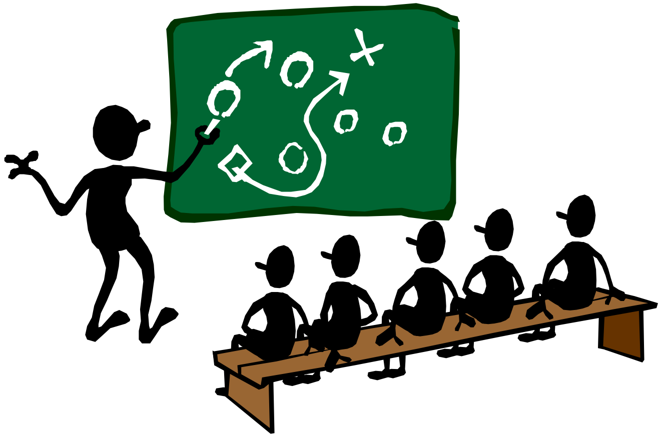
Algorithms Chapter 1 & 2
Getting Started



Juinn-Dar Huang

Professor

jdhuang@mail.nctu.edu.tw

July 2007

Rev. '08, '11, '12, '15, '16, '18, '19, '20

Algorithms

- Algorithm
  - is any well-defined computational procedure
  - takes some value or set of values as input
  - produces some value or set of values as output

  ➔ is a sequence of computational steps that transfer the input into the output
- An algorithm is a tool for solving a well-specified computational problem
- The problem statement
  - specifies the desired input/output relationship

  ➔ the algorithm is a specific procedure for achieving that I/O relationship

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Sorting Problem

- Sorting problem
  - input: a sequence of n numbers $< a_1, a_2, \ldots, a_n >$
  - output: a permutation $< a_1', a_2', \ldots, a_n' >$ of the input sequence such that $a_1' \le a_2' \le \ldots \le a_n'$
- For example
  - input: <31, 41, 59, 26, 41, 58>

    an instance of the problem
  - output: <26, 31, 41, 41, 58, 59>
- An instance of the problem
  - consists of the input needed to compute a solution (output) to the problem

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Correctness of Algorithm

- Correct algorithm
  - halts with the correct output for every input instance
- A correct algorithm solves the given computational problem
- Incorrect algorithm
  - might halt with an incorrect output
  - might not halt for some input instances

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Beyond the Correctness

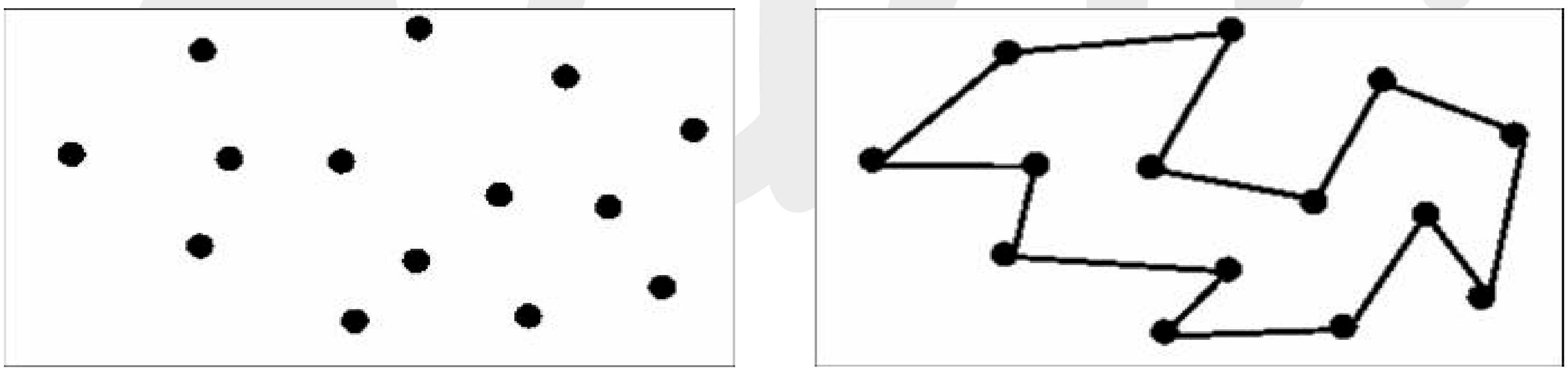
- Make the algorithm correct first
- Reality
  - computers are not infinitely fast
  - memory is not infinitely large
- In addition to correctness, a good algorithm should be efficient in both time and space

- Correctness vs. Efficiency ?

