北京航空航天大學
COLLEGE OF SOFTWARE BEIHANG UNIVERSITY 软件学院

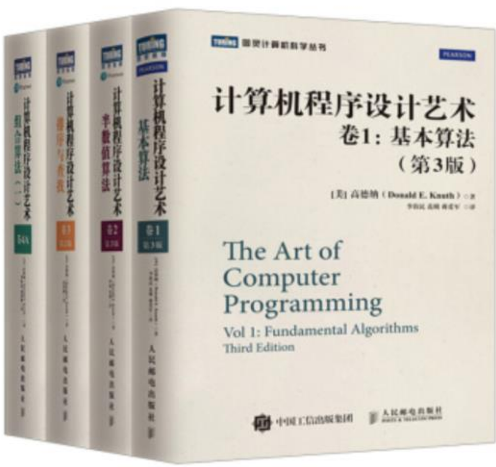
Part II

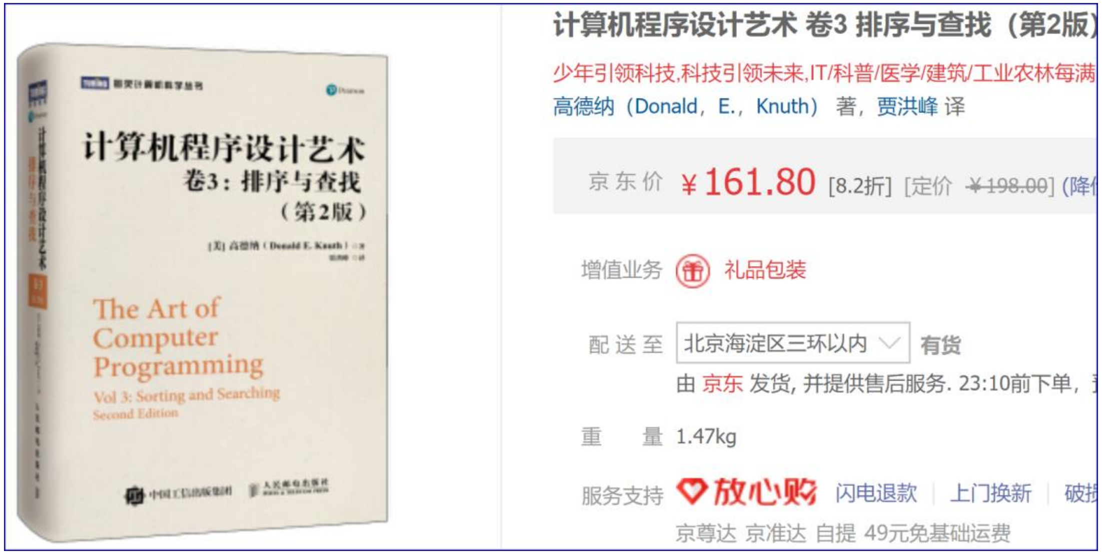
Sorting and Order Statistics

北京航空航天大學
COLLEGE OF SOFTWARE BEIHANG UNIVERSITY 软件学院

千万不要觉得排序简单！

Sorting and Order Statistics





Sorting and Order Statistics

- **Heapsort**
  - **Maintaining the heap property**
  - **Building a heap**
  - **The heapsort algorithm**
  - **Priority queues**
- **Quicksort**
  - **Description and Performance**
  - **A randomized version of quicksort**
  - **Analysis of quicksort**
- **Lower bounds for sorting**
- **Medians and Order Statistics**
  - **Minimum and maximum**
  - **Selection in expected linear time**

6 Heapsort

```
HEAPSORT(A)
BUILD-MAX-HEAP(A)
for i = A.length downto 2
    exchange A[1] with A[i]
    A.heap-size = A.heap-size − 1
    MAX-HEAPIFY(A, 1)
```

调整二叉树的元素位置，使其为最大堆

```
BUILD-MAX-HEAP(A)
A.heap-size = A.length
for i = ⌊A.length/2⌋ downto 1
    MAX-HEAPIFY(A, i)
```

调整二叉树的元素位置，使其为最大堆

```
MAX-HEAPIFY(A, i)
l = LEFT(i)
r = RIGHT(i)
if l ≤ A.heap-size and A[l] > A[i]
    largest = l
else largest = i
if r ≤ A.heap-size and A[r] > A[largest]
    largest = r
if largest ≠ i
    exchange A[i] with A[largest]
    MAX-HEAPIFY(A, largest)
```

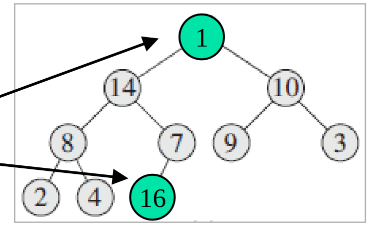
调整二叉树的元素位置，使其为最大堆

1) 建堆（得到最大堆）

2) 交换元素（最大元素定位）

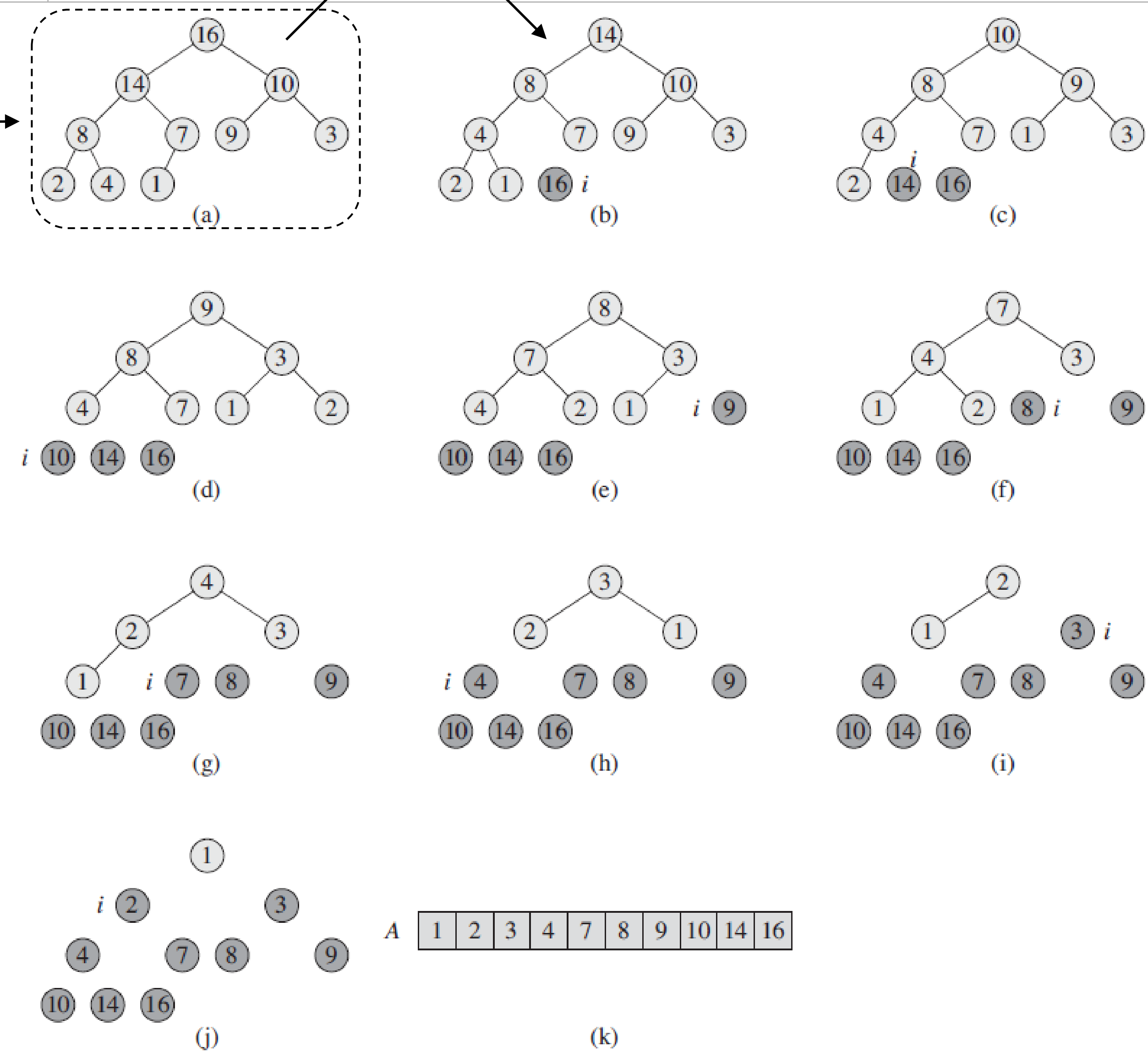
3) 最大堆维护（子序列中【除去上一步的最大元素】，最大堆性质被破坏了），然后回到第3步

```
HEAPSORT(A)
BUILD-MAX-HEAP(A)
for i = A.length downto 2
    exchange A[1] with A[i]
    A.heap-size = A.heap-size − 1
    MAX-HEAPIFY(A, 1)
```



1 Build: 建立一个最大堆
2 交换堆中的首尾元素
3 维护最大堆

```
BUILD-MAX-HEAP(A)
A.heap-size = A.length
for i = ⌊A.length/2⌋ downto 1
    MAX-HEAPIFY(A, i)
```



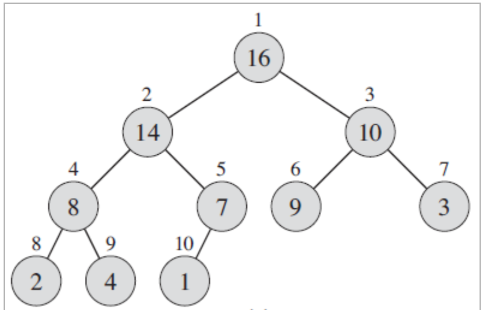

```
MAX-HEAPIFY(A, i)
l = LEFT(i)
r = RIGHT(i)
if l ≤ A.heap-size and A[l] > A[i]
    largest = l
else largest = i
if r ≤ A.heap-size and A[r] > A[largest]
    largest = r
if largest ≠ i
    exchange A[i] with A[largest]
    MAX-HEAPIFY(A, largest)
```

6 Heapsort

- **Running time: $\Theta(n \lg n)$**
- **Using a data structure "heap" to manage information**
- **Not only is the heap data structure useful for heapsort, but it also makes an efficient priority queue**

Applications: we use min-heaps to implement min-priority queues in Chapters 16 (Greedy Algorithms), 23 (Minimum Spanning Trees), and 24 (Single-Source Shortest Paths).

