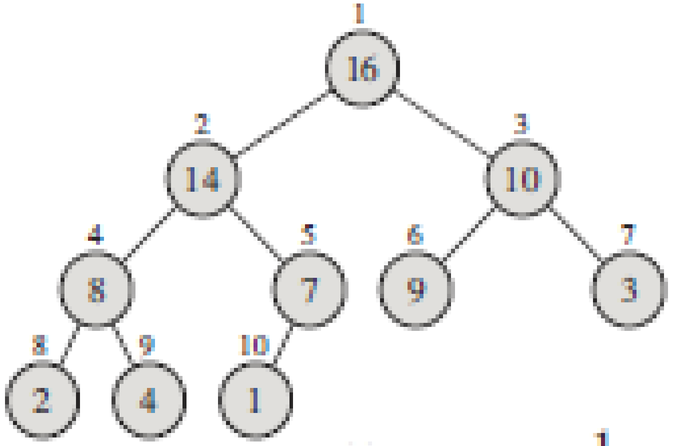
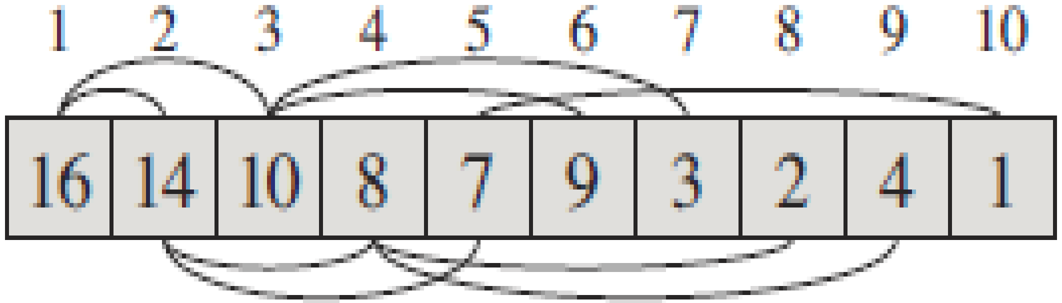
计算机问题求解 – 论题2-12
– 堆与堆排序

2016年05月05日





问题1:

为什么有时可以将数组理解为二叉树?

为什么数组会有一个A.heap-size?

问题2:

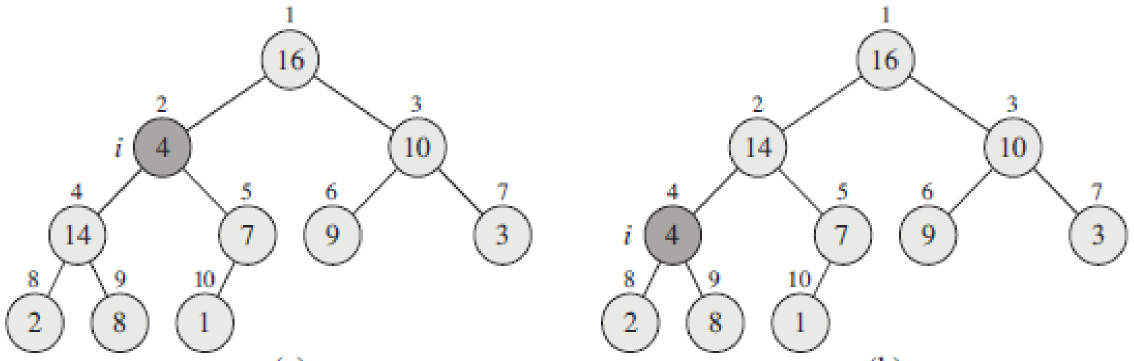
堆与我们上次讨论的队列与栈最突出的差别是什么?