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Traveling Salesman Problem (TSP)

- **Input:** A set of points (cities) *P* together with a distance *d*(*p*, *q*) between any pair *p*, *q* ∈ *P*
- **Output:** What is the shortest circular route that starts and ends at a given point and visits all the points



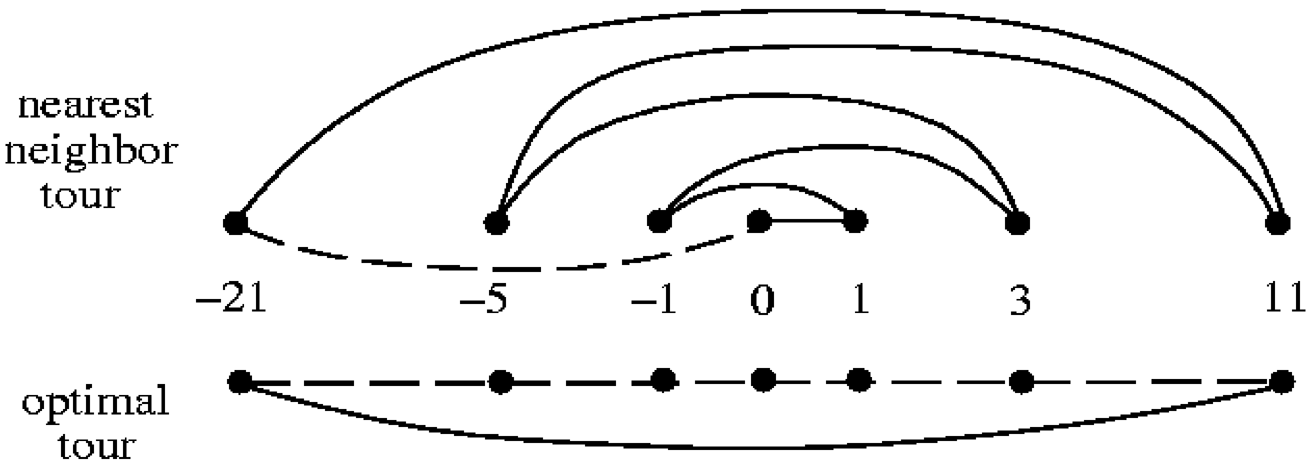
- Correct and efficient algorithms?

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Nearest Neighbor Tour

1. pick and visit an initial point p_0
2. $P \leftarrow p_0$
3. $i \leftarrow 0$
4. **while** there are unvisited points **do**
5. visit p_i's nearest unvisited point p_{i+1}
6. $i \leftarrow i + 1$
7. return to p_0 from p_i

- Simple to implement and very efficient, but **incorrect!**



A Correct but Inefficient Algorithm

1. $d \leftarrow \infty$
2. for each of the n! permutations π_i of the n points
3.   **if** (cost(π_i) $\leq d$) **then**
4.   d $\leftarrow$ cost(π_i)
5. $T_{min} \leftarrow \pi_i$
6. **return** T_{min}

- **Correctness?** Try all possible orderings of the points ➔ Guarantee to end up with the shortest possible tour
- **Efficiency?** Try ***n!*** possible routes!
  - 120 routes for 5 points; 3,628,800 routes for 10 points
  - no known efficient and correct algorithm for TSP!
  - TSP is **NP-complete**

- Think Deep: What will you do?