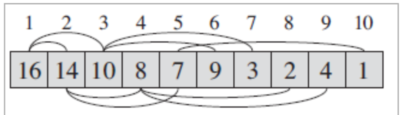
6.1 Heaps



(a)

- **(binary) heap : complete binary tree**

  (priority queue)
- ***A*: An array can represent a heap**



(b)

- ***A.length* : the number of elements in the array;**

  ***A.heap-size* : the number of elements in the heap**

  (0 ≤ *A.heap-size* ≤ *A.length*)

PARENT(i)
1 **return** $\lfloor i/2 \rfloor$

LEFT(i)
1 **return** $2i$

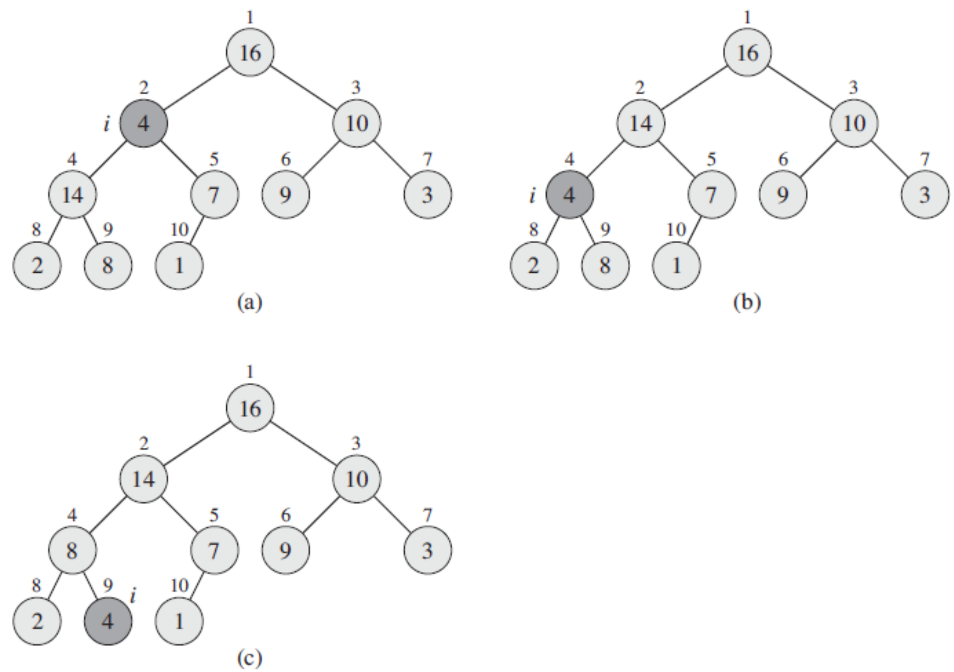
RIGHT(i)
1 **return** $2i + 1$

- **max-heap : *A*[PARENT(*i*)] ≥ *A*[*i*], for every node *i* other than the root.**
- **min-heap : ?**

6.2 Maintaining the heap property

- **MAX-HEAPIFY assumes that the trees rooted at LEFT(*i*) and RIGHT(*i*) are max-heaps, but that *A*(*i*) might be smaller than its children, thus violating the max-heap property.**
- **MAX-HEAPIFY lets the value at *A*(*i*) "float down" in the max-heap.**

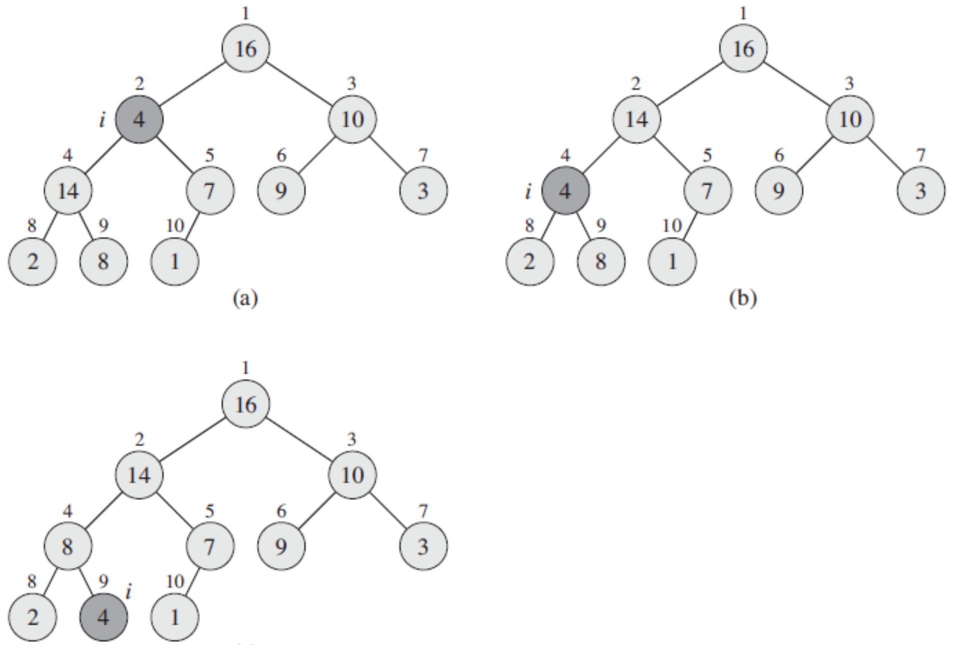
```
MAX-HEAPIFY(A, i)
    l = LEFT(i)
    r = RIGHT(i)
    if l ≤ A.heap-size and A[l] > A[i]
        largest = l
    else largest = i
    if r ≤ A.heap-size and A[r] > A[largest]
        largest = r
    if largest ≠ i
        exchange A[i] with A[largest]
        MAX-HEAPIFY(A, largest)
```



- **The running time?**

6.2 Maintaining the heap property

```
MAX-HEAPIFY(A, i)
  l = LEFT(i)
  r = RIGHT(i)
  if l ≤ A.heap-size and A[l] > A[i]
        largest = l
  else largest = i
  if r ≤ A.heap-size and A[r] > A[largest]
        largest = r
  if largest ≠ i
        exchange A[i] with A[largest]
        MAX-HEAPIFY(A, largest)
```



(a) (b) (c)

- **The running time?**

  For node *i* , the children's subtrees each have size at most 2*n*/3—the worst case occurs when the bottom level of the tree is exactly half full.

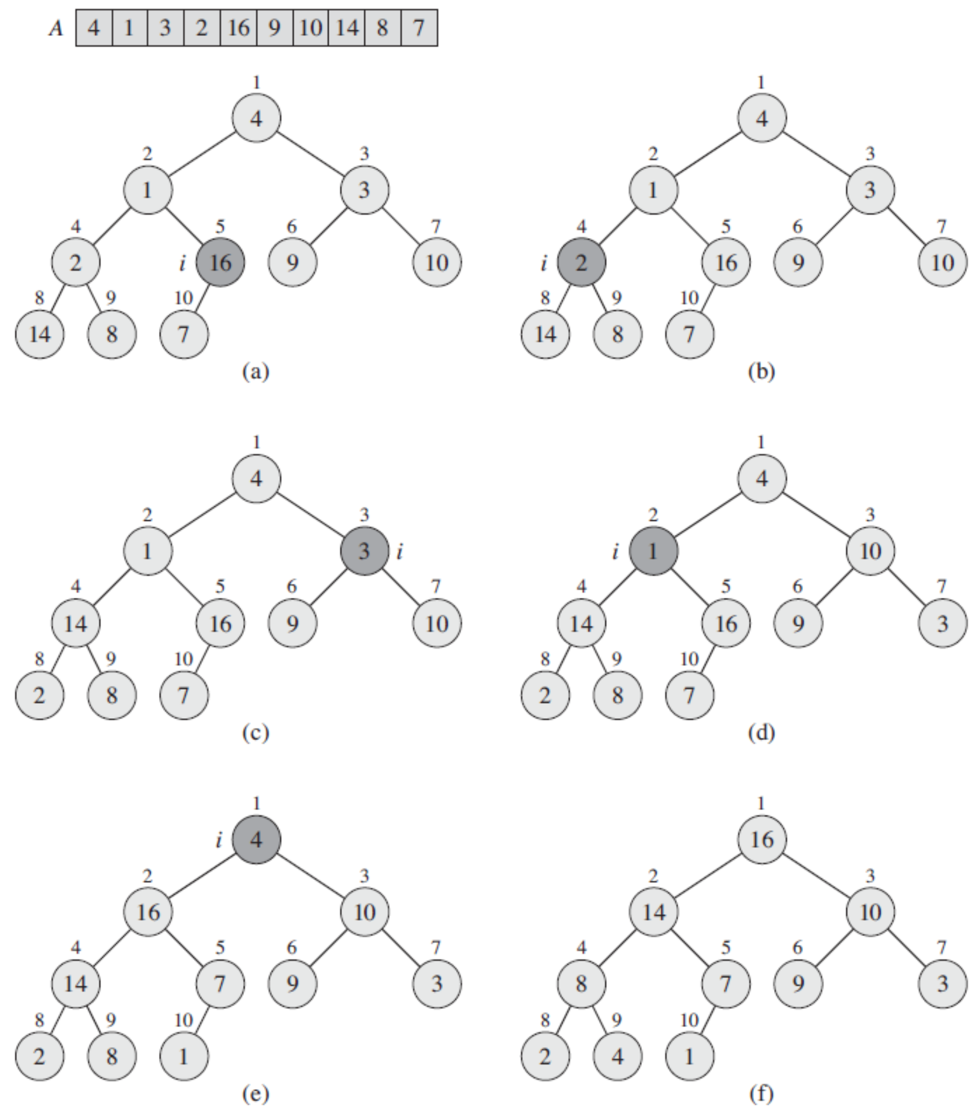
  $T(n) \le T(2n/3) + \Theta(1)$ **Answer ? Master method.**

- **In fact, the running time of MAX-HEAPIFY on a node of height *h* is *O*(*h*).**

6.3 Building a Heaps

- We use the procedure MAX-HEAPIFY in a bottom-up manner to convert an array $A[1 \,..\, n]$ into a max-heap.
- The elements in the subarray $A[(\text{floor}(n/2) + 1) \,..\, n]$ are all leaves of the tree, and so each is a 1-element heap to begin with. (Exercise 6.1-7)

```
BUILD-MAX-HEAP(A)
A.heap-size = A.length
for i = ⌊A.length/2⌋ downto 1
    MAX-HEAPIFY(A, i)
```



- Correct ?
  Loop invariant.
- The running time?

