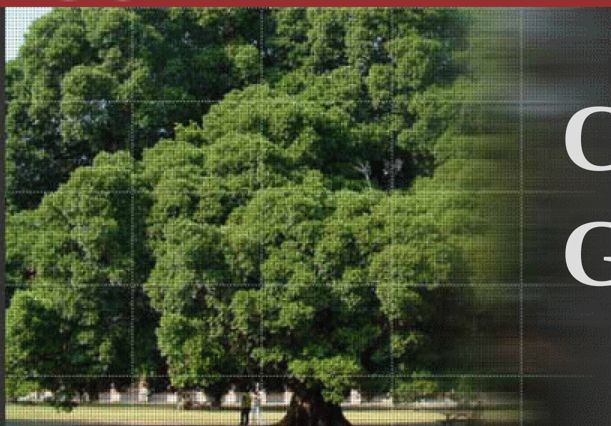
National Cheng Kung University



Chapter 2
Getting Started

Sun-Yuan Hsieh

謝孫源 教授

成功大學資訊工程學系

2.1 Insertion sort

- **Example:** Sorting problem
  - Input: A sequence of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$
  - Output: A permutation $\langle a_1', a_2', \ldots, a_n' \rangle$ of the input sequence such that $a_1' \le a_2' \le \cdots \le a_n'$
- The number that we wish to sort are known as the *keys*.

Pseudocode

- Insertion sort

```
Insertion-sort(A)
for j ← 2 to length[A]
  do key ← A[j]
      *Insert A[j] into the sorted sequence A[1,…, j – 1]
      i ← j – 1
      while i > 0 and A[i] > key
          do A[i + 1] ← A[i]
             i ← i – 1
      A[i + 1] ← key
```

The operation of Insertion-Sort

(a)

1	2	3	4	5	6
5	2	4	6	1	3

(b)

1	2	3	4	5	6
2	5	4	6	1	3

(c)

1	2	3	4	5	6
2	4	5	6	1	3

(d)

1	2	3	4	5	6
2	4	5	6	1	3

(e)

1	2	3	4	5	6
1	2	4	5	6	3

(f)

1	2	3	4	5	6
1	2	3	4	5	6

- ***Sorted in place:***
  - The numbers are rearranged within the array A, with at most a constant number of them sorted outside the array at any time.
- ***Loop invariant:***
  - At the start of each iteration of the for loop of line 1-8, the subarray $A[1,\ldots, j-1]$ consists of the elements originally in $A[1,\ldots, j-1]$ but in sorted order.

2.2 Analyzing algorithms

- Analyzing an algorithm has come to mean predicting the resources that the algorithm requires.
  - ***Resources:*** memory, communication, bandwidth, logic gate, **time**.
  - **Assumption:** one processor, ***RAM***
  - constant-time instruction: **arithmetic** (add, subtract, multiply, divide, remainder, floor, ceiling); **data movement** (load, store, copy); **control** (conditional and unconditional bramch, subroutine call and return)
  - Date type: integer and floating point
  - Limit on the size of each word of data

2.2 Analyzing algorithms

- The best notion for **input size** depends on the problem being studied.
- The **running time** of an algorithm on a particular input is the number of primitive operations or “steps” executed. It is convenient to define the notion of step so that it is as machine-independent as possible.

Analysis of insertion sort

Insertion-sort(A)	cost	cost
1. **for** $j \leftarrow 2$ **to** length[A]	c_1	n
2. **do** key $\leftarrow A[j]$	c_2	$n-1$
3. *Insert $A[j]$ into the sorted sequence $A[1,\ldots, j-1]$	0	
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_j: the number of times the while loop test in line 5 is executed for the value of j.

Analysis of insertion sort

- The running time

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

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

- $t_j = 1$, for $j = 2, 3, \ldots, n$: Linear function on n

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

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

Analysis of insertion sort

- $t_j = j$, for $j = 2, 3, \ldots, n$: Quadratic function on n

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

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

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

$$- (c_2 + c_4 + c_5 + c_8)$$

Worst-case and average-case analysis

- Usually, we concentrate on finding only on the ***worst-case running time***.
- Reason:
  - It is an upper bound on the running time.
  - The worst case occurs fair often.
  - The average case is often as bad as the worst case. For example, the insertion sort. Again, quadratic function.