Getting Started

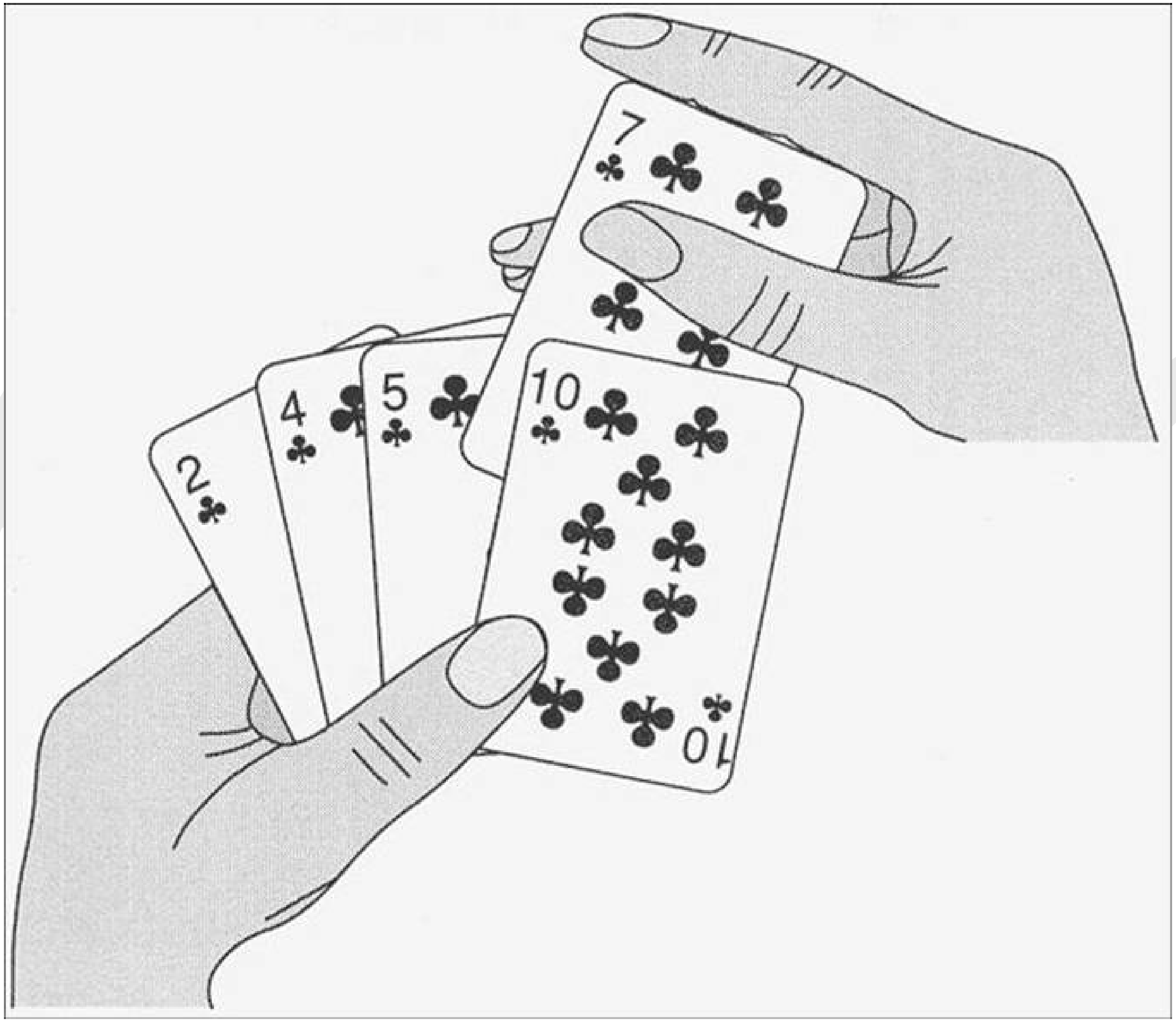
Sorting Problem

- Sorting problem
  - input: a sequence of n numbers $< a_1, a_2, \ldots, a_n >$
  - output: a permutation $< a_1', a_2', \ldots, a_n' >$

    of the input sequence such that $a_1' \le a_2' \le \ldots \le a_n'$
- For example
  - input: <31, 41, 59, 26, 41, 58>
  - output: <26, 31, 41, 41, 58, 59>
- The numbers under sorting are also known as keys
- How to sort?