其特征与对象的内容相关，
一定是源于具体应用。

堆（偏序树）性质

- 树*T* 满足偏序树性质 当且仅当 树中任一结点的键值不小于（或不大于）其子结点（如果有）的键值。
- 此性质在数组实现中的表示：
  - Max-heap: $A[\textsc{Parent}(i)] \geq A[i]$
  - Min-heap: $A[\textsc{Parent}(i)] \leq A[i]$

如果我们要定义堆的**ADT**，在其数据部分，我们应该给出什么约束？

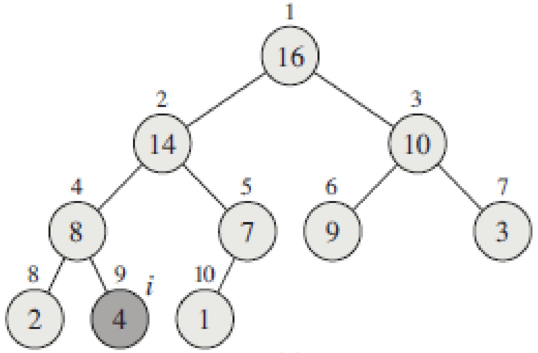


(a) (b)

问题3：Max-Heapify的precondition是什么？

特别解释一下

largest



(c)

```
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)
```

问题4:

你能利用上图解释Max-Heapify吗？

Worst-case Analysis for Max-Heapify

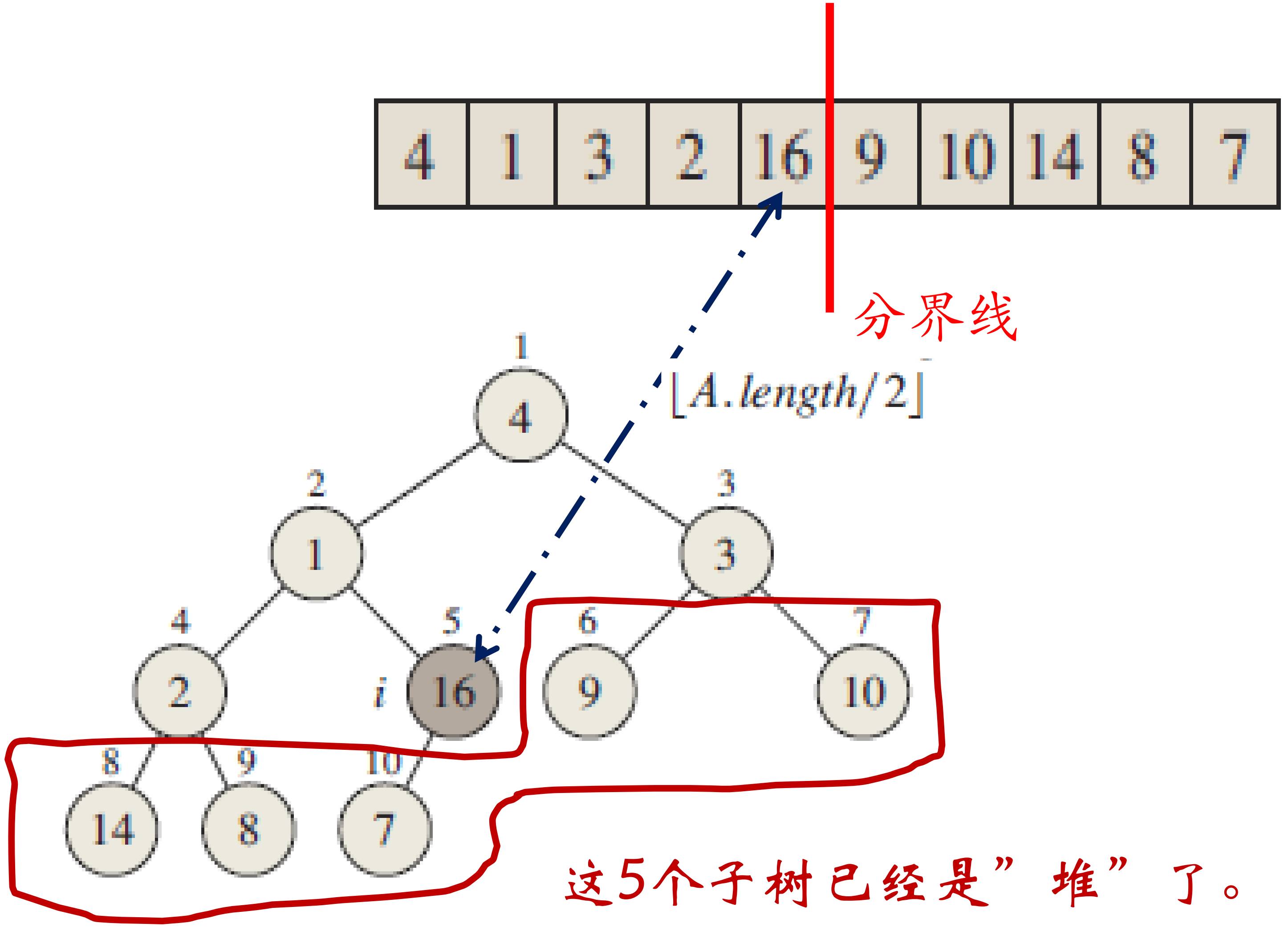
- 过程Max-Heapify中不包含循环，所以，如果不递归，其代价是*O*(1)。
  - 如果考虑比较运算的次数，每“下沉”一层，执行2次比较。

问题5：为什么？

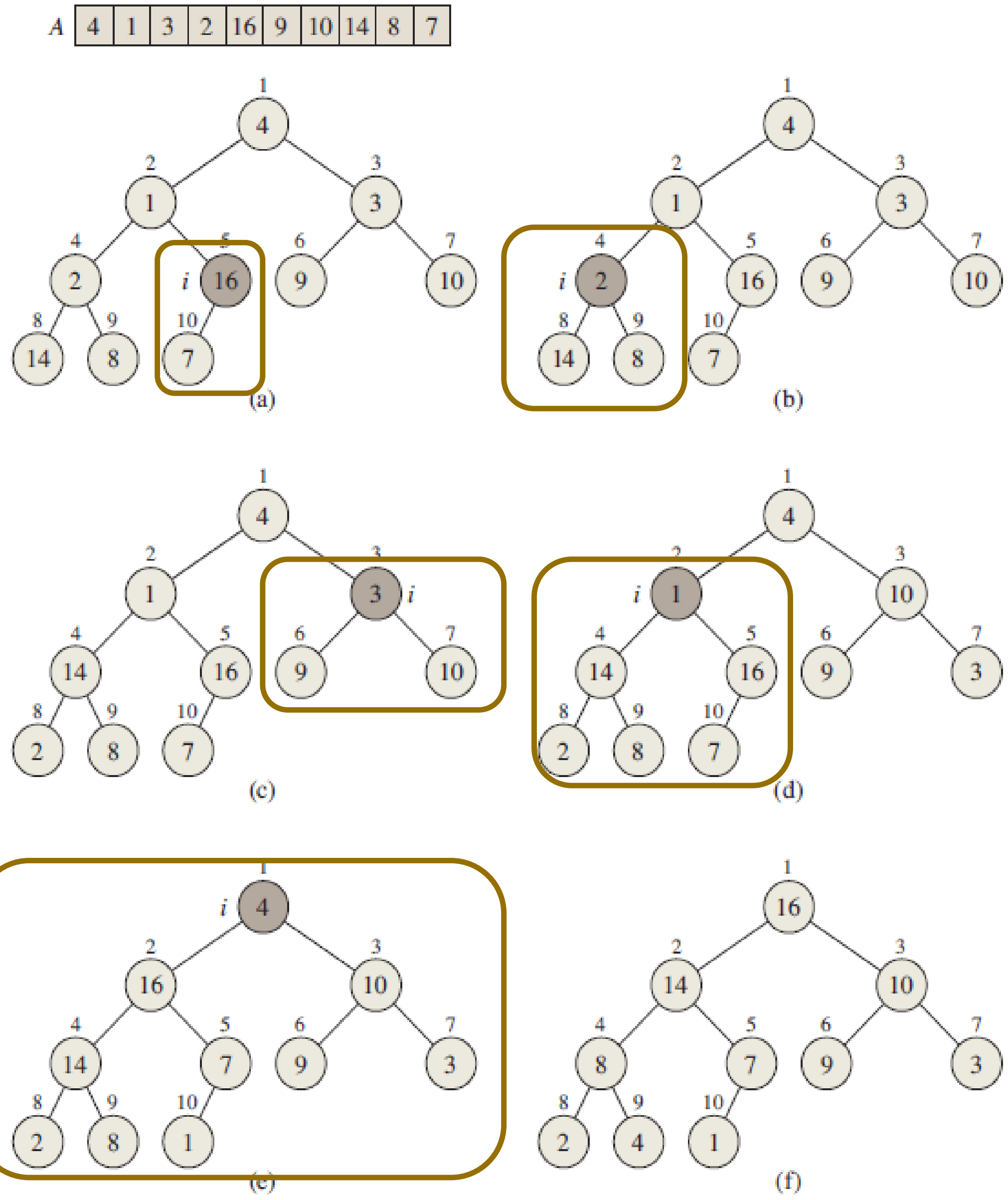
- 递归： $T(n) \le T(2n/3) + \Theta(1)$

The solution to this recurrence, by case 2 of the master theorem (Theorem 4.1), is $T(n) = O(\lg n)$. Alternatively, we can characterize the running time of MAX-HEAPIFY on a node of height h as $O(h)$.

造“堆”：自底向上



4 1 3 2 16 9 10 14 8 7
分界线
⌊A.length/2⌋
1
4
2
1
3
3
4
2
5
i 16
6
9
7
10
8
14
9
8
10
7
这5个子树已经是”堆”了。



问题6：这个循环的 invariant 是什么？

BUILD-MAX-HEAP(A)

$A.heap\text{-}size = A.length$

for $i = \lfloor A.length/2 \rfloor$ **downto** 1

  MAX-HEAPIFY(A, i)

Built-Max-Heap正确性证明

At the start of each iteration of the **for** loop of lines 2–3, each node $i+1, i+2, \ldots, n$ is the root of a max-heap.

Initialization: Prior to the first iteration of the loop, $i = \lfloor n/2 \rfloor$. Each node $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \ldots, n$ is a leaf and is thus the root of a trivial max-heap.

Maintenance: To see that each iteration maintains the loop invariant, observe that the children of node i are numbered higher than i. By the loop invariant, therefore, they are both roots of max-heaps. This is precisely the condition required for the call MAX-HEAPIFY(A, i) to make node i a
the MAX-HEAPIFY call preserves the property th
are all roots of max-heaps. Decrementing i in t
the loop invariant for the next iteration.

Termination: At termination, $i = 0$. By the loop invariant, … , n is the root of a max-heap. In particular, node 1 is.

为什么算法是
downto 1，而不是
upto length/2？

A Poor Upper Bound

We can compute a simple upper bound on the running time of BUILD-MAX-HEAP as follows. Each call to MAX-HEAPIFY costs $O(\lg n)$ time, and BUILD-MAX-HEAP makes $O(n)$ such calls. Thus, the running time is $O(n \lg n)$. This upper bound, though correct, is not asymptotically tight.

问题7:

为什么这个Bound不很好？

关于堆的两点数学知识

假设二叉树的高度是h，结点数是n，则：

$$h = \lfloor \lg n \rfloor$$

n个元素的堆所包含的

高度为h的结点个数最多是：

$$\left\lceil \frac{n}{2^{h+1}} \right\rceil$$

建堆的时间复杂度是线性的

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

其中：$\displaystyle\sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h} \in O\left(\sum_{h=0}^{\infty} \frac{h}{2^h} \right)$，而 $\displaystyle\sum_{h=0}^{\infty} \frac{h}{2^h} = \sum_{h=0}^{\infty} h\left(\frac{1}{2}\right)^h$