6.3 Building a Heaps

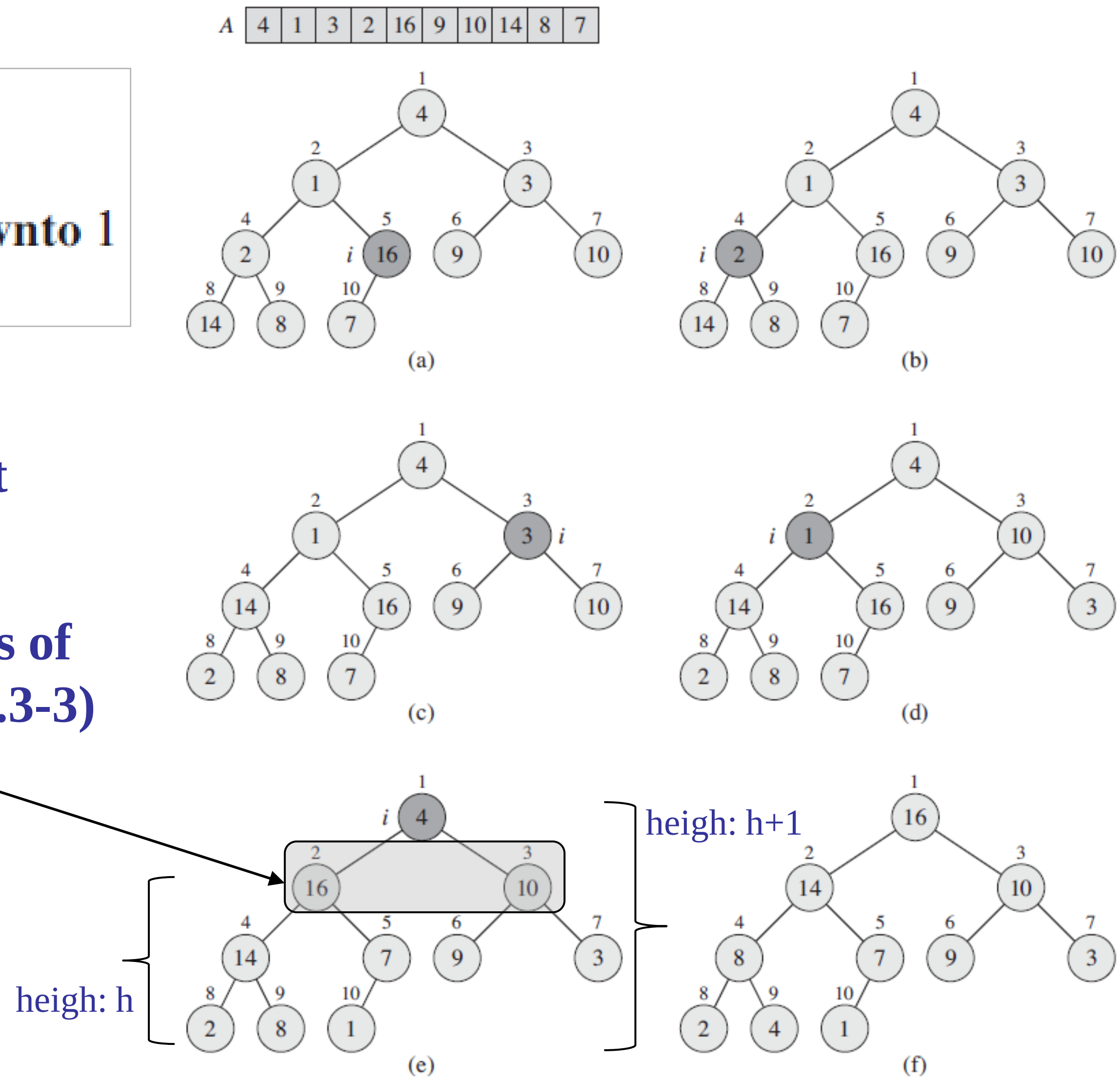
BUILD-MAX-HEAP(A)

```
A.heap-size = A.length
for i = ⌊A.length/2⌋ downto 1
    MAX-HEAPIFY(A, i)
```

- **The running time?** **O($n$lg$n$), correct, but not asymptotically tight.**
- **at most ceil($n/2^{h+1}$) nodes of any height h (Exercise 6.3-3)**

$$\sum_{h=0}^{\lfloor \lg n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h) = O\left(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h}\right)$$

$$O\left(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h}\right) = O\left(n \sum_{h=0}^{\infty} \frac{h}{2^h}\right) = O(n) .$$



$O(n)$

在高度h的地方，最多有 $n/(2^{(h+1)})$ 个节点，
每个节点调用MAX-HEAPIFY时花的时间为$O(h)$，
高度从0到 lg$n$.

6.3 Building a Heaps

$$\sum_{h=0}^{\lfloor \lg n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h) = O\left(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h}\right)$$

$$O\left(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h}\right) = O\left(n \sum_{h=0}^{\infty} \frac{h}{2^h}\right) = O(n) .$$

$$\sum_{h=0}^{\infty} \frac{h}{2^h} = 2 .$$

大学知识：A.8 .

中学知识：

let: $\sum_{h=0}^{\infty} \frac{h}{2^h} = t$

So: $\sum_{h=0}^{\infty} \frac{h}{2^h} = \sum_{h=0} \frac{h-1+1}{2^h} = \sum_{h=0} \left(\frac{1}{2^h} + \frac{h-1}{2^h}\right)$

$= \sum_{h=0} \frac{1}{2^h} + \frac{1}{2} \sum_{h=0} \frac{h-1}{2^{h-1}}$

$= 2 + \frac{1}{2}\left(-2 + \sum_{h=0} \frac{h}{2^h}\right)$

$= 2 + \frac{1}{2}(-2 + t) = t$

$\Rightarrow t = 2$

6.4 The heapsort algorithm

```
HEAPSORT(A)
    BUILD-MAX-HEAP(A)
    for i = A.length downto 2
        exchange A[1] with A[i]
        A.heap-size = A.heap-size − 1
        MAX-HEAPIFY(A, 1)
```

n

$O(n)$

```
BUILD-MAX-HEAP(A)
    A.heap-size = A.length
    for i = ⌊A.length/2⌋ downto 1
        MAX-HEAPIFY(A, i)
```

$O(n\lg n)$

$O(h)$

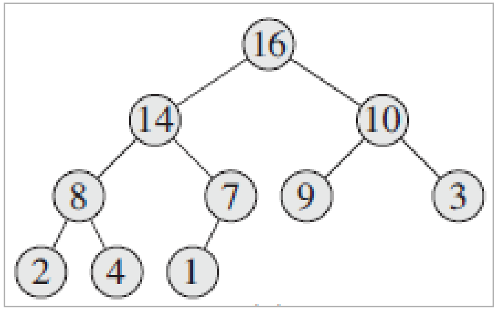
```
MAX-HEAPIFY(A, i)
    l = LEFT(i)
    r = RIGHT(i)
    if l ≤ A.heap-size and A[l] > A[i]
        largest = l
    else largest = i
    if r ≤ A.heap-size and A[r] > A[largest]
        largest = r
    if largest ≠ i
        exchange A[i] with A[largest]
        MAX-HEAPIFY(A, largest)
```

- **The running time of Heapsort ?**

6.5 Priority queues

- **One of the most popular applications of a heap:**
  - **An efficient priority queue (max-priority, min-priority)**
- **Applications:**

6.5 Priority queues



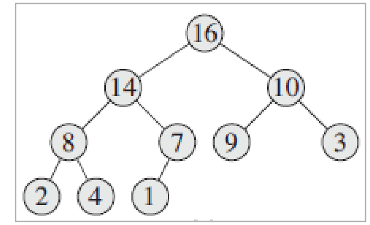
- **One of the most popular applications of a heap:**

  An efficient priority queue (max-priority, min-priority)

- **A max-priority queue supports the following operations:**
  - **MAXIMUM(*S*)**: returns the element of S with the largest key.
  - **EXTRACT-MAX**(S): removes and returns the element of S with the largest key.
  - **INCREASE-KEY**(S, x, k): increases the value of element x's key to the new value k, which is assumed to be at least as large as x's current key value.
  - **INSERT(*S*,** x): inserts the element x into the set S, which is equivalent to the operation $S = S \cup \{x\}$.

- **Applications:**
  - **A max-priority queue can be used to schedule jobs on a shared computer.**
  - **A min-priority queue can be used in an event-driven simulator.**

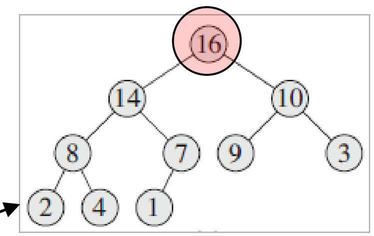
Operations of a max-priority queue



HEAP-MAXIMUM(A)