Order of growth

- In some particular cases, we shall be interested in ***average-case***, or ***expect*** running time of an algorithm.

- It is the ***rate of growth***, or ***order of growth***, of the running time that really interests us.

2.3 Designing algorithms

- There are many ways to design algorithms:
  - **Incremental approach:** insertion sort
  - **Divide-and-conquer:** merge sort
    - recursive:
      - divide
      - conquer
      - combine

Pseudocode

- Merge sort

Merge(A, p, q, r)

1. $n_1 \leftarrow q - p + 1$

2. $n_2 \leftarrow r - q$

3. create array $L[1,\ldots, n_1 + 1]$ and $R[1,\ldots, n_2 + 1]$

4. **for** $i \leftarrow 1$ **to** n_1

5.   **do** $L[i] \leftarrow A[p + i - 1]$

6. **for** $j \leftarrow 1$ **to** n_2

7.   **do** $R[j] \leftarrow A[q + j]$

8. $L[n_1 + 1] \leftarrow \infty$

9. $R[n_2 + 1] \leftarrow \infty$

Pseudocode

$i \leftarrow 1$

$j \leftarrow 1$

for $k \leftarrow p$ **to** r

    do if $L[i] \leq R[j]$

        then $A[k] \leftarrow L[i]$

            $i \leftarrow i + 1$

        else $A[k] \leftarrow R[j]$

            $j \leftarrow j + 1$

The operation of lines 10-17

	8	9	10	11	12	13	14	15	16	17
A	…	2	4	5	7	1	2	3	6	…
		k								

	1	2	3	4	5
L	2	4	5	7	∞
	i				

	1	2	3	4	5
R	1	2	3	6	∞
	j				

(a)

The operation of lines 10-17

	8	9	10	11	12	13	14	15	16	17
A	…	1	4	5	7	1	2	3	6	…
			k							

	1	2	3	4	5
L	2	4	5	7	∞
	i				

	1	2	3	4	5
R	1	2	3	6	∞
		j			

(b)

The operation of lines 10-17

	8	9	10	11	12	13	14	15	16	17
A	…	1	2	5	7	1	2	3	6	…
				k						

	1	2	3	4	5
L	2	4	5	7	∞
		i			

	1	2	3	4	5
R	1	2	3	6	∞
		j			

(c)

The operation of lines 10-17

	8	9	10	11	12	13	14	15	16	17
A	…	1	2	2	7	1	2	3	6	…
					k					

	1	2	3	4	5
L	2	4	5	7	∞
		i			

	1	2	3	4	5
R	1	2	3	6	∞
			j		

(d)

The operation of lines 10-17

	8	9	10	11	12	13	14	15	16	17
A	…	1	2	2	3	1	2	3	6	…
						k				

	1	2	3	4	5
L	2	4	5	7	∞
		i			

	1	2	3	4	5
R	1	2	3	6	∞
				j	

(e)

The operation of lines 10-17

	8	9	10	11	12	13	14	15	16	17
A	…	1	2	2	3	4	2	3	6	…
							k			

	1	2	3	4	5
L	2	4	5	7	∞
			i		

	1	2	3	4	5
R	1	2	3	6	∞
				j	

(f)

The operation of lines 10-17

	8	9	10	11	12	13	14	15	16	17
A	…	1	2	2	3	4	5	3	6	…
								k		

	1	2	3	4	5
L	2	4	5	7	∞
				i	

	1	2	3	4	5
R	1	2	3	6	∞
				j	

(g)

The operation of lines 10-17

	8	9	10	11	12	13	14	15	16	17
A	…	1	2	2	3	4	5	6	6	…
									k	

	1	2	3	4	5
L	2	4	5	7	∞
				i	

	1	2	3	4	5
R	1	2	3	6	∞
					j

(h)