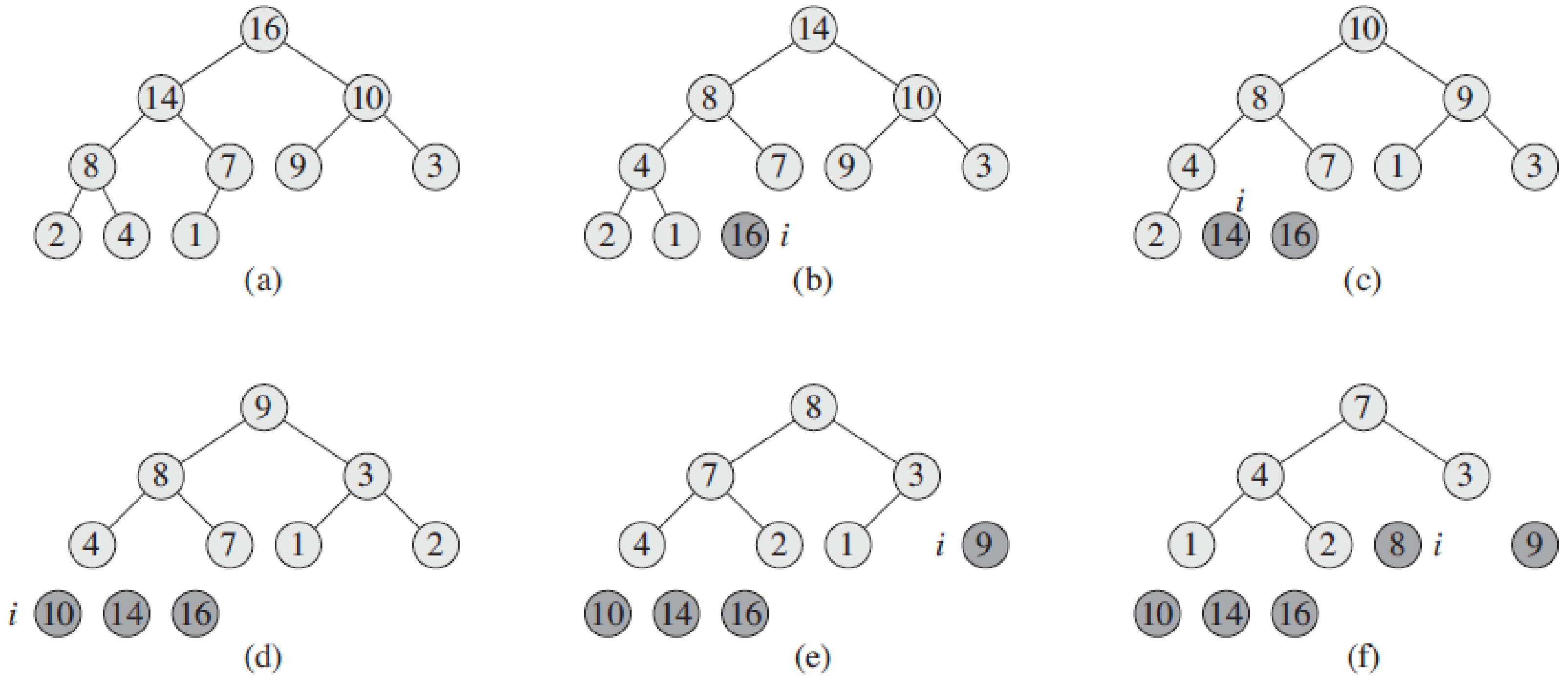
即：$\displaystyle\sum_{h=0}^{\infty} hx^h, (x = \frac{1}{2})$，　而 $\displaystyle\sum_{h=0}^{\infty} hx^h = \frac{x}{(1-x)^2}$

$\therefore$

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

堆排序

```
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)
```



堆排序算法：

问题9:

怎么体现in-place，即“原地输出”？

问题10：

你能解释为什么复杂度是$O(n\lg n)$，这是**worst-case**，还是**average**？

问题11：

你能否通过比较priority-queue与一般的queue，说明抽象数据类型对于计算机问题求解的意义？

Max-Priority Queue

INSERT(S, x) inserts the element x into the set S, which is equivalent to the operation $S = S \cup \{x\}$.

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.

抽象数据类型是为了减轻人思考的负担，而不是为了减轻计算机执行的负担。关键是如何实现！

实现：Array $\longrightarrow$ Heap $\longrightarrow$ Priority Queue

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

```
HEAP-EXTRACT-MAX(A)
if A.heap-size < 1
    error "heap underflow"
max = A[1]
A[1] = A[A.heap-size]
A.heap-size = A.heap-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)
```

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

Open Topics：

1，写出堆的ADT及其形式规约

2，用二叉树→堆→优先队列的方式给出优先队列的实现

3，堆排序是stable的吗？证明或举例

家庭作业

- TC pp.153-: ex.6.1-2, 6.1-4, 6.1-7
- TC pp.156-: ex.6.2-2, 6.2-5, 6.2-6
- TC pp.159-: ex.6.3-3
- TC pp.160-: ex.6.4-2, 6.4-4
- TC pp.164-: ex.6.5-5, 6.5-7, 6.5-9