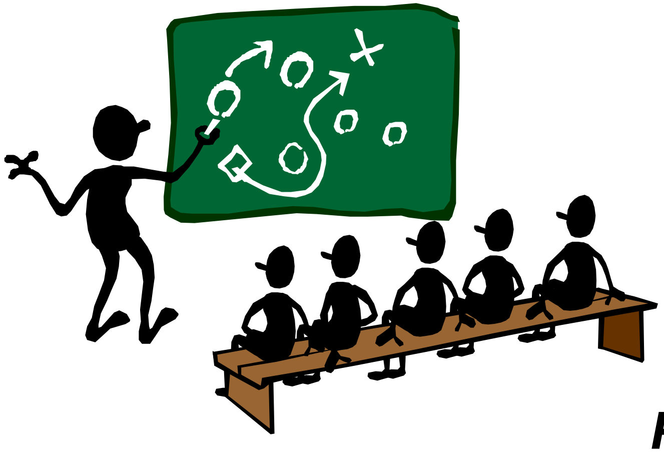
Algorithms – Chapter 7
Quick Sort



Juinn-Dar Huang

Professor

jdhuang@mail.nctu.edu.tw

August 2007

Rev. ’08, ’11, ’12, ’15, ’16, ’18, ’19, ’20, ’21

Divide-and-Conquer Paradigm

- Like Merge Sort, Quick Sort is based on divide-and-conquer paradigm
- Divide
  - partition the array A[p..r] into 2 (possibly empty) subarrays A[p..q–1] and A[q+1..r] such that
  - each element of A[p..q–1] ≤ A[q] and each element of A[q+1..r] > A[q]
- Conquer
  - sort the 2 subarrays by Quick Sort recursively
- Combine
  - nothing to do here

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

Pseudo Code

QUICKSORT(A,p,r)

1 **if** $p < r$

2 **then** $q \leftarrow PARTITION(A, p, r)$

3 QUICKSORT($A,p,q-1$)

4 QUICKSORT($A,q+1,r$)

To sort an entire array A, use QUICKSORT(A, 1, length(A))