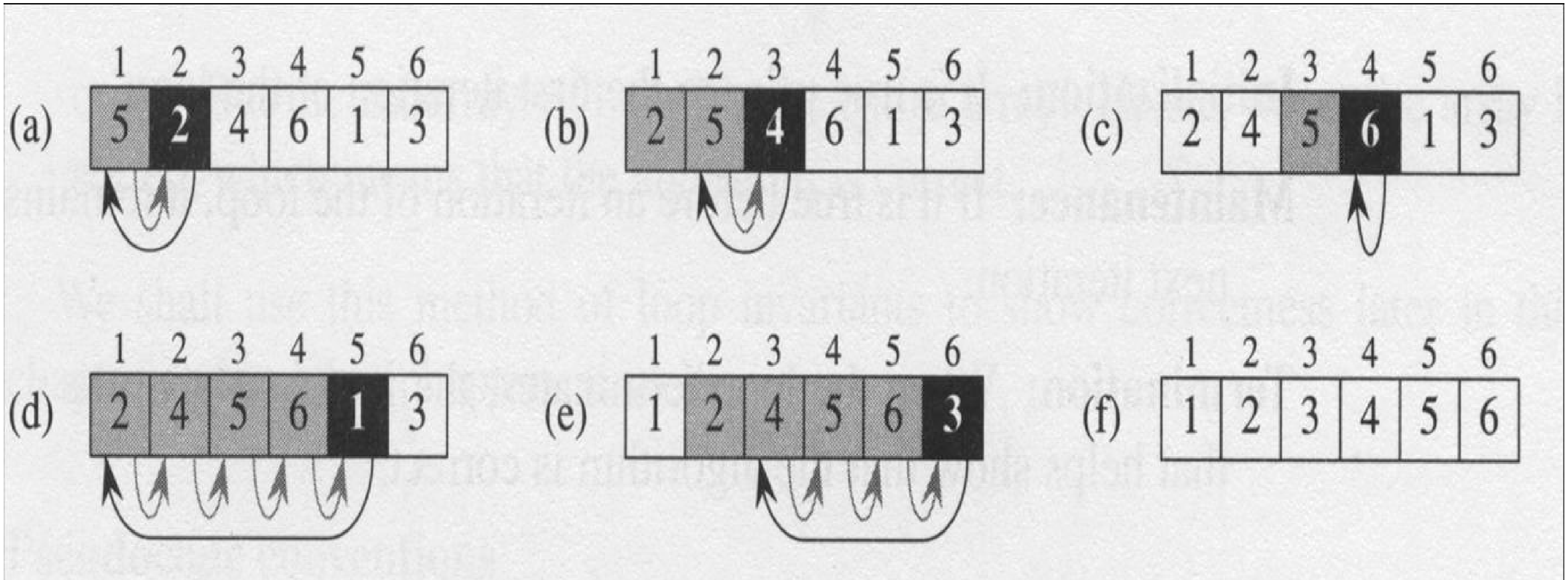
Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Sorting a Hand of Cards



Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Insertion Sort



Insertion sort is an efficient algorithm for sorting a small number of elements

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
```

Juinn-Dar Huang
jdhuang@mail.nctu.edu.tw

Analyzing Algorithms

- Analyzing an algorithm has come to mean predicting the resources that the algorithm requires
  - CPU time
  - memory space
- Assumption
  - single processor
    (not multiprocessor, not parallel computing)
  - RAM as memory (not disk)

Juinn-Dar Huang
jdhuang@mail.nctu.edu.tw

Running Time vs. Input Size

- The running time needed by Insertion-Sort depends on the input size
  - 10 numbers vs. 1 million numbers
- In general
  - input size ⬆ ➔ running time ⬆
  - the running time is typically described as a function of the input size
- How to calculate running time?
  - count the number of primitive operations (steps)
  - machine-independent

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Analysis of Insertion Sort

Insertion-sort(A)	cost	times
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 .. 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

Best-Case Analysis

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

- T(n) does not solely depend on n
- Best case: the array is already sorted
  - $t_j = 1$ for j = 2, 3, …, n

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

  - T(n) is a **linear** function of n

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Worst-Case Analysis

- Worst case: the array is sorted in reverse order
  - $t_j = j$

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

  - T(n) is a **quadratic** function of n

Juinn-Dar Huang   jdhuang@mail.nctu.edu.tw

Best/Worst/Average Cases

- We usually concentrate on finding the worst-case running time
  - the longest running time for any input size of n
- Why?
  - worst case gives an upper bound
  - for some algorithms, the worst case occurs fairly *often*
    - e.g., searches in databases
  - the average case is *often* roughly as bad as the worst case
    - e.g., for insertion sort, T(n) is still a quadratic function of n

Juinn-Dar Huang
jdhuang@mail.nctu.edu.tw

Order of Growth

- Why is the running time typically described as a function of the input size?
  - to know how the running time grows as the input size grows
- It is the rate of growth, or order of growth, of the running time that really interests us
- For insertion sort
  - the worst-case running time, $\Theta(n^2)$
- Assume Algo1 with $\Theta(n^2)$ and Algo2 with $\Theta(n^3)$
  - Algo1 is considered more efficient than Algo2
  - i.e., for a large enough n, Algo1 runs faster than Algo2

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Divide-and-Conquer

- Many algorithms are recursive
  - they call themselves recursively one or more times to deal with subproblems
- Divide-and-conquer paradigm
  - **divide** the problem into a number of subproblems
  - **conquer** the subproblems by solving them recursively
  - **combine** the solutions to the subproblems into the solution to the original problem

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Merge Sort

- Divide-and-conquer
  - divide: divide the n-element sequence into 2 subsequences of n/2 elements each
  - conquer: sort the 2 subsequences recursively
  - combine: merge the 2 sorted subsequences to produce the answer

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

MERGE(A, p, q, r) (1/2)

- MERGE(A, p, q, r)
  - given that A[p .. q] and A[q+1 .. r] are 2 sorted subarrays
  - MERGE(A, p, q, r) produce a sorted A[p .. r] subarray

$n_1 \leftarrow q - p + 1$

$n_2 \leftarrow r - q$

create array L[1 .. $n_1 + 1$] and R[1 .. $n_2 + 1$]

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

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

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

    do R[i] $\leftarrow$ A[$q + j$]

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

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

sentinels

MERGE(A, p, q, r) (2/2)

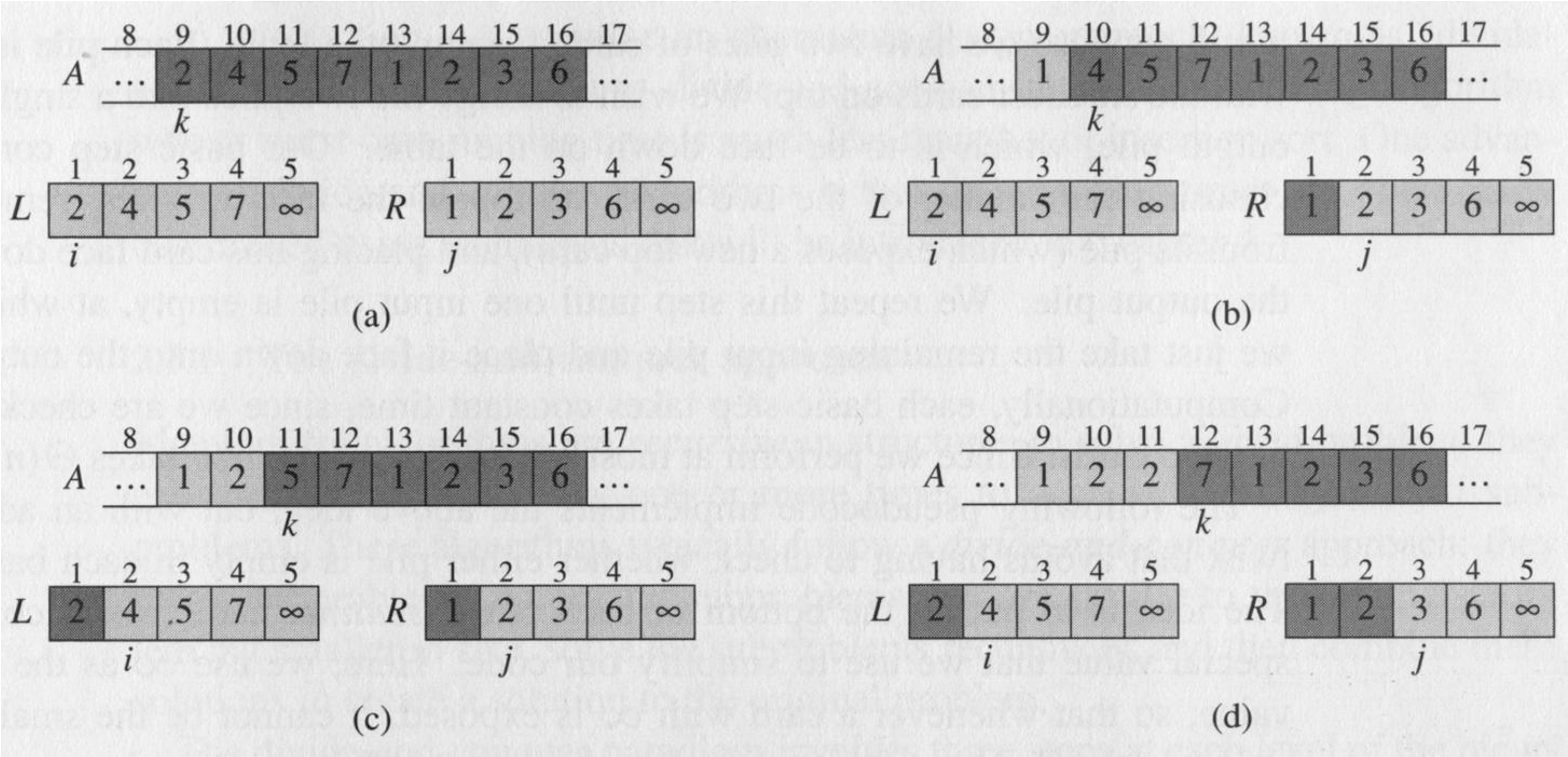
Time: Θ(n), n = number of elements

