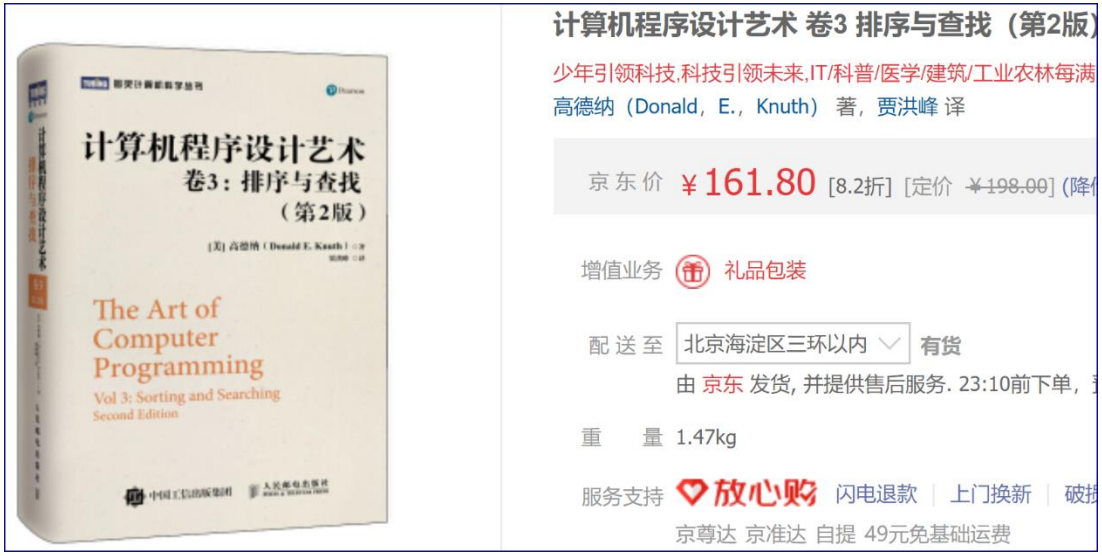
Part II

Sorting and Order Statistics

千万不要觉得排序简单!

Sorting and Order Statistics

计算机程序设计艺术中文版：
- 卷1基本算法第3版
- 卷2半数值算法第3版
- 卷3排序与查找第2版
- 卷4A组合算法一
- MMIX增补（套装共5册）



Sorting and Order Statistics

Chapter 6 Heapsort

- Maintaining the heap property
- Building a heap
- The heapsort algorithm
- Priority queues

Chapter 7 Quicksort

- Description and Performance
- A randomized version of quicksort
- Analysis of quicksort

Chapter 8 Sorting in Linear Time*

Chapter 9 Medians and Order Statistics

- Minimum and maximum
- Selection in expected linear time

6 Heapsort

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

调整二叉树的元素位置，使其为最大堆

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

调整二叉树的元素位置，使其为最大堆

```
MAX-HEAPIFY(A, i)
 1  l = LEFT(i)
 2  r = RIGHT(i)
 3  if l ≤ A.heap-size and A[l] > A[i]
 4      largest = l
 5  else largest = i
 6  if r ≤ A.heap-size and A[r] > A[largest]
 7      largest = r
 8  if largest ≠ i
 9      exchange A[i] with A[largest]
10      MAX-HEAPIFY(A, largest)
```

调整二叉树的元素位置，使其为最大堆

堆排序算法：

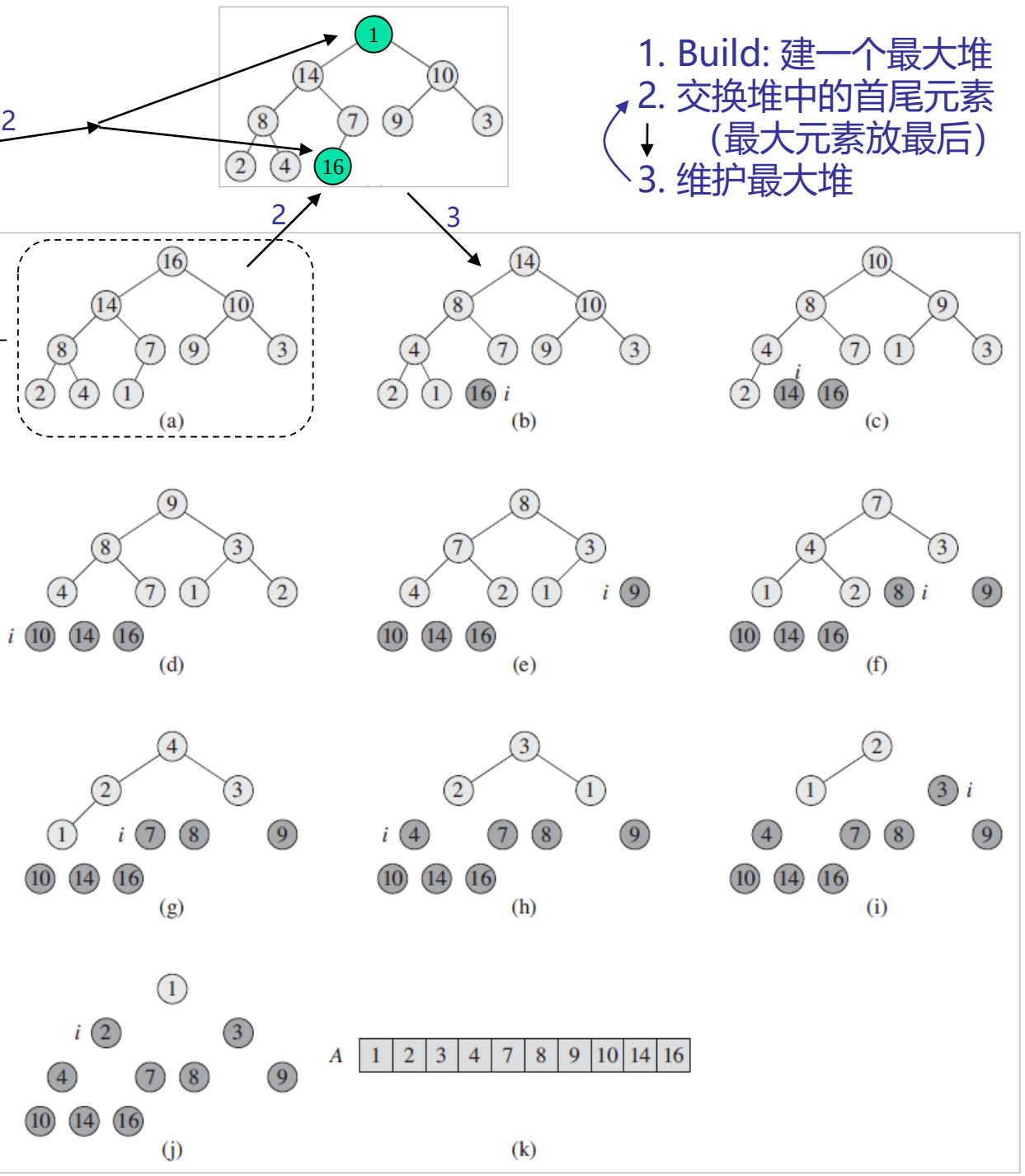
1. 建堆（得到最大堆）
2. 交换元素（最大元素定位到最后一个位置）
3. 最大堆维护（子序列中，最大堆性质被破坏了【上一步中最大元素被改变】），回到第2步

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

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

```
MAX-HEAPIFY(A, i)
 1  l = LEFT(i)
 2  r = RIGHT(i)
 3  if l ≤ A.heap-size and A[l] > A[i]
 4      largest = l
 5  else largest = i
 6  if r ≤ A.heap-size and A[r] > A[largest]
 7      largest = r
 8  if largest ≠ i
 9      exchange A[i] with A[largest]
10      MAX-HEAPIFY(A, largest)
```

1. Build: 建一个最大堆
2. 交换堆中的首尾元素
   （最大元素放最后）
3. 维护最大堆

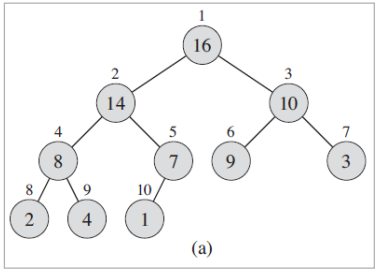


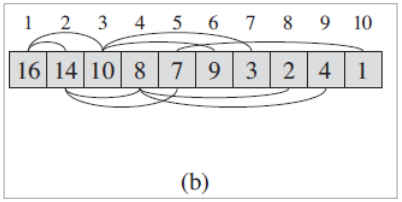
6 Heapsort

- **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





工程模型 → 物理模型 → 数学模型

or 计算模型

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

(二叉树的深度为h，除第 h 层外，其它各层 (1～h-1) 的结点数都达到最大个数，第h层所有的结点都连续集中在最左边)

- ***A*: An array can represent a heap**

- ***A.length* : the number of elements in the array;**
  ***A.heap-size* : the number of elements in the heap**
  ($0 \le A.heap\text{-}size \le A.length$)

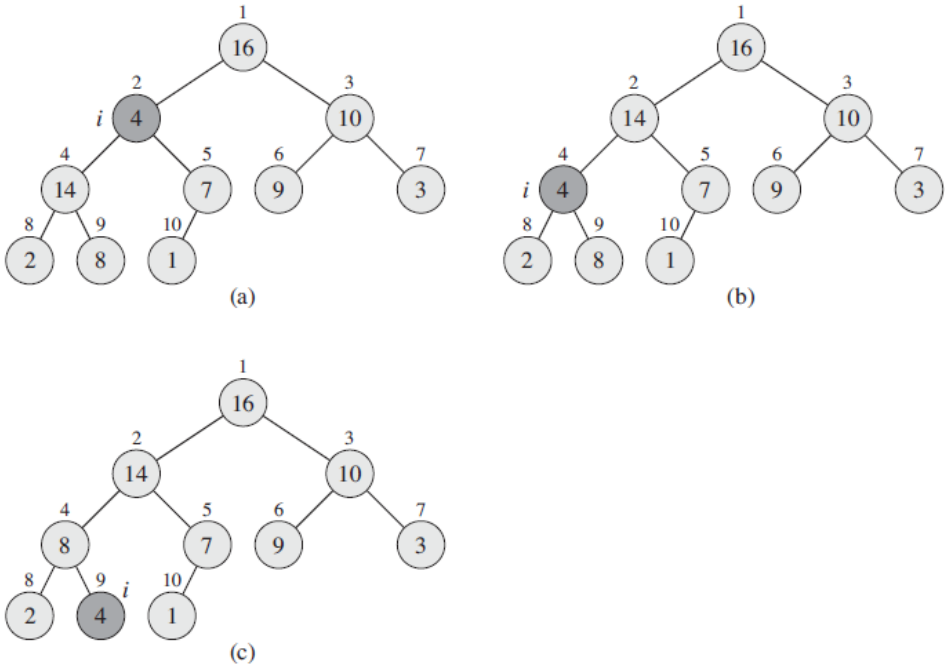
```
PARENT(i)
1  return ⌊i/2⌋

LEFT(i)
1  return 2i

RIGHT(i)
1  return 2i + 1
```

- **max-heap : $A[PARENT(i)] \ge A[i]$, 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.**

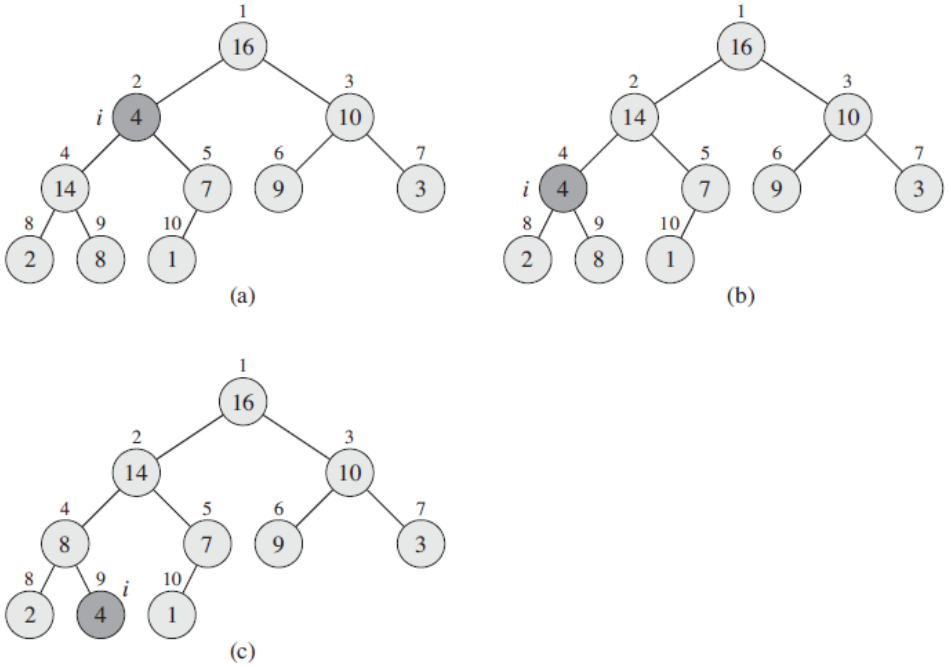
```
MAX-HEAPIFY(A, i)
 1  l = LEFT(i)
 2  r = RIGHT(i)
 3  if l ≤ A.heap-size and A[l] > A[i]
 4       largest = l
 5  else largest = i
 6  if r ≤ A.heap-size and A[r] > A[largest]
 7       largest = r
 8  if largest ≠ i
 9       exchange A[i] with A[largest]
10       MAX-HEAPIFY(A, largest)
```



- **The running time?**

6.2 Maintaining the heap property

```
MAX-HEAPIFY(A, i)
 1  l = LEFT(i)
 2  r = RIGHT(i)
 3  if l ≤ A.heap-size and A[l] > A[i]
 4        largest = l
 5  else largest = i
 6  if r ≤ A.heap-size and A[r] > A[largest]
 7        largest = r
 8  if largest ≠ i
 9        exchange A[i] with A[largest]
10        MAX-HEAPIFY(A, largest)
```



- **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*).**

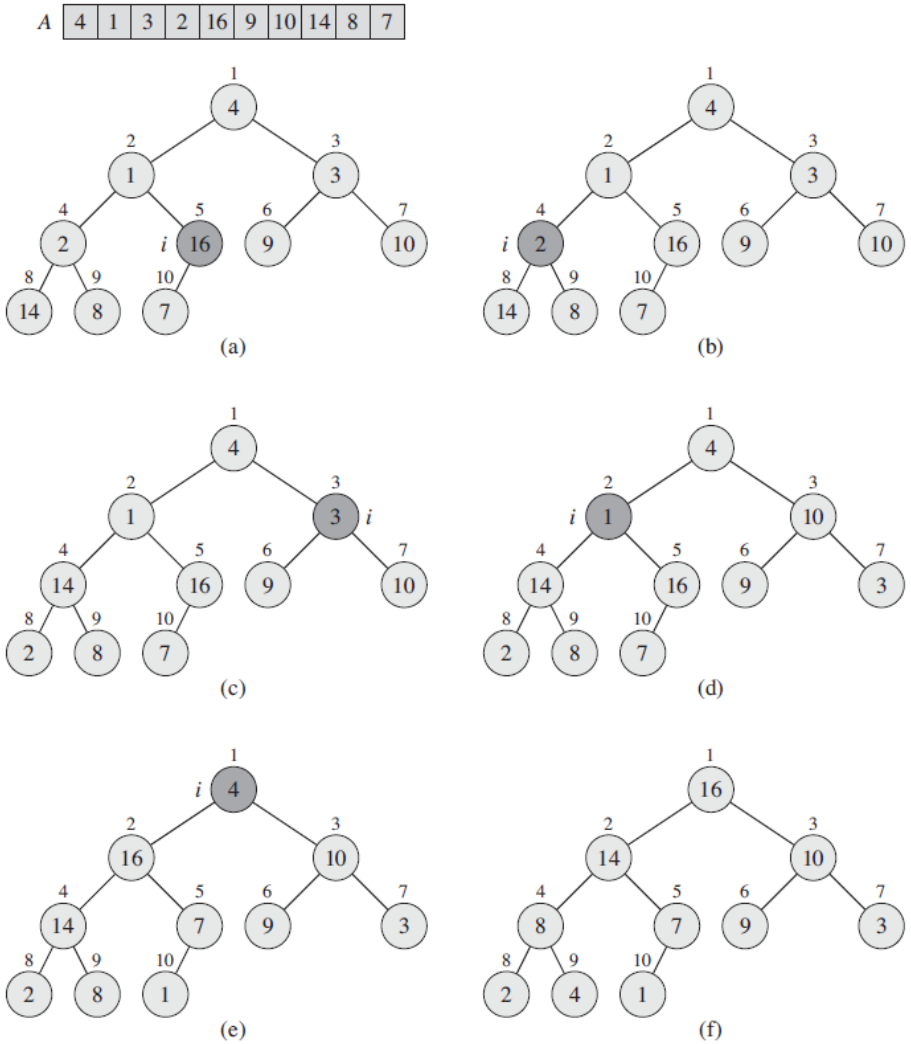
6.3 Building a Heaps

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

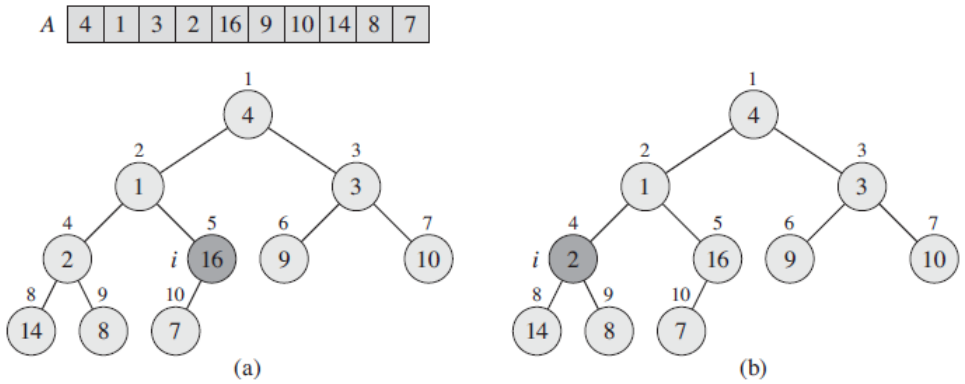
- **Correct ?**

  Loop invariant.
- **The running time?**



6.3 Building a Heaps

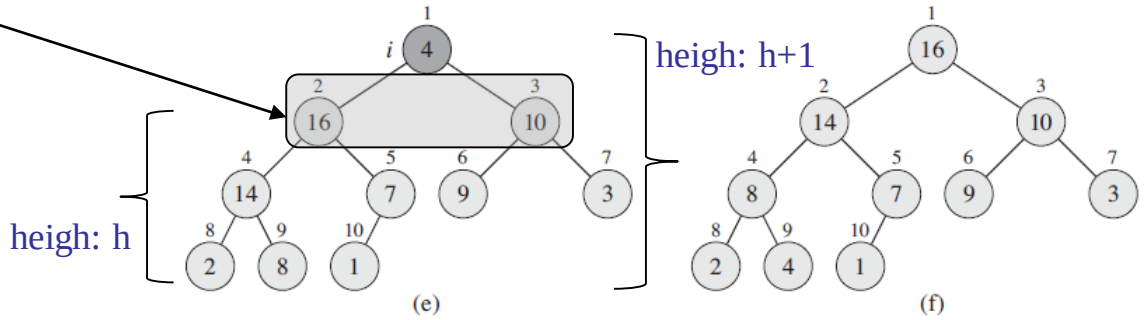
```
BUILD-MAX-HEAP(A)
1  A.heap-size = A.length
2  for i = ⌊A.length/2⌋ downto 1
3      MAX-HEAPIFY(A, i)
```



- **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*.

或者直观地说，*n*/2个节点，高度为lg*n*的节点1个（根节点），高度为(lg*n*)-1的节点2个，高度为(lg*n*)-2的节点4个，……，也就是说，大部分节点的高度很小，并不是所有的*n*/2个节点的高度都是lg*n*

6.3 Building a Heaps

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

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

?

6.4 The heapsort algorithm

The running time of Heapsort ?

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

n

O(*n*)

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

O(*n*lg*n*)

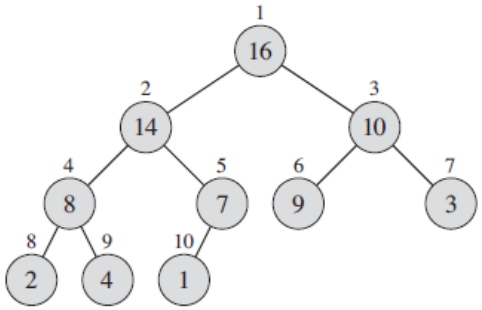
O(*h*)

```
MAX-HEAPIFY(A, i)
 1  l = LEFT(i)
 2  r = RIGHT(i)
 3  if l ≤ A.heap-size and A[l] > A[i]
 4      largest = l
 5  else largest = i
 6  if r ≤ A.heap-size and A[r] > A[largest]
 7      largest = r
 8  if largest ≠ i
 9      exchange A[i] with A[largest]
10      MAX-HEAPIFY(A, largest)
```

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

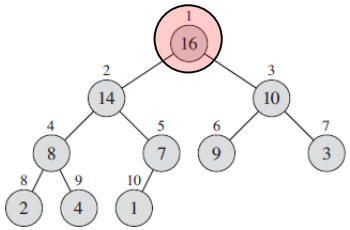


- **A max-priority queue supports the following operations:**
  - **MAXIMUM(*S*)**: returns the element of *S* with the largest key.
  - **EXTRACT-MAX**(*S*): removes and returns the element of *S* with the largest key.
  - **INCREASE-KEY**(*S*, *i*, *k*): increases the value of element *i*'s key to the new value *k*, which is assumed to be at least as large as *i*'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: schedule jobs on a shared computer.**
  - **A min-priority queue: an event-driven simulator.**

Operations of a max-priority queue

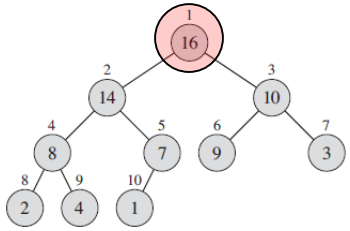
Θ(1)

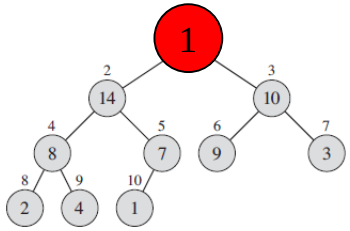
```
HEAP-MAXIMUM(A)
1   return A[1]
```

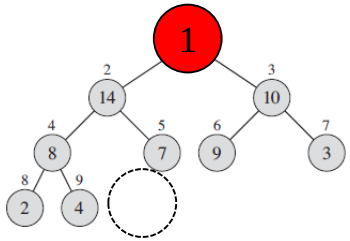


O(lg*n*)

```
HEAP-EXTRACT-MAX(A)
1   if A.heapsize < 1
2       error “heap underflow”
3   max = A[1]
4   A[1] = A[A.heapsize]
5   A.heapsize--
6   MAX-HEAPIFY(A, 1)
7   return max
```

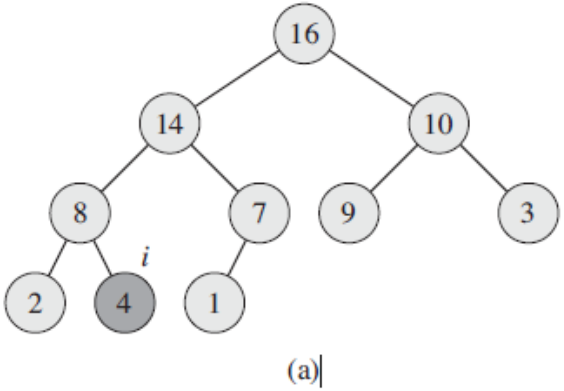




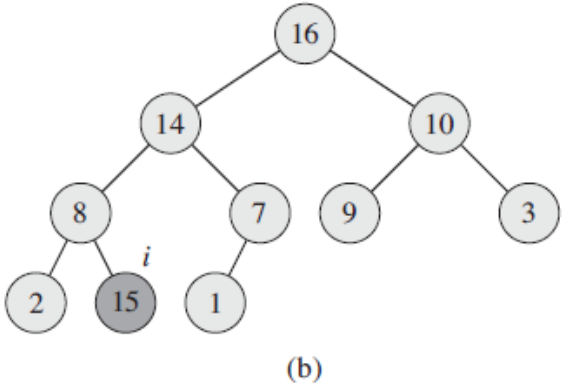


```
HEAP-INCREASE-KEY(A, i, key)
1   if key < A[i]
2       error "new key is smaller than current key"
3   A[i] = key
4   while i > 1 and A[PARENT(i)] < A[i]
5       exchange A[i] with A[PARENT(i)]
6       i = PARENT(i)
```

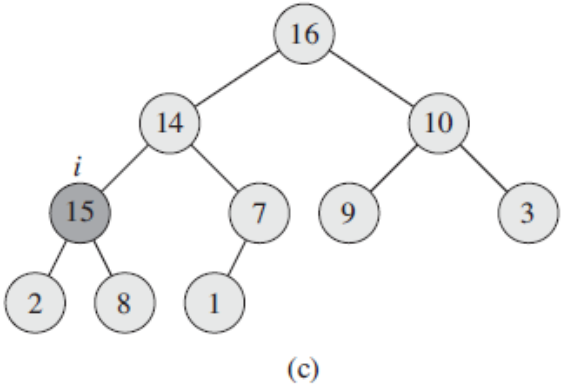
O(lg*n*)



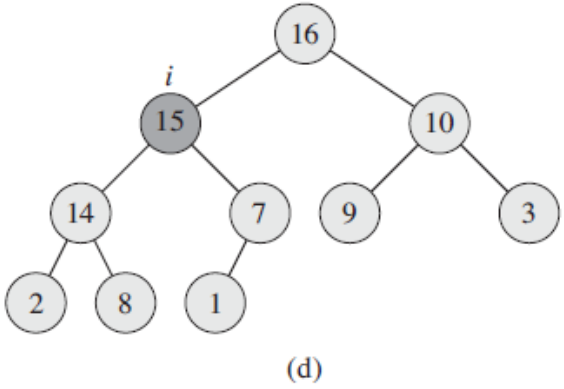
(a)



(b)



(c)



(d)

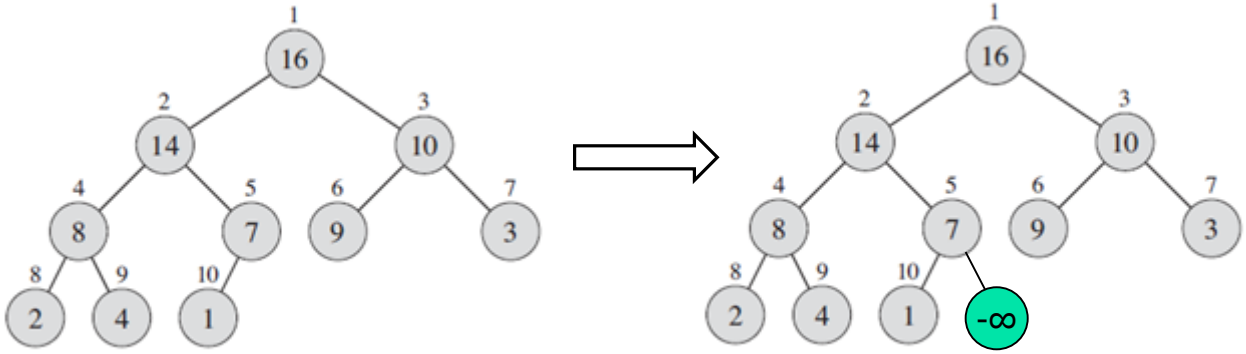
Operations of a max-priority queue

O(lg*n*)

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

O(lg*n*)

```
MAX-HEAP-INSERT(A, key)
1   A.heapsize++
2   A[A.heapsize] = -∞
3   HEAP-INCREASE-KEY(A, A.heapsize, key)
```



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
- 堆排序是否是稳定的?

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 $\Theta(n\lg n)$ are quite small.**

7.1 Description of quicksort

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

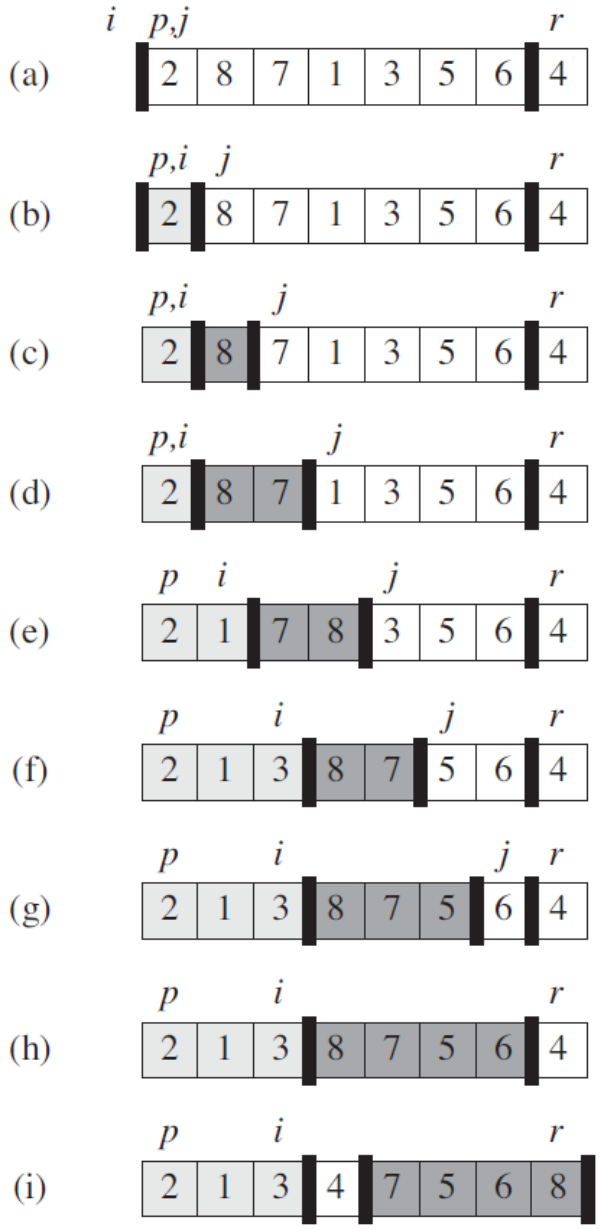
执行过程中，i 之前（含）的元素小于等于哨兵（标记） $A[r]$， i 之后的大于 $A[r]$。
即 $p \sim i$ 的元素小于等于 $A[r]$，其后 $A[i+1] \sim A[j-1]$ 的元素比$A[r]$大。
最后，交换$A[i+1]$与$A[r]$。

A[*p* ~ *i*] ，≤A[*r*]

A[*i*+1 ~ *r*-1] , > A[*r*]

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

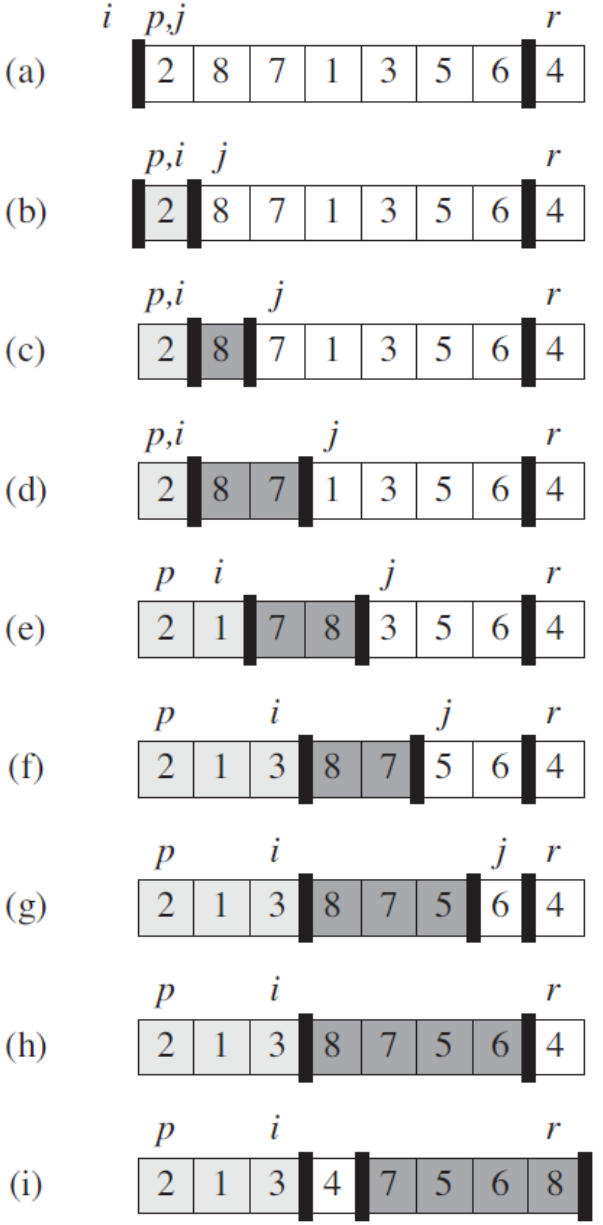
0 ≤ *i* ≤ *r*-1



7.2 Performance of quicksort

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



- **Worst-case partitioning (∈ Unbalanced)**

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

- **Best-case partitioning (∈ Balanced)**

  $T(n) = 2T(n/2) + \Theta(n)$? ← strong-balanced

- **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?**

weak-balanced

7.2 Performance of quicksort

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

- **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)
1  if p < r
2      q = PARTITION(A, p, r)
3      QUICKSORT(A, p, q − 1)
4      QUICKSORT(A, q + 1, 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
```

- **Worst-case partitioning**

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

- **Unbalanced partitioning**

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

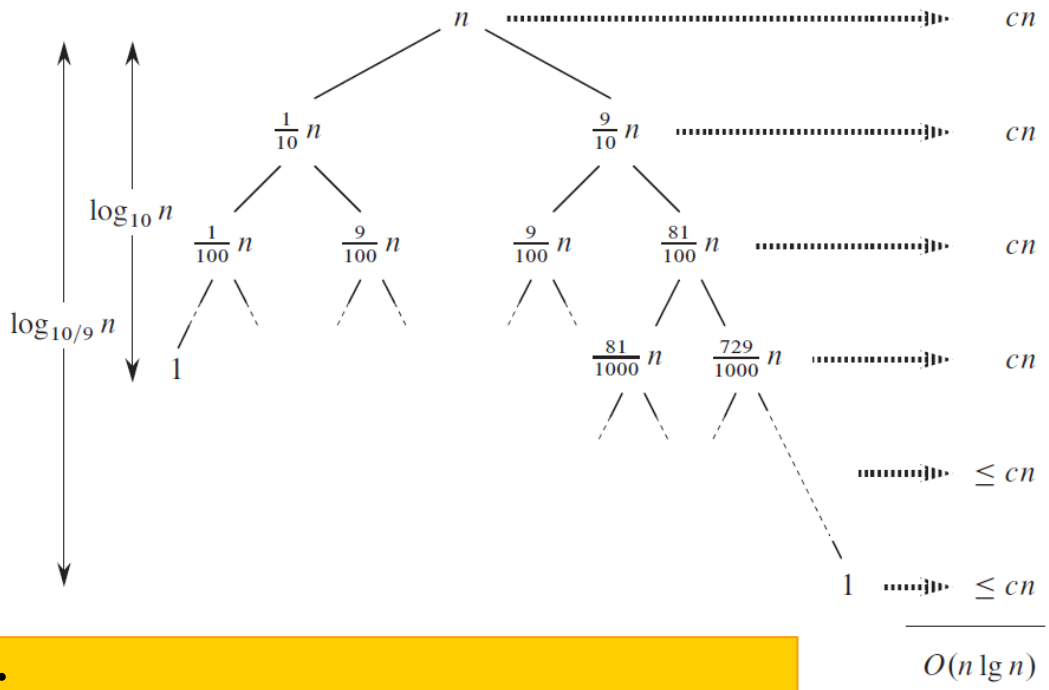
7.2 Performance of quicksort

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

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)

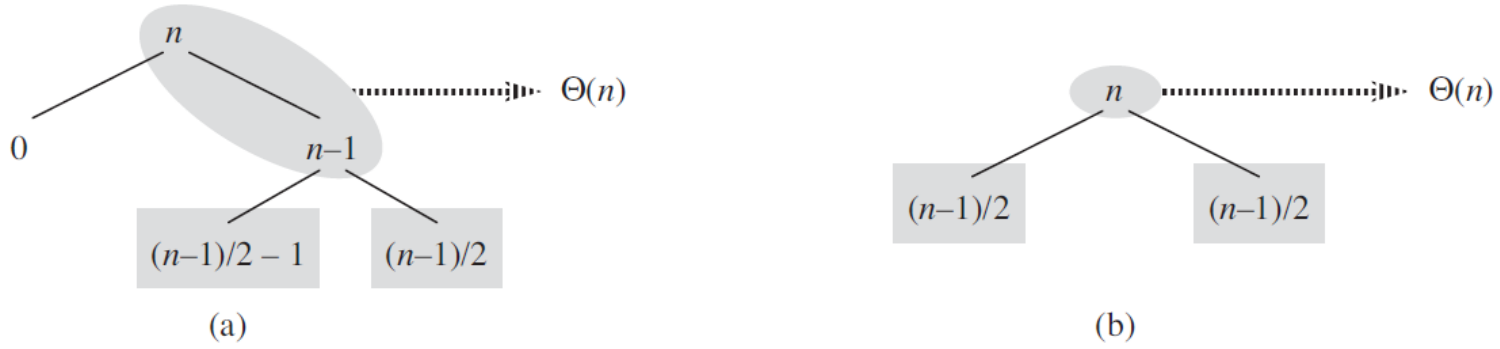
7.2 Performance of quicksort

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

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.



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

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

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

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。

7.4 Analysis of quicksort

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

```
RANDOMIZED-PARTITION(A, p, r)
1  i = RANDOM(p, r)
2  exchange A[r] with A[i]
3  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
```

- **Worst-case analysis**

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

  $0 \le q \le n-1$

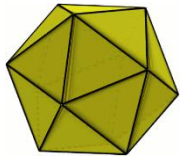
  Substitution method, $\Theta(n^2)$

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

7.4 Analysis of quicksort (1)

```
Times-QS
1  for i = 1 to m  //(m = k·n)
2       RANDOMIZED-QUICKSORT(A, p, r)
```

```
RANDOMIZED-QUICKSORT(A, p, r)
1  if p < r
2       q = RANDOMIZED-PARTITION(A, p, r)
3       RANDOMIZED-QUICKSORT(A, p, q − 1)
4       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

$T(n) = T(n\text{-}1) + T(0) + \Theta(n)$ -------- k times

$T(n) = T(n\text{-}2) + T(1) + \Theta(n)$ -------- k times

$T(n) = T(n\text{-}3) + T(2) + \Theta(n)$ -------- k times

…

$T(n) = T(n\text{-}n/2) + T(n/2\text{-}1) + \Theta(n)$ -- k times

…

$T(n) = T(2) + T(n\text{-}3) + \Theta(n)$ -------- k times

$T(n) = T(1) + T(n\text{-}2) + \Theta(n)$ -------- k times

$T(n) = T(0) + T(n\text{-}1) + \Theta(n)$ -------- k times

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 ($x << n$?):

$$\frac{xn^2 + (n-x)n\lg n}{n}$$

$$= \frac{xn^2 + n^2\lg n - xn\lg n}{n}$$

$$= xn + n\lg n - x\lg n$$

$$\le xn + n\lg n$$

$$\le n\lg n + n\lg n \quad \cdots \text{ (if } x \le \lg n)$$

$$= 2n\lg n$$

7.4 Analysis of quicksort (2)

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

```
RANDOMIZED-PARTITION(A, p, r)
1  i = RANDOM(p, r)
2  exchange A[r] with A[i]
3  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 $z_1, z_2, \ldots, z_n$**
- **$Z_{ij} = \{ z_i, z_{i+1}, \ldots, z_j \}$: set of elements between z_i and z_j, inclusive. z_i is the ith smallest element.**
- **Indicator random variables:**

  $X_{ij} = I\{ z_i$ 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 (2) - Indicator random variables

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

```
RANDOMIZED-PARTITION(A, p, r)
1  i = RANDOM(p, r)
2  exchange A[r] with A[i]
3  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.
- $Z_{ij} = \{ z_i, z_{i+1}, \cdots, z_j \}$
- Indicator random variables:

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

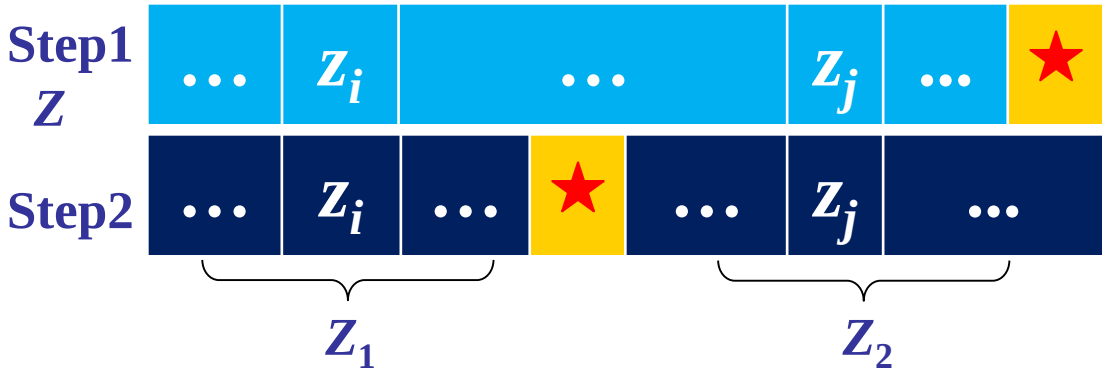
$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}$$

- 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 (2) - Indicator random variables

```
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
```

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，哨兵（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

Z 中的其他元素仅与★比较一次

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

(2) 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）

(3) 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 (2)

- **Indicator random variables:**

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

- **The total number of comparisons**

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

```
PARTITION(A, p, r)
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
```

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

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

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

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

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

```
RANDOMIZED-PARTITION(A, p, r)
1  i = RANDOM(p, r)
2  exchange A[r] with A[i]
3  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
```

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 => k = lg(n+1)

每次分区有最多n-1次比较

⇨ $O(n\lg n)$

Exercise for chapter 7

- 随机产生一组数据（如1M），多次运行quicksort算法，观察partition算法中第4行（元素比较）执行的次数，并对你的观察结果进行思考。
- 研究库函数 qsort 的原理。

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

```
RANDOMIZED-PARTITION(A, p, r)
1  i = RANDOM(p, r)
2  exchange A[r] with A[i]
3  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
```

8 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.

Lower bounds for comparison sort

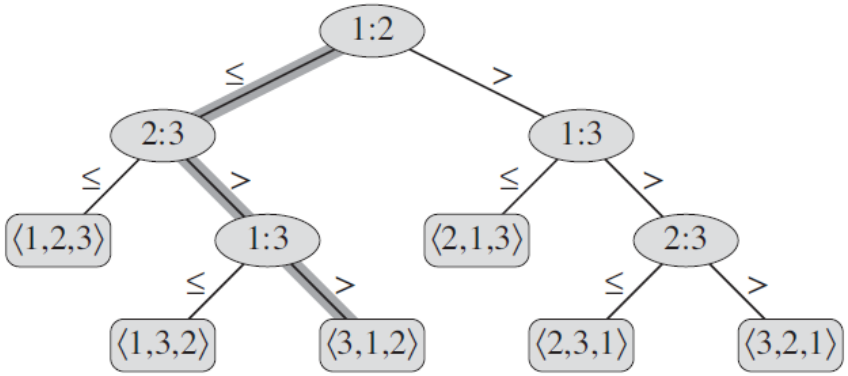
- **Algorithms**
  - ✓ **bubble, select, insert, merge, heap, quick, …**
- **Comparison sort**
  - ✓ **The sorted order they determine is based only on comparisons between the input elements.**
  - ✓ **We use only comparisons between elements to gain order information about an input sequence $<a_1, a_2,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

$\Omega(n\lg n)$? （计算时间至少为 $n\lg n$，最好情况除外）

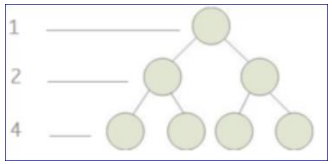
The decision-tree model

- **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）

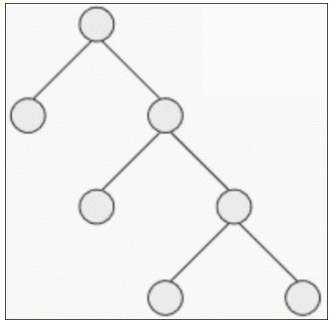
Example: The decision tree for insertion sort operating on three elements



“中国版”的满二叉树

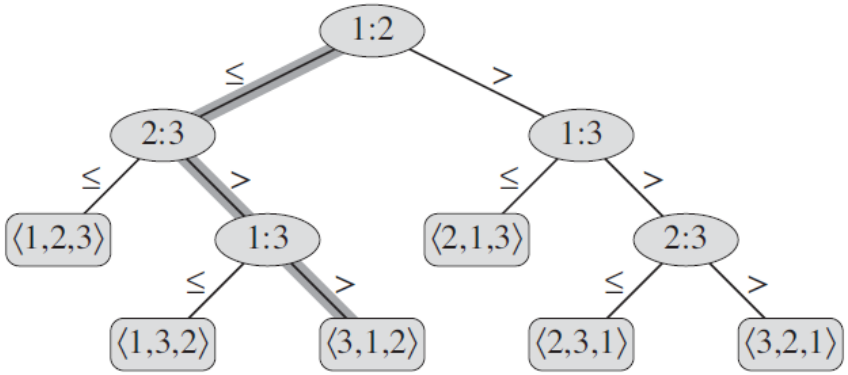


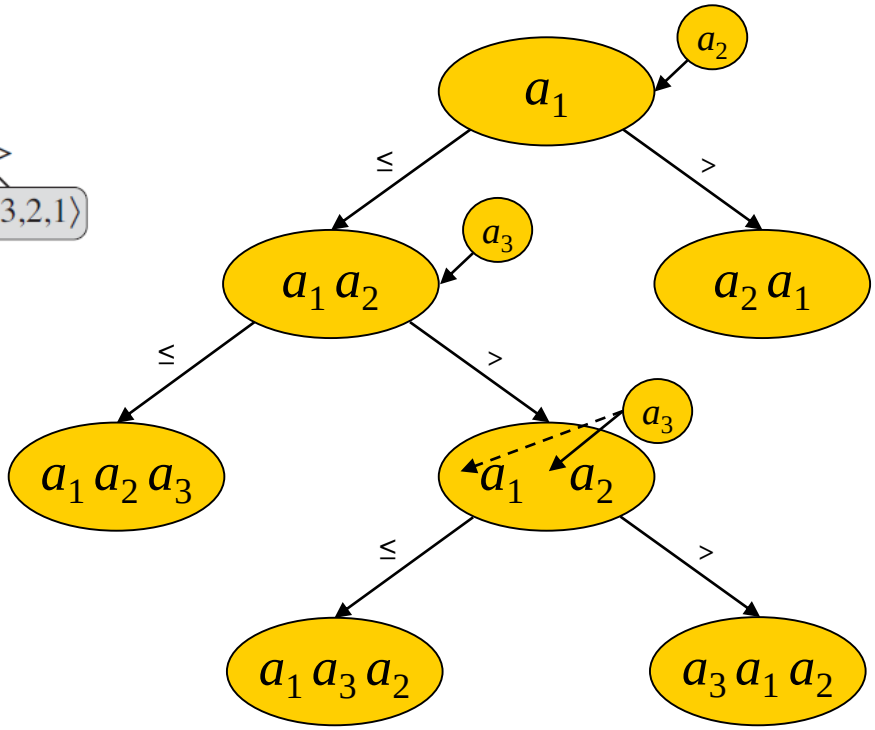
“外国版”的满二叉树（结点要么叶子，要么有两个孩子）



The decision-tree model

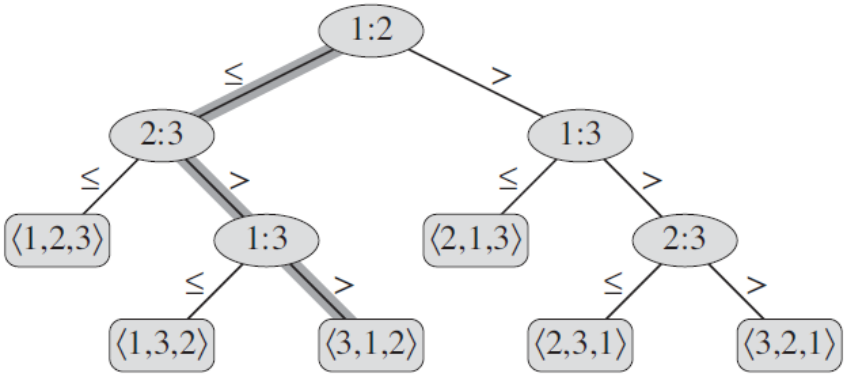
Example: The decision tree for insertion sort operating on three elements





The decision-tree model

- **In a decision tree, each leaf is a permutation (a solution of sort)** $\langle \pi(1), \pi(2), \ldots, \pi(n) \rangle$ **of** $\langle 1, 2, \ldots, n \rangle$



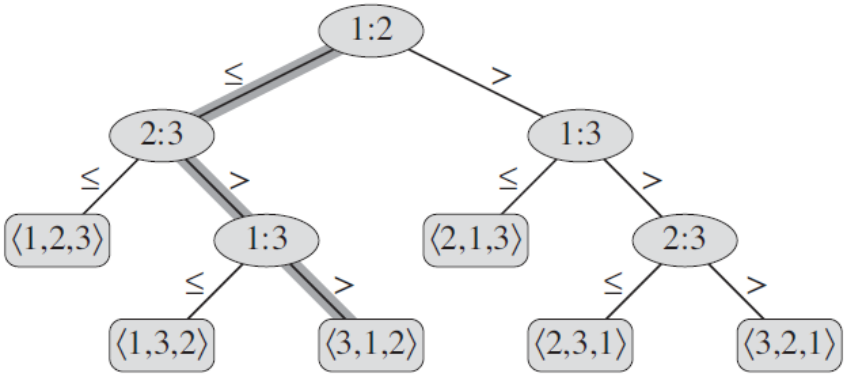
- **There have** $n!$ **permutations of** $\langle 1, 2, \ldots, n \rangle$
- **A correct sorting algorithm must be able to produce a permutation(leaf) that establish the ordering**

$$a_{\pi(1)} \leq a_{\pi(2)} \leq \ \ldots \ \leq a_{\pi(n)}$$

- **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 $\ge n$ 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

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

The decision-tree model

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

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

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

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

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

两边都取对数

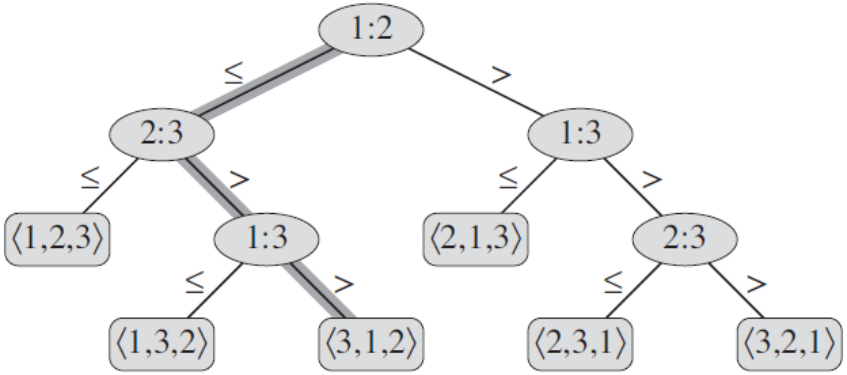
$(n/2)\lg(n/2) < \lg(n!) < n\lg n$

$(n/2)\lg(n/2) = (n/2)\lg n - n/2 > (n/3)\lg n$

$\therefore\ c_1 * n\lg n < \lg(n!) < c_2 * n\lg n$

这里 $c_1 = 1/3$, $c_2 = 1$

The decision-tree model



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

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

Google 学术搜索 A tournament problem

文章 找到约 231,000 条结果 (用时0.04秒)

时间不限
2021以来
2020以来
2017以来
自定义范围...

A tournament problem
LR Ford Jr, SM Johnson - The American Mathematical Monthly, 1959 - Taylor & Francis
Introduction. In his book, t Steinhaus discusses the problem of ranking n objects according to some transitive characteristic, by means of successive pairwise comparisons. In this paper we shall adopt the terminology of a tennis tournament by n players. The problem may be …
☆ 99 被引用次数: 183 相关文章 所有 6 个版本

A tournament problem

在引用文章中搜索

[图书] Introduction to algorithms
TH Cormen, CE Leiserson, RL Rivest, C Stein - 2009 - books.google.com
Some books on algorithms are rigorous but incomplete; others cover masses of material but lack rigor. Introduction to Algorithms uniquely combines rigor and comprehensiveness. The book covers a broad range of algorithms in depth, yet makes their design and analysis …
☆ 99 被引用次数: 59543 相关文章 所有 72 个版本 »

9 Medians and Order Statistics

The *i*th **order statistic** of a set of *n* elements is the *i*th smallest element.

- ✓ the **minimum** of a set of elements is the first order statistic ($i = 1$).
- ✓ the **maximum** is the *n*th order statistic ($i = n$).
- ✓ A **median**, informally, is the "halfway point" of the set.

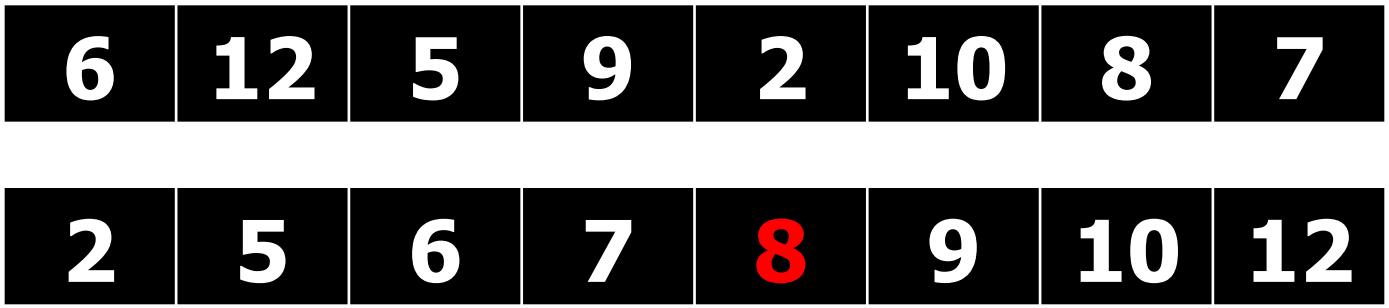
6	12	5	9	2	10	8	7

Minimum: 2

Maximum: 12

9 Medians and Order Statistics

- 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?



The 5th **order statistic** is 8

```
MINIMUM(A)
1  min = A[1]
2  for i = 2 to A.length
3       if min > A[i]
4            min = A[i]
5  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: $\Theta(n)$.

求第 i 小的元素

Firstly, p is 1, r is n

```
RANDOMIZED-SELECT(A, p, r, i)
1  if p == r
2        return A[p]
3  q = RANDOMIZED-PARTITION(A, p, r)
4  k = q − p + 1
5  if i == k                    // the pivot value is the answer
6        return A[q]
7  elseif i < k
8        return RANDOMIZED-SELECT(A, p, q − 1, i)
9  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 $A[q]$

// 左边的 k-1 个元素比 $A[q]$ 小，第 k 大的元素是 $A[q]$

9.2 Selection in expected linear time

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

```
RANDOMIZED-SELECT(A, p, r, i)
1  if p == r
2       return A[p]
3  q = RANDOMIZED-PARTITION(A, p, r)
4  k = q − p + 1
5  if i == k            // the pivot value is the answer
6       return A[q]
7  elseif i < k
8       return RANDOMIZED-SELECT(A, p, q − 1, i)
9  else return RANDOMIZED-SELECT(A, q + 1, r, i − k)
```

- **Worst-case running time**

  $T(n) = T(n\text{-}1) + O(n)$,

  $\Theta(n^2)$

- **A special case**

  $q = (r\text{-}p)/2$, 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

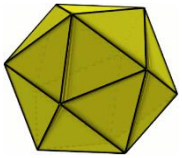
$T(n) = T(\max(k\text{-}1, n\text{-}k)) + O(n)$

```
RANDOMIZED-SELECT(A, p, r, i)
1  if p == r
2      return A[p]
3  q = RANDOMIZED-PARTITION(A, p, r)
4  k = q − p + 1
5  if i == k            // the pivot value is the answer
6      return A[q]
7  elseif i < k
8      return RANDOMIZED-SELECT(A, p, q − 1, i)
9  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.)

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

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

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

Using substitution,

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

9.2 Selection in expected linear time

```
RANDOMIZED-SELECT(A, p, r, i)
1  if p == r
2       return A[p]
3  q = RANDOMIZED-PARTITION(A, p, r)
4  k = q − p + 1
5  if i == k            // the pivot value is the answer
6       return A[q]
7  elseif i < k
8       return RANDOMIZED-SELECT(A, p, q − 1, i)
9  else return RANDOMIZED-SELECT(A, q + 1, r, i − k)
```

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

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

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

Using substitution,

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

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$n = 10^6$) 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

有的课，1600+ 人的榜单实时刷新

思考

“370万”的实时评测记录，如何高效排序、查找

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				
3703660		4610	Wrong Answer	0.8	c	338	65	1732	2021-10-10 17:23:04
3703659		4624	Accepted	1	c	635	264	1740	2021-10-10 17:22:18
3703658		4646	Accepted	1	c	883	20039	1688	2021-10-10 17:22:15
3703657		4536	Time Limit Exceed	0	c	246	2185	1684	2021-10-10 17:22:06
3703656		4643	Wrong Answer	0.1	c	385	325	1708	2021-10-10 17:21:39
3703655		4642	Compile Error	0	c	2522	0	0	2021-10-10 17:20:14
3703654		4610	Wrong Answer	0	c	330	74	1732	2021-10-10 17:20:05
3703653		4581	Accepted	1	c	2325	113	1688	2021-10-10 17:19:26
3703652		4630	Wrong Answer	0.405941	c	568	455	2568	2021-10-10 17:19:19
3703651		4646	Time Limit Exceed	0.6	c	908	18040	1676	2021-10-10 17:18:04
3703650		4646	Wrong Answer	0	c	752	24732	1684	2021-10-10 17:18:03
3703649		4643	Wrong Answer	0.1	c	397	110	1704	2021-10-10 17:17:57
3703648		4640	Wrong Answer	0	c	1886	498	1736	2021-10-10 17:16:57
3703647		4646	Wrong Answer	0	c	407	17949	1676	2021-10-10 17:16:40
3703646		4640	Time Limit Exceed	0	c	1866	10000	0	2021-10-10 17:16:03

首页 上一页 1 下一页