The operation of lines 10-17

	8	9	10	11	12	13	14	15	16	17
A	…	1	2	2	3	4	5	6	7	…
										k

	1	2	3	4	5
L	2	4	5	7	∞
					i

	1	2	3	4	5
R	1	2	3	6	∞
					j

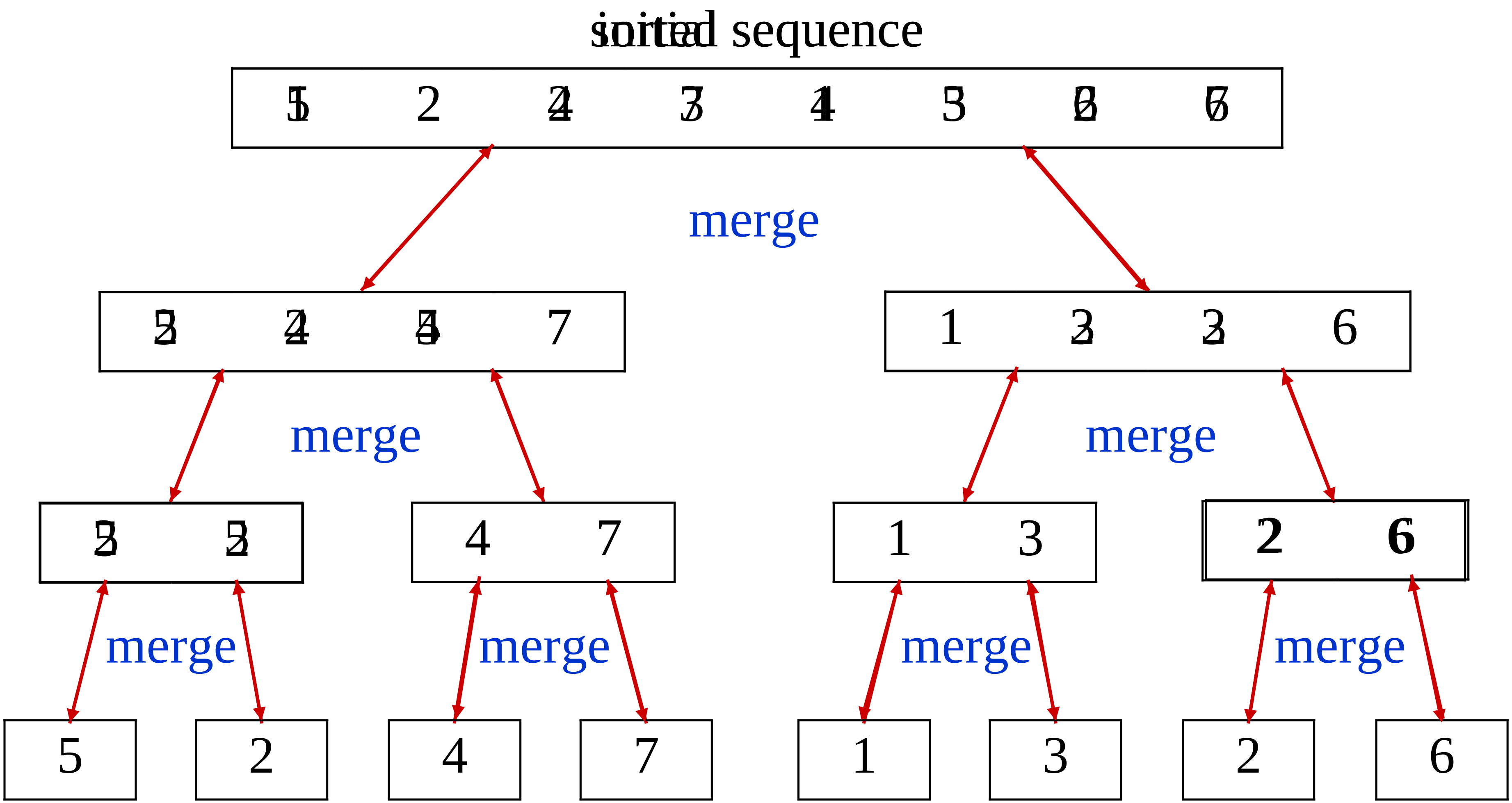
(i)

Pseudocode

MERGE-SORT(A, p, r)

1. **if** $p < r$
2. **then** $q \leftarrow \lfloor (p + r)/2 \rfloor$
3. MERGE-SORT(A, p, q)
4. MERGE-SORT(A, $q + 1$, r)
5. MERGE(A, p, q, r)