```
    i ← 1
    j ← 1
for k ← p to r
    do if L[ i ] ≤ R[ j ]
        then A[ k ] ← L[ i ]
             i ← i + 1
    else A[ k ] ← R[ j ]
             j ← j + 1
```

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

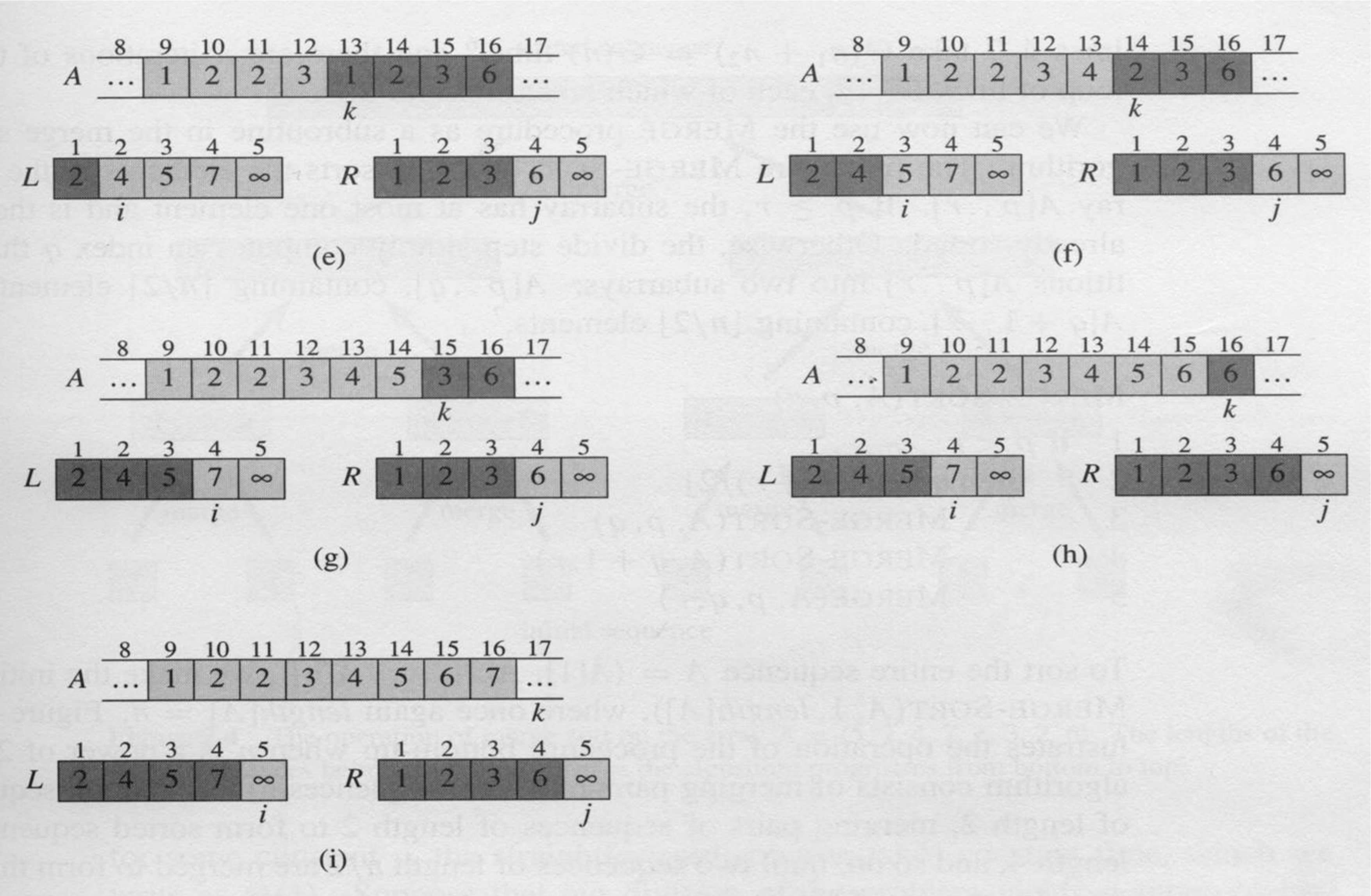
Illustration of MERGE (1/2)

MERGE(A, 9, 12, 16)



Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Illustration of MERGE (2/2)



Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Merge Sort

