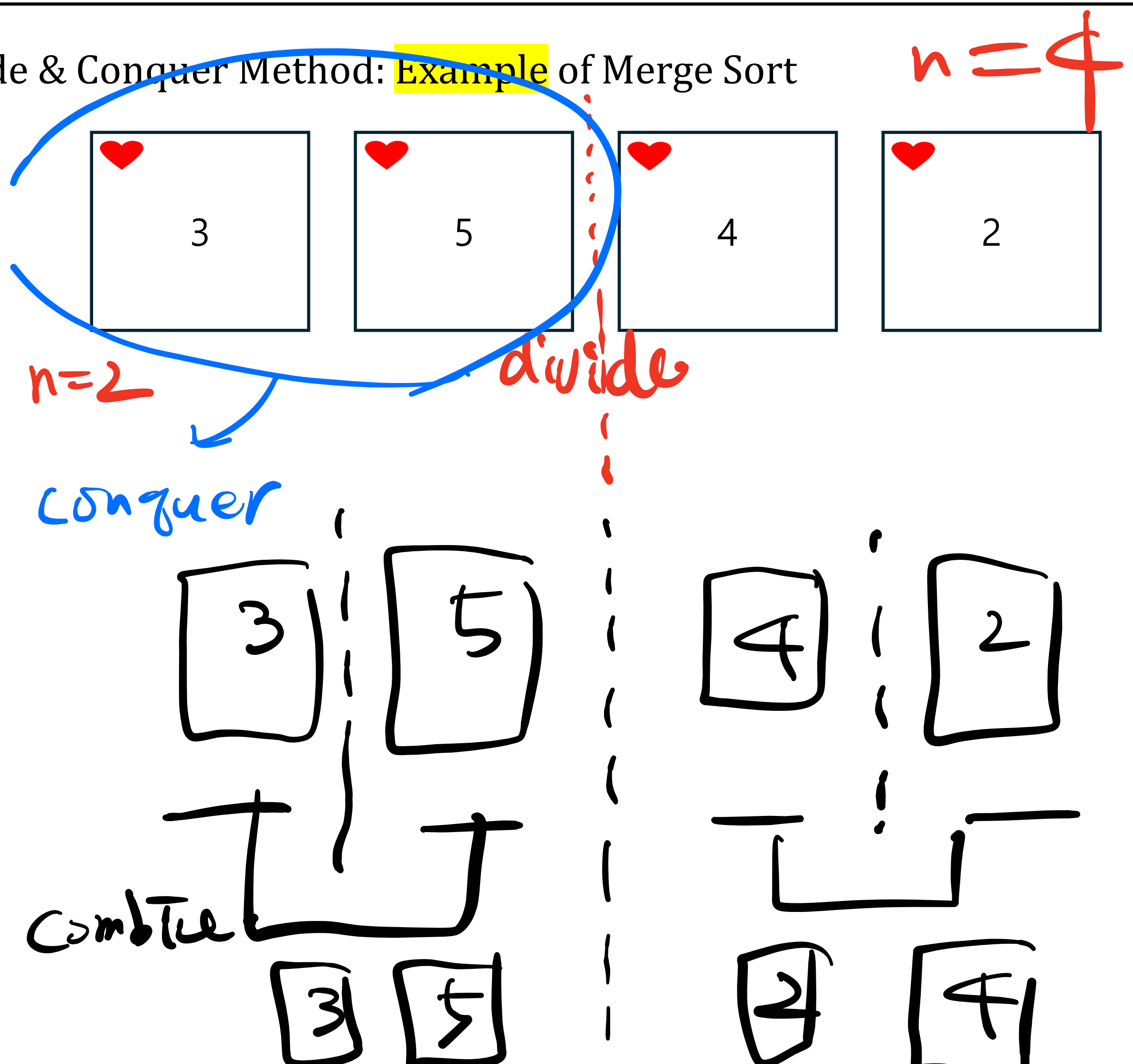
- 2.1 Insertion sort
- 2.2 Analyzing algorithms
- 2.3 Designing algorithms

2.3.1 Divide & Conquer Method



Incremental Approach:
Insertion Sort

2.3.1 Divide & Conquer Method: Example of Merge Sort



Number of Comparisons for $n = 4$

2 3 4 5

$n=4$

Algorithm	$number\ of\ comparsions$
Insertion Sort $\Theta(n^2)$	6
Merge Sort	5

$\Theta(n \log n)$

General problem solving strategy of Divide and Conquer Method

Divide the problem into one or more subproblems that are smaller instances of the same problem.

Conquer the subproblems by solving them recursively.

Combine the subproblem solutions to form a solution to the original problem.

Question?