The operation of Merge sort



Analysis of Merge sort

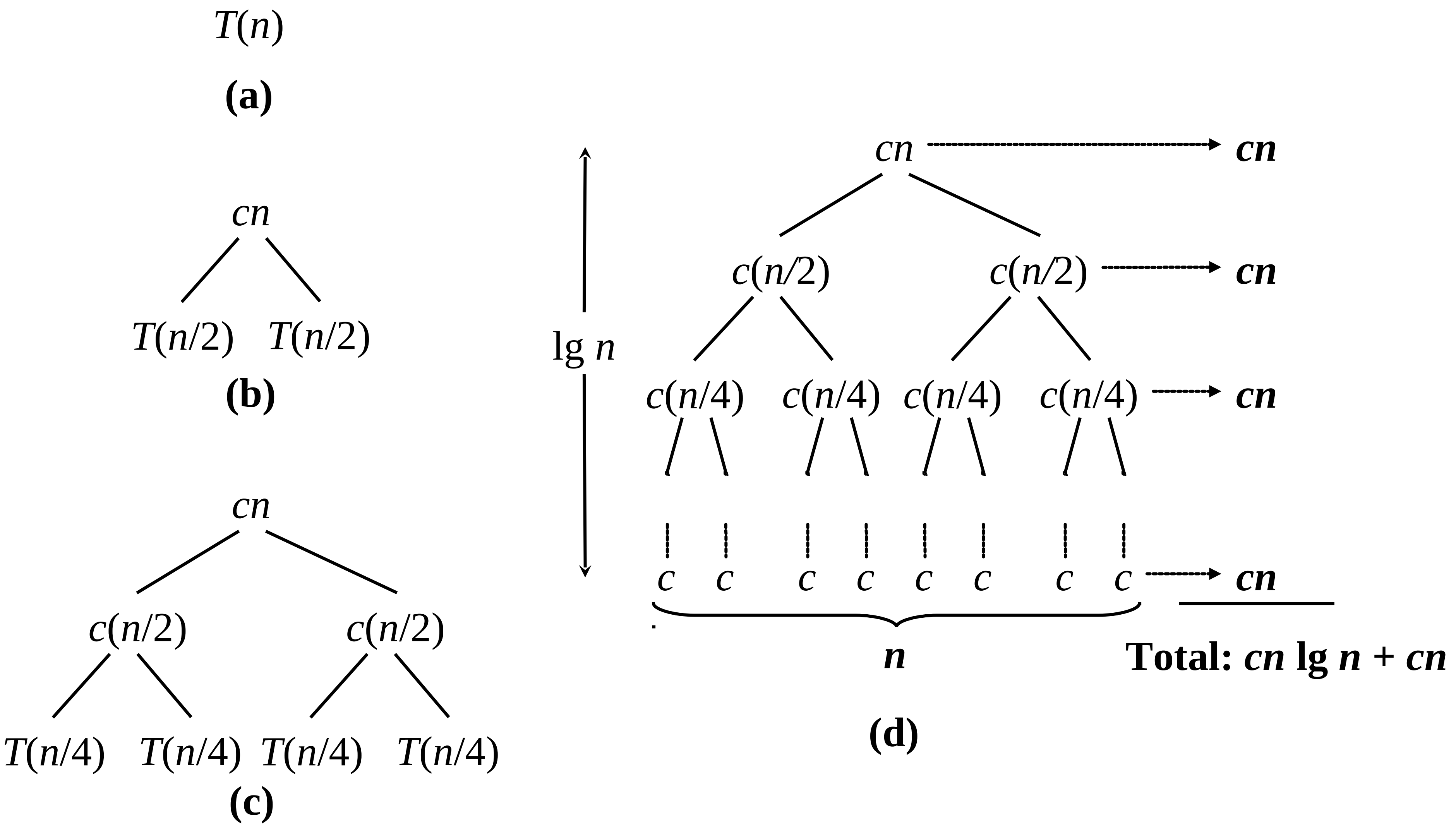
- Analyzing divide-and-conquer algorithms
  - $$T(n) = \begin{cases} \Theta(1) & \text{if } n \le c \\ aT(n/b) + D(n) + C(n) & \text{otherwise} \end{cases}$$
  - See Chapter 4.
- Analysis of merge sort
  - $$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$
  - $T(n) = \Theta(n \log n)$

Analysis of Merge sort

- $$T(n) = \begin{cases} c & \text{if } n = 1 \\ 2T(n/2) + cn & \text{if } n > 1 \end{cases}$$

  where the constant c represents the time require to solve problem of size 1 as well as the time per array element of the divide and combine steps.

The construction of a recursion tree



- Outperforms insertion sort!