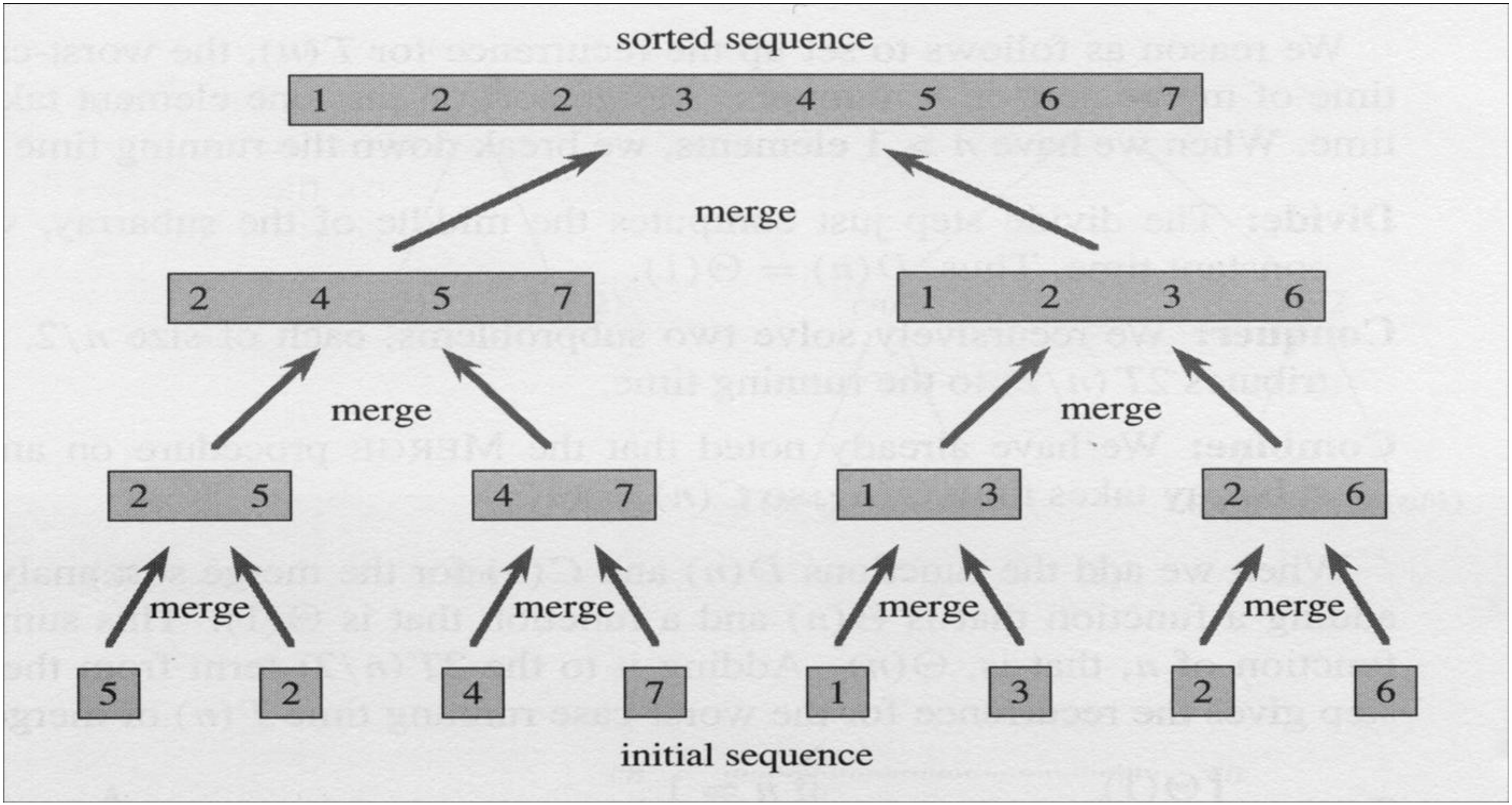
- MERGE-SORT(A, p, r)
  - sort the elements in A[p .. r]

---

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

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Illustration of MERGE-SORT



Analysis of Merge Sort (1/2)

- The running time of a recursive algorithm can be expressed as a recurrence equation
  - discuss later in Chap 4
  - "master theorem" can be found in discrete math
- Analysis of merge sort