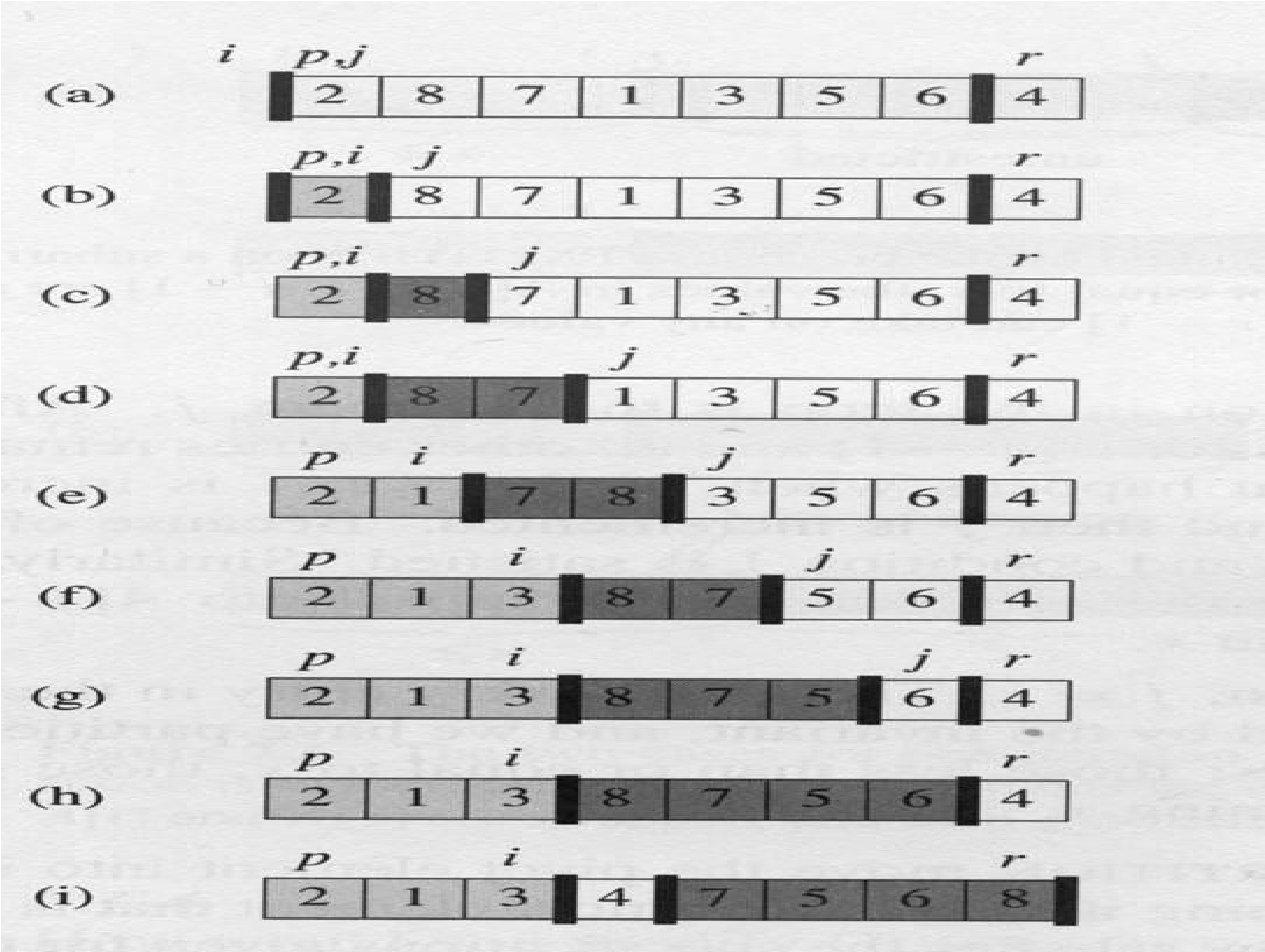
Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

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

How to Partition

```
PARTITION(A, p, r)
x ← A[r]                          * as a pivot
i ← p – 1
for j ← p to r – 1
    do if  A[j] ≤ x
          then  i ← i + 1
                exchange A[i] ↔ A[j]
exchange A[i + 1] ↔ A[r]
return  i + 1
```

Example

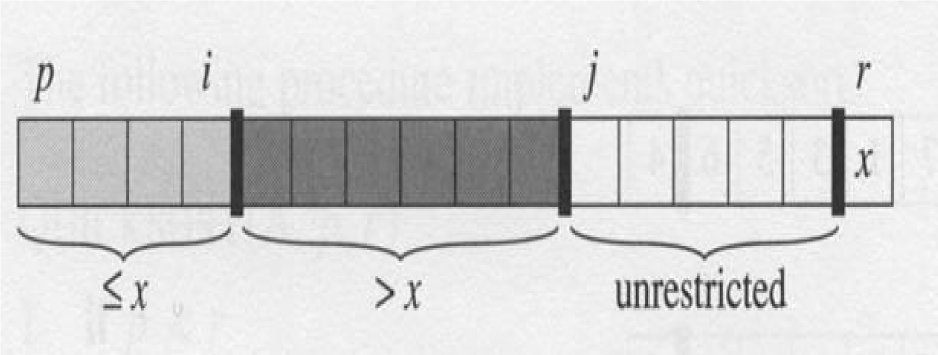


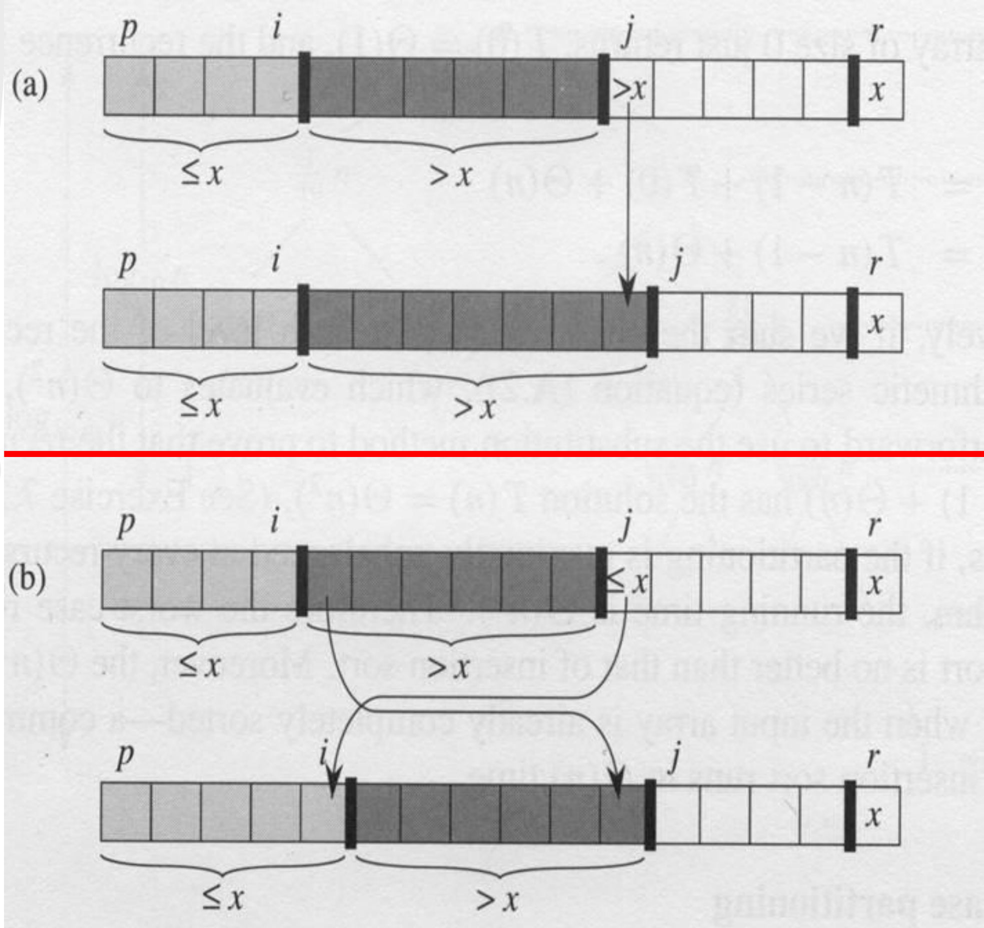
(a)
i
p,j
r
2 8 7 1 3 5 6 4
(b)
p,i
j
r
2 8 7 1 3 5 6 4
(c)
p,i
j
r
2 8 7 1 3 5 6 4
(d)
p,i
j
r
2 8 7 1 3 5 6 4
(e)
p
i
j
r
2 1 7 8 3 5 6 4
(f)
p
i
j
r
2 1 3 8 7 5 6 4
(g)
p
i
j
r
2 1 3 8 7 5 6 4
(h)
p
i
r
2 1 3 8 7 5 6 4
(i)
p
i
r
2 1 3 4 7 5 6 8

Loop Invariant

- At the beginning of each iteration of the loop of lines 3-6, for any array index k,

1. if p ≤ k ≤ i, then A[k] ≤ x.
2. if i + 1 ≤ k ≤ j – 1, then A[k] > x.
3. if k = r, then A[k] = x.





Time Complexity of Partition

```
PARTITION(A, p, r)
x ← A[r]                              * as a pivot
i ← p – 1
for j ← p to r – 1
     do if  A[j] ≤ x
              then  i ← i + 1
                    exchange A[i] ↔ A[j]
exchange A[i + 1] ↔ A[r]
return  i + 1
```

Time complexity : Θ(n), where n = r – p + 1

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

Worst Case of Quick Sort

- The runtime of Quick Sort depends on whether the partitioning is balanced or not

- Worst-case partitioning
  - partitioning produces one subarray with n-1 elements and the other subarray with 0 element

$$T(n) = T(n-1) + \Theta(n)$$

$$= \sum_{k=1}^{n} \Theta(k) = \Theta\left(\sum_{k=1}^{n} k\right) = \boxed{\Theta(n^2)}$$

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

Best Case of Quick Sort

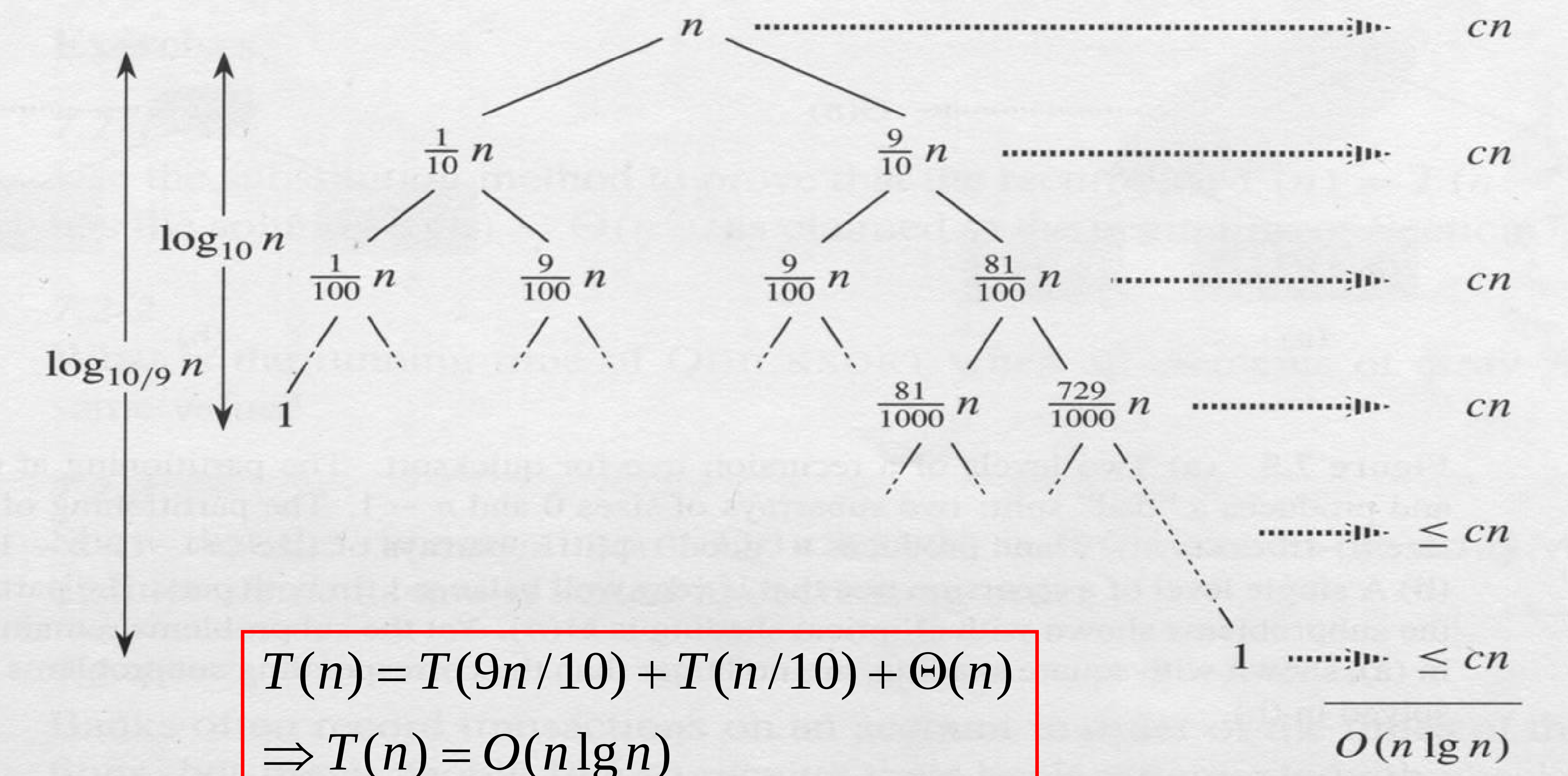
- Best-case partitioning
  - partitioning produces 2 subarrays, one is of size $\lfloor n/2 \rfloor$ and the other of size $\lceil n/2 \rceil - 1$
  - perfect balance

$$T(n) = 2T(n/2) + \Theta(n)$$

$$\Rightarrow T(n) = \boxed{\Theta(n \lg n)}$$

Not a Perfect Balance?

- With a 9-to-1 split
  - how about 99-to-1 split?



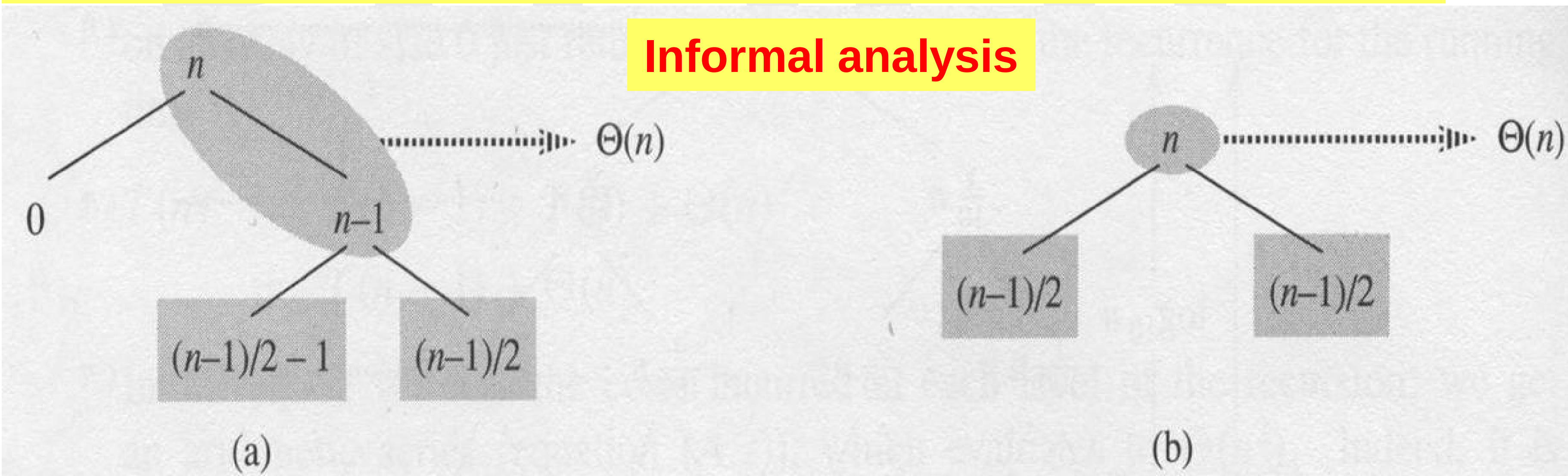
$$T(n) = T(9n/10) + T(n/10) + \Theta(n)$$
$$\Rightarrow T(n) = O(n \lg n)$$

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

Average Case of Quick Sort

- Assume all permutations are equally likely
- It is expected
  – some partitioning are well balanced
  – some are fairly unbalanced

The average case performance is similar to that of the best case



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

Randomized Quick Sort

RANDOMIZED_PARTITION(A,p,r)

1 $i \leftarrow RANDOM(p,r)$

2 exchange $A[r] \leftrightarrow A[i]$

3 **return** PARTITION(A,p,r)

random sampling

The split of the input array should be reasonably well balanced on average no matter what the initial input order is

RANDOMIZED_QUICKSORT(A,p,r)

1 **if** $p < r$

2 **then** $q \leftarrow RANDOMIZED\_PARTITION(A,p,r)$

3 RANDOMIZED_QUICKSORT($A,p,q-1$)

4 RANDOMIZED_QUICKSORT($A,q+1,r$)

Precise Worst-Case Analysis

$$T(n) = \max_{0 \le q \le n-1} (T(q) + T(n-q-1)) + \Theta(n)$$

guess $T(n) \le cn^2$

$$T(n) \le \max_{0 \le q \le n-1} (cq^2 + c(n-q-1)^2) + \Theta(n)$$

$$= c \max_{0 \le q \le n-1} (q^2 + (n-q-1)^2) + \Theta(n)$$

$$\le cn^2 - c(2n-1) + \Theta(n)$$

$$\le cn^2$$

max value @ q = 0 or n-1

pick the constant c large enough so that the $c(2n-1)$ term dominates the $\Theta(n)$ term.

$\Rightarrow T(n) = \Theta(n^2)$

Expected Runtime – Precise Analysis

- Each time *PARTITON* is called
  - a pivot element *p* is selected
  - *p* is never a pivot again for the future *PARTITON* calls
  - ➔ at most n calls to *PARTITION* in *QUICKSORT*

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

Precise Analysis (1/5)

- ***Lemma***

  Let X be the TOTAL number of comparisons performed in Line 4 of *PARTITION* over the entire execution of *Quick Sort* on an n-element array. Then the runtime of *Quick Sort* is $O(n+X)$

PARTITION(A, p, r)
1. $x \leftarrow A[r]$
2. $i \leftarrow p - 1$
3. **for** $j \leftarrow p$ **to** $r - 1$ **do**
4.   **if** $A[j] \leq x$
5.   **then** $i \leftarrow i+1$
6.   exchange $A[i] \leftrightarrow A[j]$
7. exchange $A[i+1] \leftrightarrow A[r]$
8. **return** $i+1$

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

Precise Analysis (2/5)

- Assume A contains elements $z_1, z_2, \ldots, z_n$; with z_i being the i-th smallest element
- Define the set $Z_{ij} = \{z_i, z_{i+1}, \ldots, z_j\}$, $i < j$
- Fact
  - each pair of elements is compared **at most once**

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

Precise Analysis (3/5)

- Define X_{ij} = I {z_i is compared to z_j}

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}$$

$$E[X] = E\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}\right]$$

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E\left[X_{ij}\right]$$

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \text{ is compared to } z_j\}$$

Precise Analysis (4/5)

z_i and z_j are compared iff the first element to be chosen as a pivot from Z_{ij} is either z_i or z_j

$$\begin{aligned}\Pr\{z_i \text{ is compared to } z_j\} &= \Pr\{z_i \text{ or } z_j \text{ is first pivot chosen from } Z_{ij}\} \\ &= \Pr\{z_i \text{ is first pivot chosen from } Z_{ij}\} \\ &+ \Pr\{z_j \text{ is first pivot chosen from } Z_{ij}\} \\ &= \frac{1}{j-i+1} + \frac{1}{j-i+1} \\ &= \frac{2}{j-i+1}\end{aligned}$$

hard to understand!

$$\therefore E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

Precise Analysis (5/5)

$$E[X]=\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\frac{2}{j-i+1}$$

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

(k = j – i)

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

$$=\sum_{i=1}^{n-1}O(\lg n)$$

$$\sum_{k=1}^{n}\frac{1}{k}=\ln n+O(1)$$

$$=O(n\lg n)$$

For RANDOMIZED-PARTITION, the expected runtime of Quick Sort is O(nlgn)

Further Topics

- Problem 7-1 (p185)
  - the original version of Quick Sort
- Problem 7-4 (p188)
  - consider about the stack
- Problem 7-5 (p188)
  - median-of-3 partition
  - a way to improve RANDOMIZED-QUICKSORT