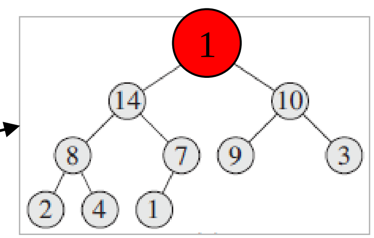
return $A[1]$

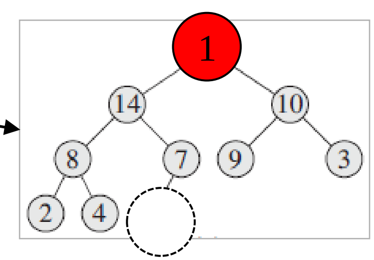
$\Theta(1)$



HEAP-EXTRACT-MAX(A) **$O(\lg n)$**

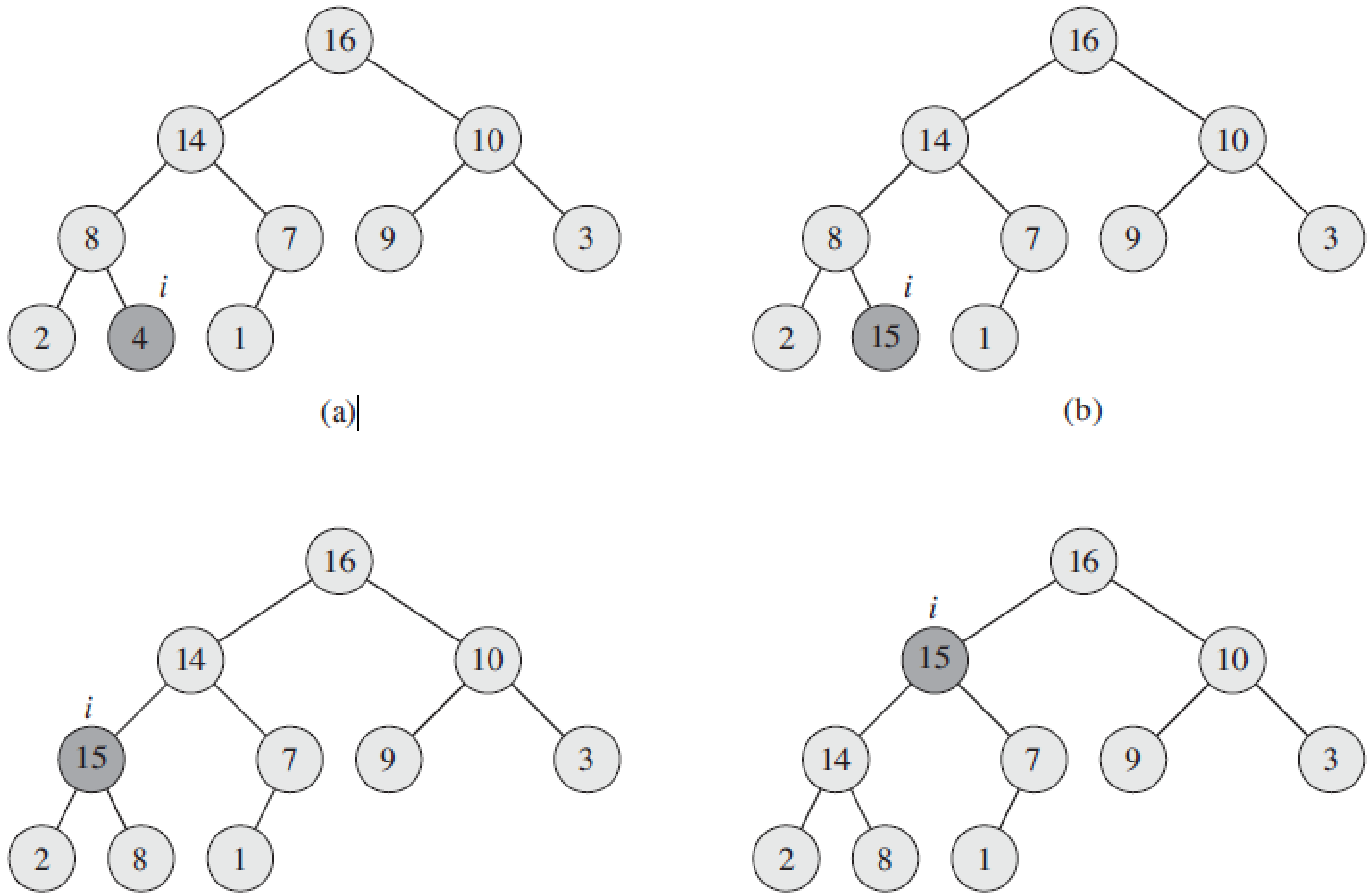
if $A.heap\text{-}size < 1$
    error "heap underflow"
$max = A[1]$
$A[1] = A[A.heap\text{-}size]$
$A.heap\text{-}size = A.heap\text{-}size - 1$
MAX-HEAPIFY$(A, 1)$
return max





```
HEAP-INCREASE-KEY(A, i, key)
if key < A[i]
    error "new key is smaller than current key"
A[i] = key
while i > 1 and A[PARENT(i)] < A[i]
    exchange A[i] with A[PARENT(i)]
    i = PARENT(i)
```

O(lg$n$)



(a) (b) (c) (d)

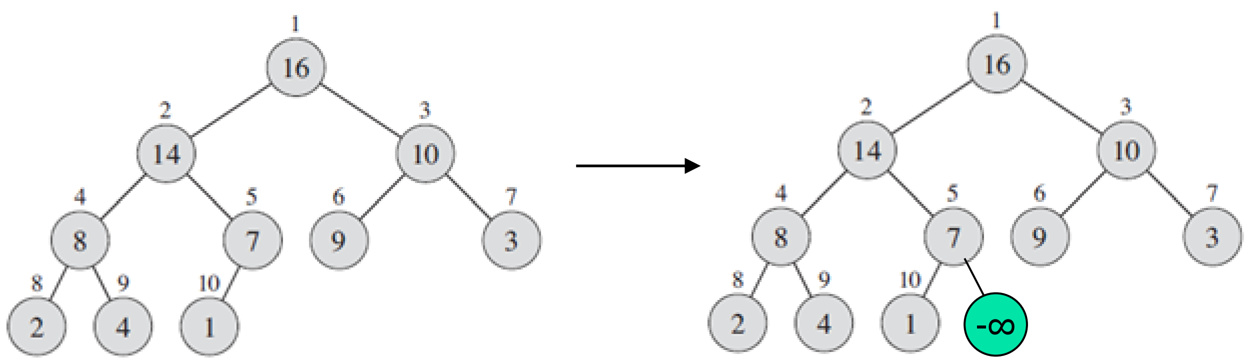
Operations of a max-priority queue

```
HEAP-INCREASE-KEY(A, i, key)
if key < A[i]
    error "new key is smaller than current key"
A[i] = key
while i > 1 and A[PARENT(i)] < A[i]
    exchange A[i] with A[PARENT(i)]
    i = PARENT(i)
```

O(lg*n*)

```
MAX-HEAP-INSERT(A, key)
A.heap-size = A.heap-size + 1
A[A.heap-size] = −∞
HEAP-INCREASE-KEY(A, A.heap-size, key)
```

O(lg*n*)



Exercise for chapter 6

- **把课本上最大堆、堆排序、最大优先队列的所有算法程序实现**

- **用最小堆重复chapter6**

7 Quicksort

- **Worst-case running time: $\Theta(n^2)$**
- **Expected running time: $\Theta(n\lg n)$**
- **Quicksort is often the best practical choice for sorting because it is remarkably efficient on the average. The constant factors hidden in the $\Theta(n\lg n)$ notation are quite small.**

Quicksort

CAR Hoare - The Computer Journal, 1962 - academic.oup.com

A description is given of a new method of sorting in the random-access store of a computer. The method compares very favourably with other known methods in speed, in economy of storage, and in ease of programming. Certain refinements of the method, which may be …

☆ 99 被引用次数：1447 相关文章 所有 7 个版本

7.1 Description of quicksort

```
QUICKSORT(A, p, r)
  if p < r
      q = PARTITION(A, p, r)
      QUICKSORT(A, p, q − 1)
      QUICKSORT(A, q + 1, r)
```

```
PARTITION(A, p, r)
  x = A[r]
  i = p − 1
  for j = p to r − 1
      if A[j] ≤ x
          i = i + 1
          exchange A[i] with A[j]
  exchange A[i + 1] with A[r]
  return i + 1
```

- **Correct?**
  Loop invariant.

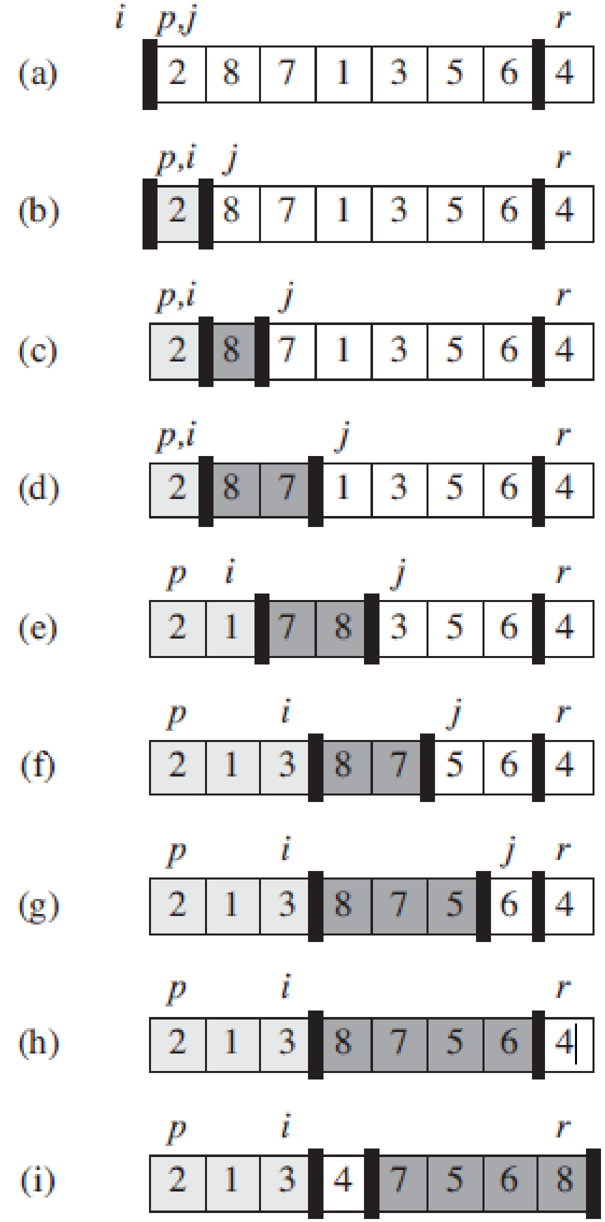
执行过程中，i 之前（含）的元素都比 $A[r]$ 小，之后的比 $A[r]$ 大。即 $p \sim i$ 的元素比 $A[r]$ 小，其后 $A[i+1] \sim A[j\text{-}1]$ 的元素比$A[r]$大。最后，交换$A[i+1]$与$A[r]$。

$A[p \sim i]$, $\leq A[r]$

$A[i+1 \sim r\text{-}1]$, $> A[r]$

swap($A[i+1]$, $A[r]$), when termination

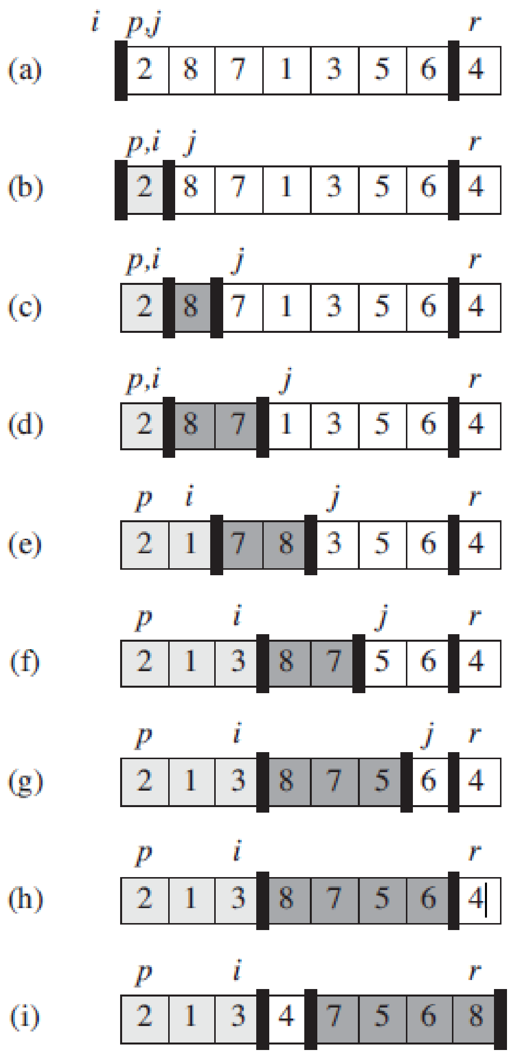
$0 \leq i \leq r\text{-}1$



7.2 Performance of quicksort

```
QUICKSORT(A, p, r)
if p < r
    q = PARTITION(A, p, r)
    QUICKSORT(A, p, q - 1)
    QUICKSORT(A, q + 1, r)
```

```
PARTITION(A, p, r)
x = A[r]
i = p - 1
for j = p to r - 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```



- **Worst-case partitioning ($\in$ Unbalanced)**

  $T(n) = T(n-1) + T(0) + \Theta(n)$?

- **Best-case partitioning ($\in$ Balanced)**

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

- **Balanced partitioning (e.g.)**

  $T(n) = T(9n/10) + T(n/10) + \Theta(n)$?

  $T(n) = T(99n/100) + T(n/100) + \Theta(n)$?

- **Running time for the average case?**

7.2 Performance of quicksort

```
QUICKSORT(A, p, r)
if p < r
    q = PARTITION(A, p, r)
    QUICKSORT(A, p, q − 1)
    QUICKSORT(A, q + 1, r)
```

```
PARTITION(A, p, r)
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

- **Worst-case partitioning**

  $$T(n) = T(n-1) + T(0) + \Theta(n)$$
  $$= n + T(n-1) = n + n-1 + T(n-2)$$
  $$= \ldots\ldots = n + n-1 + \ldots + 1 = \Theta(n^2)$$

  什么情况下出现最坏分区？

- **Best-case partitioning**

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

  Master method: $\Theta(n\lg n)$

  什么情况下出现最好分区？

7.2 Performance of quicksort

```
QUICKSORT(A, p, r)
if p < r
    q = PARTITION(A, p, r)
    QUICKSORT(A, p, q − 1)
    QUICKSORT(A, q + 1, r)
```

```
PARTITION(A, p, r)
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

- **Worst-case partitioning**

  $$T(n) = T(n\text{-}1) + T(0) + \Theta(n) = \Theta(n^2)$$

- **Unbalanced partitioning**

  $$T(n) = T(n\text{-}c\text{-}1) + T(c) + \Theta(n) \ ?$$

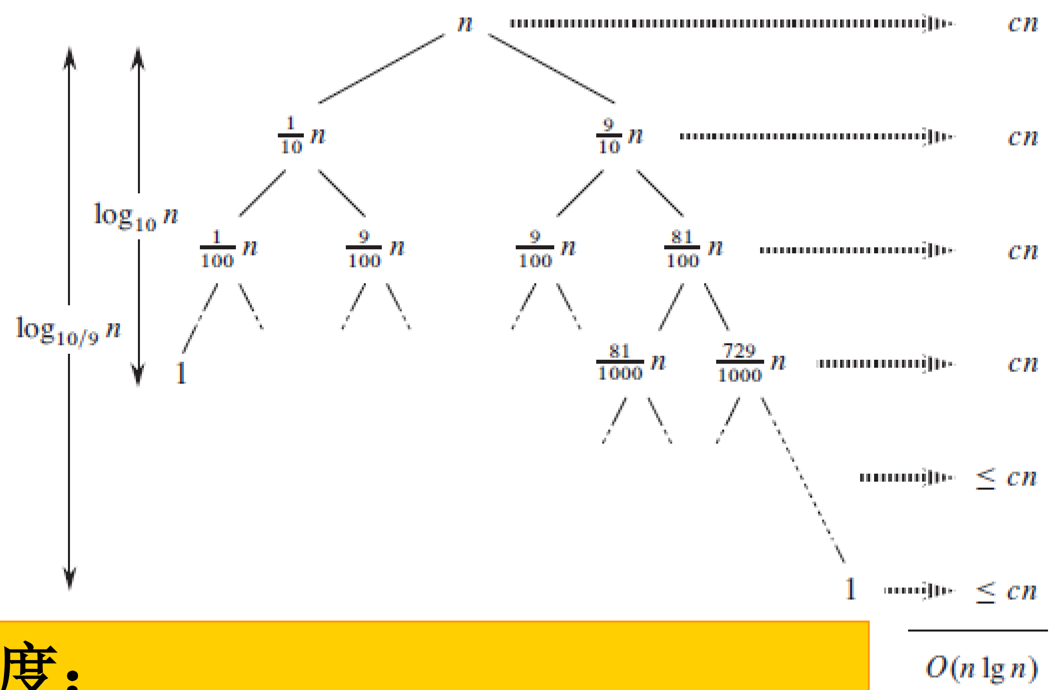
7.2 Performance of quicksort

```
QUICKSORT(A, p, r)
if p < r
    q = PARTITION(A, p, r)
    QUICKSORT(A, p, q − 1)
    QUICKSORT(A, q + 1, r)
```

```
PARTITION(A, p, r)
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

- **Balanced partitioning (e.g.)**

T(*n*) = T(9*n*/10) + T(*n*/10) + Θ(*n*) ?



树的最小高度：

n(1/10)^L = 1 时， => L = lg*n*/lg10

树的最大高度：

n(9/10)^H = 1 时， => H = lg*n*/lg(10/9)

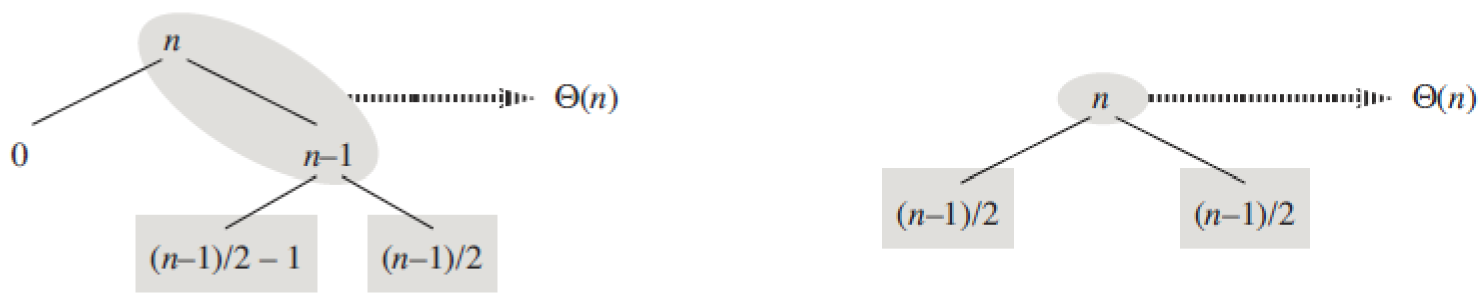
7.2 Performance of quicksort

```
QUICKSORT(A, p, r)
if p < r
    q = PARTITION(A, p, r)
    QUICKSORT(A, p, q − 1)
    QUICKSORT(A, q + 1, r)
```

```
PARTITION(A, p, r)
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

- **Average case,** **T(*n*) = ?**

Intuitively, the good and bad splits alternate levels in the tree, and that the good splits are best-case splits and the bad splits are worst-case splits.



(a) (b)

假设最好和最坏分区交叉出现，则“分区树”的高度为2*lg*n*，每一层的时间为*n*，则得O(*n*lg*n*)

7.3 A randomized version of quicksort

```
RANDOMIZED-QUICKSORT(A, p, r)
if p < r
    q = RANDOMIZED-PARTITION(A, p, r)
    RANDOMIZED-QUICKSORT(A, p, q - 1)
    RANDOMIZED-QUICKSORT(A, q + 1, r)
```

```
RANDOMIZED-PARTITION(A, p, r)
i = RANDOM(p, r)
exchange A[r] with A[i]
return PARTITION(A, p, r)
```

RANDOMIZED-PARTITION，随机分隔：从数组里随机选一个数，把它定位到它在数组里的顺序位置。

```
QUICKSORT(A, p, r)
if p < r
    q = PARTITION(A, p, r)
    QUICKSORT(A, p, q - 1)
    QUICKSORT(A, q + 1, r)
```

```
PARTITION(A, p, r)
x = A[r]
i = p - 1
for j = p to r - 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

7.4 Analysis of quicksort

```
RANDOMIZED-QUICKSORT(A, p, r)
if p < r
    q = RANDOMIZED-PARTITION(A, p, r)
    RANDOMIZED-QUICKSORT(A, p, q - 1)
    RANDOMIZED-QUICKSORT(A, q + 1, r)
```

```
RANDOMIZED-PARTITION(A, p, r)
i = RANDOM(p, r)
exchange A[r] with A[i]
return PARTITION(A, p, r)
```

```
PARTITION(A, p, r)
x = A[r]
i = p - 1
for j = p to r - 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

- **Worst-case analysis**

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

  Substitution method, $\Theta(n^2)$

- **Expected running time?**
  - **Indicator random variables**
  - **Intuitively…**

7.4 Analysis of quicksort (1)

Times-QS
for $i = 1$ to m //($m = k\cdot n$)
    RANDOMIZED-QUICKSORT(A, p, r)

RANDOMIZED-QUICKSORT(A, p, r)
if $p < r$
    q = RANDOMIZED-PARTITION(A, p, r)
    RANDOMIZED-QUICKSORT($A, p, q-1$)
    RANDOMIZED-QUICKSORT($A, q+1, r$)

- **Intuitively…**

Run RANDOMIZED-QUICKSORT m times

(Roll a dice with n points m times , each point has k times. $m = n\ k$)

$$n\left\{\begin{array}{ll}
T(n) = T(n-1) + T(0) + \Theta(n) & \text{--------}\ k \text{ times} \\
T(n) = T(n-2) + T(1) + \Theta(n) & \text{--------}\ k \text{ times} \\
T(n) = T(n-3) + T(2) + \Theta(n) & \text{--------}\ k \text{ times} \\
\ldots & \\
T(n) = T(n-n/2) + T(n/2-1) + \Theta(n) & \text{--}\ k \text{ times} \\
\ldots & \\
T(n) = T(2) + T(n-3) + \Theta(n) & \text{--------}\ k \text{ times} \\
T(n) = T(1) + T(n-2) + \Theta(n) & \text{--------}\ k \text{ times} \\
T(n) = T(0) + T(n-1) + \Theta(n) & \text{--------}\ k \text{ times}
\end{array}\right.$$

Idea of proof:
不妨令 $k = 1$，设有 x 个"非Balanced partitioning"，则有 n-x 个Balanced partitioning，则the running time of Times-QS is ($x << n$?):

$$\frac{xn^2 + (n-x)n\lg n}{n}$$
$$= \frac{xn^2 + n^2\lg n - xn\lg n}{n}$$
$$= xn + n\lg n - x\lg n$$
$$\le xn + n\lg n$$
$$\le n\lg n + n\lg n \quad \text{L} \quad (\text{if } x \le \lg n)$$
$$= 2n\lg n$$

7.4 Analysis of quicksort (2)

```
RANDOMIZED-QUICKSORT(A, p, r)
if p < r
    q = RANDOMIZED-PARTITION(A, p, r)
    RANDOMIZED-QUICKSORT(A, p, q − 1)
    RANDOMIZED-QUICKSORT(A, q + 1, r)
```

```
RANDOMIZED-PARTITION(A, p, r)
i = RANDOM(p, r)
exchange A[r] with A[i]
return PARTITION(A, p, r)
```

```
PARTITION(A, p, r)
1  x = A[r]
2  i = p − 1
3  for j = p to r − 1
4      if A[j] ≤ x
5          i = i + 1
6          exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```

- **Expected running time?**
  - **Indicator random variables**
- **running time X: the number of comparisons performed in line 4 of PARTITION.**（平均的元素比较次数）
- **For ease of analysis, we rename the elements of the array A as $z_1, z_2, \dots, z_n$**
- **$Z_{ij} = \{ z_i, z_{i+1}, \dots, z_j \}$: set of elements between z_i and z_j, inclusive. z_i is the ith smallest element.** (分区标志随机选择，该假设因此是合理的)
- **Indicator random variables:**

  $$X_{ij} = \mathrm{I}\{ z_i \text{ is compared to } z_j \}$$

  X_{ij} ：任意两个元素z_i和 z_j的比较次数
- **The total number of comparisons (running of quicksort)**

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

7.4 Analysis of quicksort - **Indicator random variables**

```
RANDOMIZED-QUICKSORT(A, p, r)
if p < r
    q = RANDOMIZED-PARTITION(A, p, r)
    RANDOMIZED-QUICKSORT(A, p, q − 1)
    RANDOMIZED-QUICKSORT(A, q + 1, r)
```

```
RANDOMIZED-PARTITION(A, p, r)
i = RANDOM(p, r)
exchange A[r] with A[i]
return PARTITION(A, p, r)
```

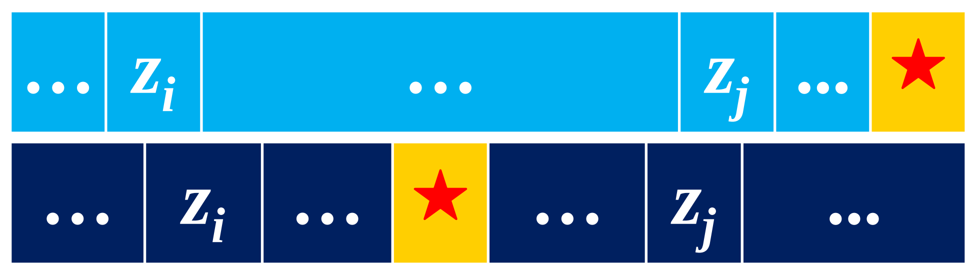
```
PARTITION(A, p, r)
1  x = A[r]
2  i = p − 1
3  for j = p to r − 1
4      if A[j] ≤ x
5          i = i + 1
6          exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```

- **X: the number of comparisons performed in line 4 of PARTITION.**
- **$Z_{ij} = \{ z_i , z_{i+1} , \ldots , z_j \}$**
- **Indicator random variables:**

  $X_{ij} = \mathrm{I}\{ z_i$ is compared to $z_j \}$
- **The total number of comparisons**

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

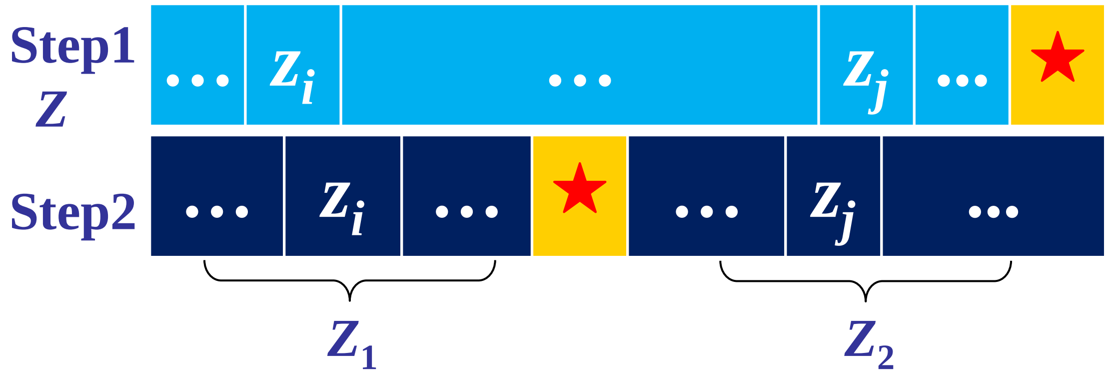
- **整个快排过程中，两个数 z_i and z_j 最多比较一次，when ?**



7.4 Analysis of quicksort - **Indicator random variables**

```
PARTITION(A, p, r)
   x = A[r]
   i = p − 1
   for j = p to r − 1
       if A[j] ≤ x
           i = i + 1
           exchange A[i] with A[j]
   exchange A[i + 1] with A[r]
   return i + 1
```

整个快排过程中，两个数 z_i and z_j 最多比较一次，出现在分区时产生了子序列 Z_{ij}



Step1: 快排过程中，出现序列 Z，标记元素为★

Step2: 对序列 Z分区后，产生两个子序列 Z_1 和 Z_2

(1) Z 中的其他元素仅与★比较一次

(2) 若 z_i 或 z_j 为★，则两者比较一次，此后**不再相遇**（不会比较）

(3) 若 z_i 与 z_j 分别被分区到 Z_1 与 Z_2，则两者无比较，此后也不会相遇（**过去没有，此时没有，将来也没有**）

(4) 若 z_i 与 z_j 被分区到同一 Z_1 或 Z_2，重复Step1和Step2的逻辑

7.4 Analysis of quicksort

- **Indicator random variables:**

  X_{ij} = I{ z_i is compared to z_j }
- **The total number of comparisons**

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

```
PARTITION(A, p, r)
x = A[r]
i = p - 1
for j = p to r - 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

$$\begin{aligned} \mathrm{E}[X] &= \mathrm{E}\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}\right] \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathrm{E}[X_{ij}] \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \text{ is compared to } z_j\} \end{aligned}$$

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

$$\mathrm{E}[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} = \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} < \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k} = \sum_{i=1}^{n-1} O(\lg n) = O(n \lg n)$$

7.4 Analysis of quicksort (3) - why is quicksork quick?

```
RANDOMIZED-QUICKSORT(A, p, r)
if p < r
    q = RANDOMIZED-PARTITION(A, p, r)
    RANDOMIZED-QUICKSORT(A, p, q − 1)
    RANDOMIZED-QUICKSORT(A, q + 1, r)
```

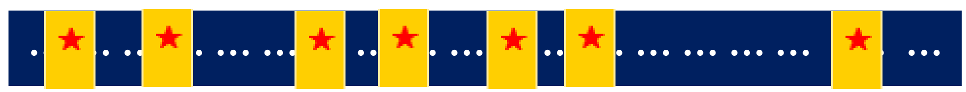
```
RANDOMIZED-PARTITION(A, p, r)
i = RANDOM(p, r)
exchange A[r] with A[i]
return PARTITION(A, p, r)
```

```
PARTITION(A, p, r)
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

Intuitively…

第 1 次分区，定位好 1 个元素

第 2 次分区，又定位好 2 个元素（已定位 2^2-1个）



第 3 次分区，又定位好 4 个元素（已定位 2^3-1 个）

……

第 k 次分区，又定位好 2^{k-1} 个元素（共定位 2^k-1 个）

$2^k - 1 = n \Rightarrow k = \lg(n+1)$

每次分区有最多n-1次比较 ⇨ $O(n\lg n)$

Lower bounds for sorting

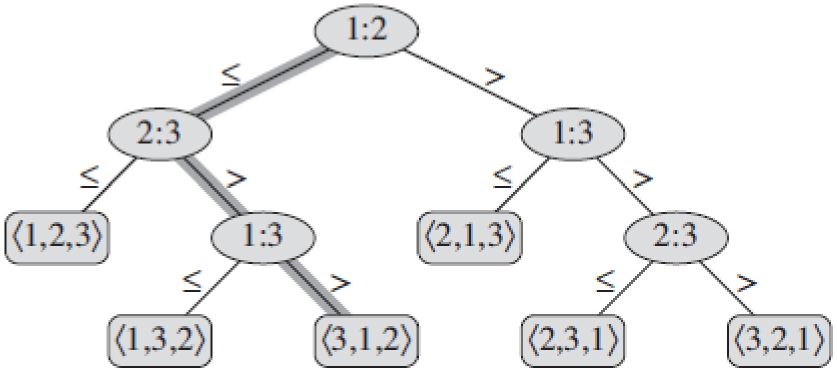
- **Comparison sort**
  - ✓ **The sorted order they determine is based only on comparisons between the input elements.**
  - ✓ **We use only comparisons between elements to gain order information about an input sequence <a_1, a_2, … , a_n>. That is, given two elements a_i and a_j , without loss of generality, we perform only comparison $a_i \le a_j$.**
- **Algorithms**
  - ✓ **bubble, select, insert, merge, heap, quick, …**
- **Running time**

  $\Omega(n\lg n)$?

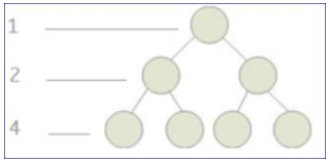
The decision-tree model

- **We can view comparison sorts abstractly in terms of decision trees.**
- **Decision tree: is a full binary tree that represents the comparisons between elements that are performed by a particular sorting algorithm operating on an input of a given size.**（给定某个输入，某种算法执行时，由元素之间的比较而产生的一个满二叉树）

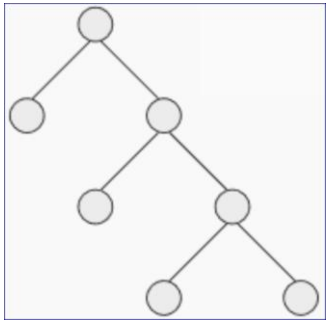
Example: The decision tree for insertion sort operating on three elements



“中国”的满二叉树

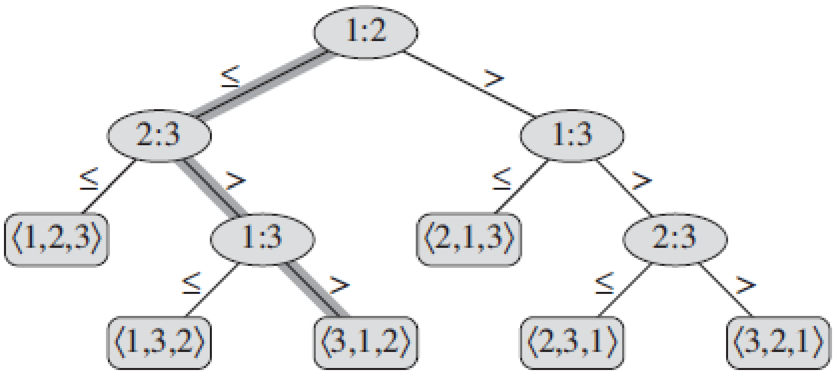


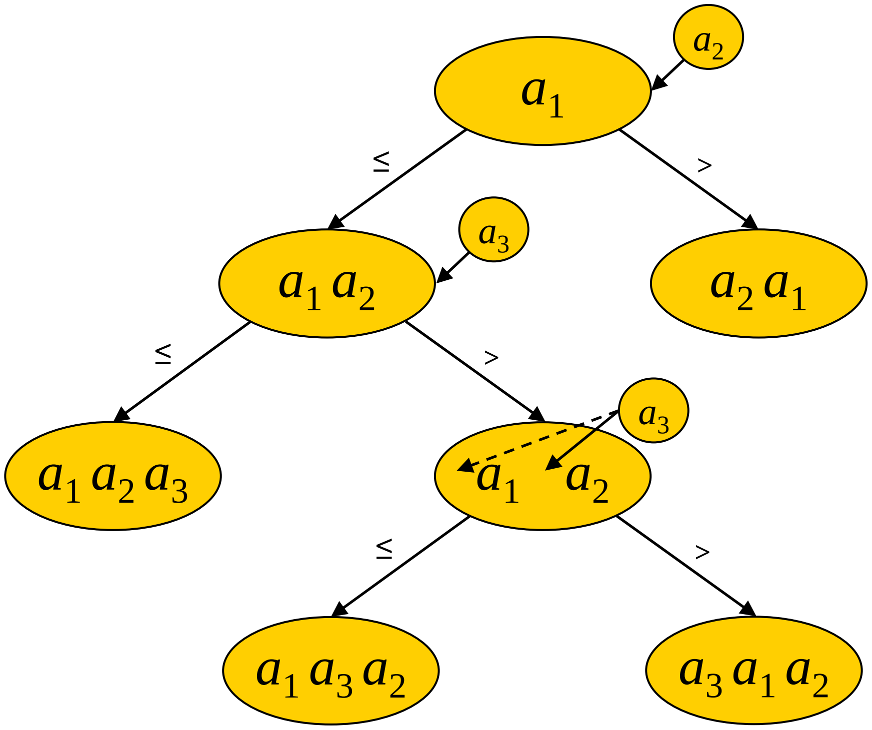
“外国”的满二叉树



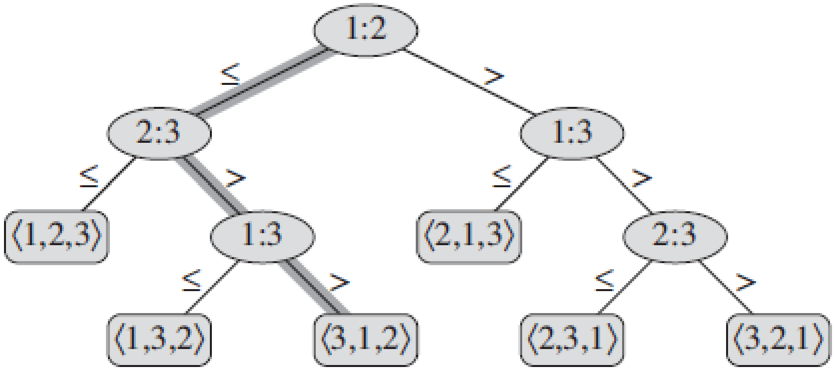
The decision-tree model

Example: The decision tree for insertion sort operating on three elements





The decision-tree model



- **In a decision tree, each leaf is a permutation (a solution of sort)**

  <π(1), π(2), … , π(*n*)> of <1, 2, … , *n*>.

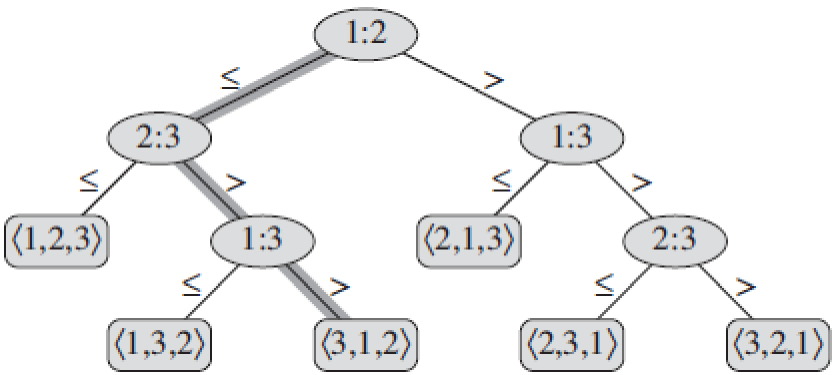
- **There have *n*! permutations of <1, 2, … , *n*>..**
- **A correct sorting algorithm must be able to produce a permutation(leaf) that establish the ordering**

  $$a_{\pi(1)} \le a_{\pi(2)} \le \dots \le a_{\pi(n)}$$

- **An actual execution of the comparison sort: A path from the root by a downward to the leaf. What's the height *h*?**

The decision-tree model

- **What's the height h for a decision tree corresponding to a comparison sort?**



- **A comparison sort on *n* elements: *n*! permutations.**
- **For a decision tree**
  - **leaves: *l***
  - **height: *h***

$$n! \le l \le 2^h$$

$n! \le l$：叶子数 ≥ n排列数，能保证所有可能的解都被包括

$l \le 2^h$：对高度为 h 的二叉树，叶子数最多为 2^h（就是满树时）

$$h \ge \lg(n!) = \Omega(n\lg n)\ \ ?$$

The decision-tree model

$$h \geq \lg(n!) = \Theta(n\lg n)\ ? \qquad \therefore\ h = \Omega(n\lg n)$$

$$\lg(n!) = \Theta(n \lg n)\ , \tag{3.19}$$

where Stirling's approximation is helpful in proving equation (3.19). The following equation also holds for all $n \geq 1$:

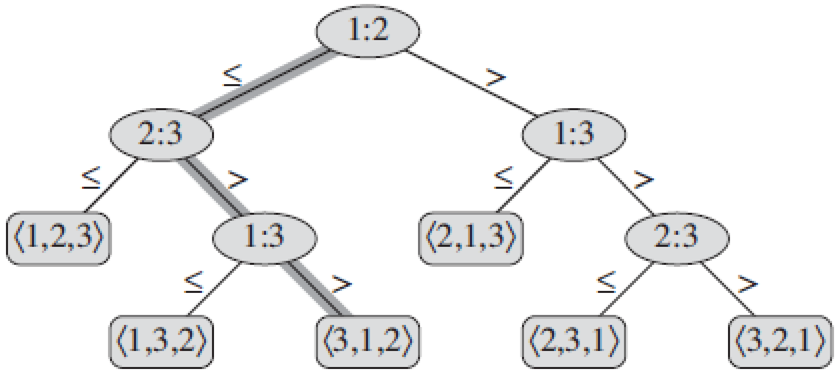
$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{\alpha_n} \tag{3.20}$$

where

$$\frac{1}{12n+1} < \alpha_n < \frac{1}{12n}\ . \tag{3.21}$$

$$(n/2)*(n/2)* \ldots *(n/2) = (n/2)^{(n/2)} < n! < n^n$$

The decision-tree model



$$n! \le l \le 2^h$$

$$h \ge \lg(n!) = \Omega(n\lg n)$$

Lestor R. Ford, Jr. and Selmer M. Johnson. A tournament problem. The American Mathematical Monthly, 66(5):387–389, 1959.

Sorting in Linear Time **

- **Sorting in Linear Time**
  - ✓ **counting sort**
  - ✓ **radix sort**
  - ✓ **bucket sort**

These algorithms use operations other than comparisons to determine the sorted order. Consequently, the $\Omega(n\lg n)$ lower bound does not apply to them.

9 Medians and Order Statistics

- The ith **order statistic** of a set of n elements is the ith smallest element.
  - ✓ the **minimum** of a set of elements is the first order statistic (i = 1).
  - ✓ the **maximum** is the nth order statistic ($i = n$).
  - ✓ A **median**, informally, is the “halfway point” of the set.

6	12	5	9	2	10	8	7

Minimum: 2

Maximum: 12

9 Medians and Order Statistics

- The *i*th **order statistic** of a set of *n* elements is the *i*th smallest element.
- For convenience, consider the problem of selecting the *i*th order statistic from a set of *n* distinct numbers.
- We can solve the selection problem in O$(n\lg n)$ time, since we can sort the numbers and then simply index the *i*th element in the output array. Can we do it better?

6	12	5	9	2	10	8	7

2	5	6	7	8	9	10	12

The 5th **order statistic** is 8

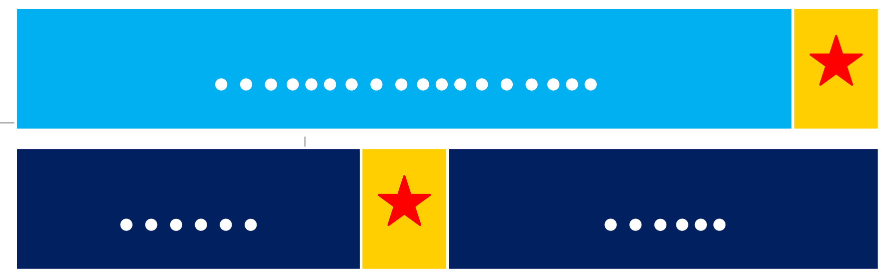
9.1 Minimum and maximum

```
MINIMUM(A)
    min = A[1]
    for i = 2 to A.length
        if min > A[i]
            min = A[i]
    return min
```

***n*-1 comparisons**

9.2 Selection in expected linear time

- The general selection problem appears more difficult than the simple problem of finding a minimum.
- Yet, surprisingly, the asymptotic running time for both problems is the same: $\Theta(n)$.



```
RANDOMIZED-SELECT(A, p, r, i)
    if p == r
        return A[p]
    q = RANDOMIZED-PARTITION(A, p, r)
    k = q - p + 1
    if i == k                    // the pivot value is the answer
        return A[q]
    elseif i < k
        return RANDOMIZED-SELECT(A, p, q - 1, i)
    else return RANDOMIZED-SELECT(A, q + 1, r, i - k)
```

// 随机分区，找到*A*中的第*k*小的元素*A*[*q*]

Firstly,

p* is 1, *r* is *n

9.2 Selection in expected linear time

$T(n) = T(\max(k-1, n-k)) + O(n)$

```
RANDOMIZED-SELECT(A, p, r, i)
if p == r
    return A[p]
q = RANDOMIZED-PARTITION(A, p, r)
k = q − p + 1
if i == k          // the pivot value is the answer
    return A[q]
elseif i < k
    return RANDOMIZED-SELECT(A, p, q − 1, i)
else return RANDOMIZED-SELECT(A, q + 1, r, i − k)
```

- **Worst-case running time**

  $T(n) = T(n-1) + O(n)$,

  $\Theta(n^2)$

- **A special case**

  $q = (r-p)/2$, then

  $T(n) = T(n/2) + O(n)$,

  $\Theta(n)$

- **Expected running time ?**

  Indicator random variables, $\Theta(n)$? *

9.2 Selection in expected linear time

$T(n) = T(\max(k-1, n-k)) + O(n)$

```
RANDOMIZED-SELECT(A, p, r, i)
if p == r
    return A[p]
q = RANDOMIZED-PARTITION(A, p, r)
k = q - p + 1
if i == k            // the pivot value is the answer
    return A[q]
elseif i < k
    return RANDOMIZED-SELECT(A, p, q - 1, i)
else return RANDOMIZED-SELECT(A, q + 1, r, i - k)
```

- **Expected running time ?**

Indicator random variables, $\Theta(n)$?

- **Intuitively,**

Run RANDOMIZED-SELECT *m* times

($m = x \cdot n$: Roll a dice with *n* points *m* times , each point has *x* times.)

$$m \cdot T(n) = x \sum_{k=1}^{n} (T(\max(k-1,\ n-k)) + O(n))$$

$$T(n) = \frac{1}{n} \sum_{k=1}^{n} (T(\max(k-1,\ n-k)) + O(n))$$

$$\le \frac{2}{n} \sum_{k=n/2}^{n-1} T(k) \ + \ O(n)$$

Using substitution,

$T(k) \le ck \Rightarrow T(n) \le cn$?

9.2 Selection in expected linear time

```
RANDOMIZED-SELECT(A, p, r, i)
    if p == r
        return A[p]
    q = RANDOMIZED-PARTITION(A, p, r)
    k = q - p + 1
    if i == k            // the pivot value is the answer
        return A[q]
    elseif i < k
        return RANDOMIZED-SELECT(A, p, q - 1, i)
    else return RANDOMIZED-SELECT(A, q + 1, r, i - k)
```

$T(n) = T(\max(k-1, n-k)) + O(n)$

$$T(n) = \frac{1}{n}\sum_{k=1}^{n}(T(\max(k-1,\ n-k)) + O(n))$$

$$\leq \frac{2}{n}\sum_{k=n/2}^{n-1} T(k) \ + \ O(n)$$

Using substitution,

$T(k) \leq ck$ => $T(n) \leq cn$?

$$T(n) \leq \frac{2}{n}\sum_{k\geq\frac{n}{2}}^{n-1} T(k) + O(n)$$

$$\leq \frac{2}{n}\sum_{k=\frac{n}{2}}^{n-1} c\cdot k \ + \ n$$

$$= \frac{2c}{n}\left(\frac{n}{2} + \frac{n}{2}+1 + \cdots + \frac{n}{2}+\frac{n}{2}-1\right) + n$$

$$= \frac{2c}{n}\cdot\left[\frac{n}{2}\cdot\frac{n}{2} + \left(\frac{n}{2}-1\right)\cdot\frac{n}{4}\right] + n$$

$$= \frac{2c}{n}\cdot\frac{n}{4}\left(\frac{3n}{2}-1\right) + n$$

$$= \frac{c}{2}\cdot\left(\frac{3n}{2}-1\right) + n$$

$$= \left(\frac{3c}{4}+1\right)n - \frac{c}{2} \quad < \ c\cdot n.$$

we need

Only if $\frac{3c}{4} + 1 < c \Rightarrow 4 < c$

作业

- 6~9章所有的课后习题

- **Running time?**

$$T(n) = T(n-3) + T(2) + \Theta(n)$$

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

$$T(n) = T(n/a) + O(n) \quad \ldots (a > 1)$$

练习

```
RANDOMIZE-IN-PLACE(A, n)
for(i=1; i<=n; i++)
      swap(A[i], A[RANDOM(i, n)])
```

产生 1, 2, 3, ... , n (如n=10000) 的随机置换 A （如上算法），

从A中取部分数据B，如前80/100，

在B中找第k小的数，

分别用：

排序法，$O(n\lg n)$；

chapter9.2的随机分区法，$O(n)$。

比较两种方法分别多少次能找到（设置一个计数器）。

思考

E1-算法第1次练习赛

- 简介
- 题目
- 排名
- 提交
- 回复&&公告
- 导出成绩

服务器当前时间
2019-10-08 17:02:50

比赛结束时间
2019-10-07 12:00:00

比赛已结束

比赛排名 更新于 2019-10-08 17:02:49

« ‹ 1 2 3 › »

排名	用户	账号	学号	得分	罚时	A 197/205	B 191/201	C 193/195	D 180/194	E 167/194	F 200/203	G 159/188
1			38	800	12:30:20	1:05:25	0:57:29	0:51:13	3:15:20(+1)	0:40:09(+1)	0:15:59	1:37:44
2				800	13:56:31	0:01:52	0:04:56	0:10:48(+1)	0:12:23	0:07:50	0:02:20	10:52:35(+4)
3				800	15:04:38	0:15:37	0:22:07	0:40:12	6:33:30	0:30:55(+1)	0:23:52	5:05:03(+2)
4				800	16:41:19	1:12:09	1:15:38(+1)	1:42:52(+1)	1:31:40	1:23:33(+1)	1:05:44	2:01:47
5				800	23:39:29	1:32:02	1:06:10	1:16:31(+1)	2:09:06	0:56:31	1:20:54	4:57:32(+11)
6				800	27:50:09	0:02:30	0:05:13	0:06:00	0:10:04	0:00:18	0:00:40	24:53:39(+4)
7				800	34:05:25	0:56:02	1:04:26(+1)	1:14:38(+1)	1:20:56	1:25:19	1:28:19	2:20:59(+1)
8				800	34:19:09	0:02:38	0:04:54	0:07:34	0:10:41	0:12:27	0:13:32	24:50:41(+3)
9				800	39:31:30	0:24:26	0:30:19(+2)	1:11:43(+2)	1:19:21	1:35:00	0:12:42	25:26:19(+5)
10				800	41:49:13	0:37:41	0:51:01(+3)	0:57:24(+1)	1:12:14	0:33:40	0:34:54	24:09:54(+1)
11				800	49:10:54	1:38:36(+3)	2:00:39(+2)	2:10:06	3:00:51	2:41:20	1:44:35	4:12:26(+3)
12				800	52:41:02	0:14:19(+1)	4:04:53(+9)	2:41:13(+1)	4:31:44	1:03:03(+2)	0:44:42	5:23:14(+7)
13				800	55:16:39	0:55:09	1:35:01(+9)	1:55:40(+2)	2:46:37(+1)	0:47:49	0:44:56	29:20:44(+5)
14				800	55:21:20	1:24:55	1:35:05	1:45:45(+1)	4:12:18(+3)	4:22:54(+1)	4:26:51	5:09:12(+2)
15				800	63:55:31	0:58:42	1:11:28(+3)	1:21:05	7:51:07(+2)	1:45:22	1:56:53	9:11:05(+5)
16			22	800	64:49:23	1:15:19	2:03:50(+5)	2:10:53	4:26:02	1:49:49	1:19:47(+1)	3:30:38(+1)

有的课，1600+ 人的榜单实时刷新

思考

所有评测记录

序号	用户	题目ID	结果	得分	语言	代码长度 (Bytes)	运行时间 (ms)	运行内存 (KB)	提交时间
			all		all				
1856356		2470	**Accepted**	1	c	244	1	1480	55 分钟 13 秒前
1856355		2470	**Compile Error**	0	c	246	0	0	56 分钟 16 秒前
1856354		2470	**Wrong Answer**	0	c	234	5	1508	58 分钟 11 秒前
1856353		2462	**Wrong Answer**	0.3	c	174	6	1488	1 小时 0 分钟前
1856352		2461	**Accepted**	1	c	338	7	2140	1 小时 3 分钟前
1856351		2456	**Accepted**	1	c	372	0	1284	1 小时 3 分钟前
1856350		2462	**Accepted**	1	c	202	0	1488	1 小时 8 分钟前
1856349		2459	**Accepted**	1	c	34206	0	1368	1 小时 10 分钟前
1856348		37	**Accepted**	1	c++	500	1	3428	1 小时 10 分钟前
1856347		2456	**Accepted**	1	c	241	1	1336	1 小时 13 分钟前
1856346		36	**Accepted**	1	c++	124	4865	3132	1 小时 20 分钟前
1856345		2461	**Accepted**	1	c	416	29	1908	1 小时 23 分钟前
1856344		35	**Accepted**	1	c++	293	25	3276	1 小时 25 分钟前
1856343		2462	**Accepted**	1	c	268	8	1508	1 小时 34 分钟前
1856342		2	**Accepted**	1	c++	127	2	3164	1 小时 39 分钟前

185+ 万的实时评测记录