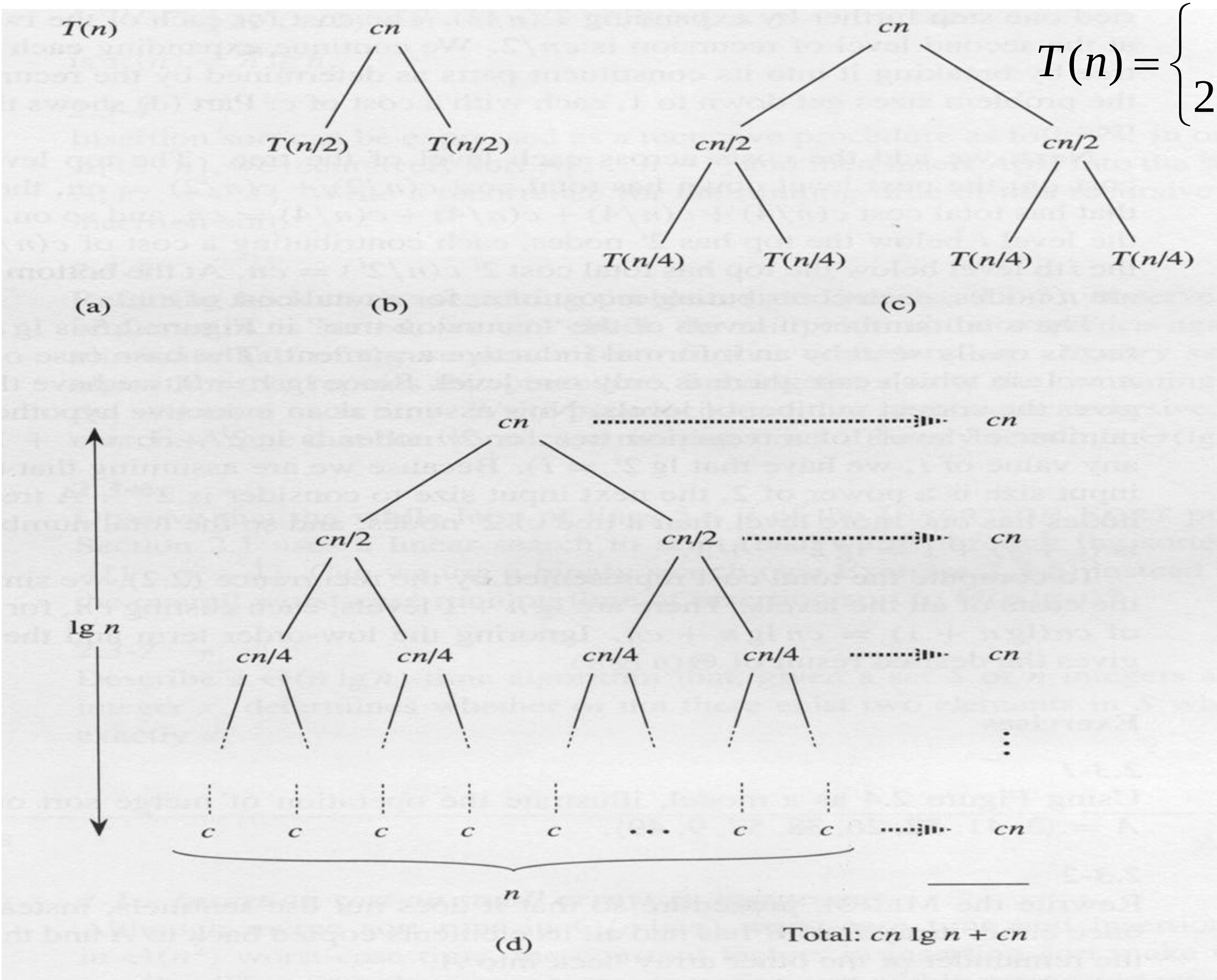
$$T(n) = \begin{cases} \Theta(1) & \text{if} \quad n = 1 \\ 2T(n/2) + \Theta(n) & \text{if} \quad n > 1 \end{cases}$$

➡ $T(n) = \Theta(n \lg n)$ by the master theorem

Hence the merge sort is better than the insertion sort when n is large

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Analysis of Merge Sort (2/2)



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

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw