Introduction to Algorithms

Date: 3/12 (Thursday)

Instructor: 유준수

Assignment

- Read 2.3
- Problems:
  - 2.3절 – 1, 4, 6

- Incremental Approach
  - Incrementing i=2 to n
- Correctness
  - State Loop Invariant
  - Check Initialization/Maintenance/Termination

Review of 2.2: Analyzing Algorithm

- Assumption: RAM Model
  1. One by one execution of instruction
  2. Instruction takes constant time (not dependent on input size n)
- Analyzing Insertion Sort

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

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

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

- t_i : number of while loop at i –th iteration
- Best-case time complexity
  - $t_i = 1\ for\ all\ i$

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

- Worst-case time complexity
  - $t_i = i\ for\ all\ i$

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

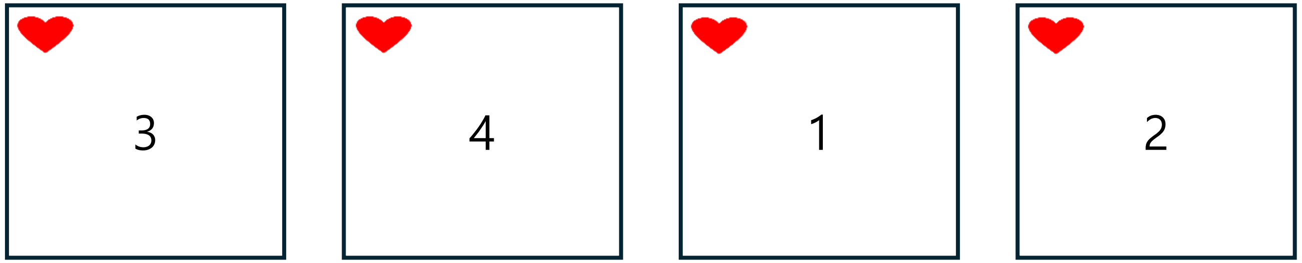
- Order of growth
  - Think of θ as “roughly proportional to” for now
  - Consider only the highest term
  - Ignore the constant term
  - E.g., $\theta(9n^3 + 2n^2 + 100) = \theta(n^3)$

Chapter 2. The Role of Algorithms in Computing

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

Divide & Conquer Method

- Divide
  - Divide the problem into sub-problem of the same kind
- Conquer
  - Solve the problem, recursively.
- Combine
  - Combine solutions to form original solution


3
4
1
2

Main Idea of Merge Sort Algorithm

Merge

- Combine two sorted arrays to output a sorted combined array

Back to Merge Sort Algorithm

How it works:

Example of n=4 and how it works

Complexity of Merge Process

```
MERGE(A, p, q, r)
n_L = q - p + 1          // length of A[p : q]
n_R = r - q              // length of A[q + 1 : r]
let L[0 : n_L - 1] and R[0 : n_R - 1] be new arrays
for i = 0 to n_L - 1     // copy A[p : q] into L[0 : n_L - 1]
    L[i] = A[p + i]
for j = 0 to n_R - 1     // copy A[q + 1 : r] into R[0 : n_R - 1]
    R[j] = A[q + j + 1]
i = 0                    // i indexes the smallest remaining element in L
j = 0                    // j indexes the smallest remaining element in R
k = p                    // k indexes the location in A to fill
// As long as each of the arrays L and R contains an unmerged element,
//     copy the smallest unmerged element back into A[p : r].
while i < n_L and j < n_R
    if L[i] ≤ R[j]
        A[k] = L[i]
        i = i + 1
    else A[k] = R[j]
        j = j + 1
    k = k + 1
// Having gone through one of L and R entirely, copy the
//     remainder of the other to the end of A[p : r].
while i < n_L
    A[k] = L[i]
    i = i + 1
    k = k + 1
while j < n_R
    A[k] = R[j]
    j = j + 1
    k = k + 1
```

Complexity of Merge Sort: Recurrence Relation

```
MERGE-SORT(A, p, r)
    if p ≥ r                          // zero or one element?
        return
    q = ⌊(p + r)/2⌋                   // midpoint of A[p : r]
    MERGE-SORT(A, p, q)               // recursively sort A[p : q]
    MERGE-SORT(A, q + 1, r)           // recursively sort A[q + 1 : r]
    // Merge A[p : q] and A[q + 1 : r] into A[p : r].
    MERGE(A, p, q, r)
```

$$T(n) = \begin{cases} T(1) = c_1 : & n = 1 \\ 2T(n/2) + \Theta(n) & \end{cases}$$

$$\Theta(n) = c_2 n$$

Complexity of Merge Sort: Time Complexity Calculation

Question?