Comparison Sorts

- Comparison sorts
  – the sorted order they determined is based only on comparisons between the input elements
  – e.g., merge/heap/quick/insertion/bubble/shell sort

- Heap Sort and Merge Sort achieve O(nlgn) in the worst case

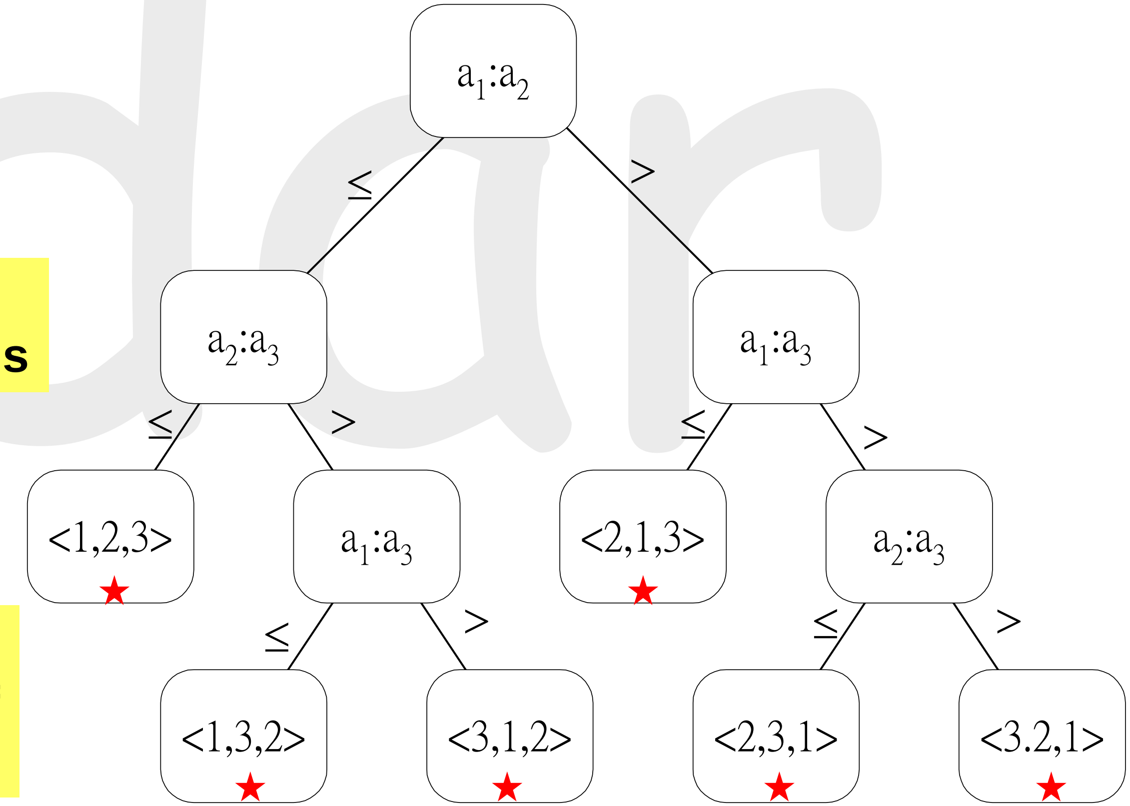
- Is it possible to find algorithms with even lower time complexity?

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

Decision Trees

- The number of comparisons in the worst case
  - the length of the longest path from the root of a decision tree to any of its reachable leaves

3! = 6 leaf nodes
➔6 possible initial permutations



Decision tree for the insertion sort on an array of 3 elements

Lower Bound for the Worst Case

- A decision tree of height h with l leaves
  - ➔ $n! = l \le 2^h$
  - ➔ $h \ge \lg(n!) = \Omega(n\lg n)$
- Theorem
  - any comparison sort algorithm requires $\Omega(n\lg n)$ comparisons in the worst case
- Heap Sort and Merge Sort are asymptotically optimal comparison sorts

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