# 算法分析与设计: 第三次作业

PB20061372　朱云沁　Apr. 11, 2023

**6.1-6** 值为 $\langle 23, 17, 14, 6, 13, 10, 1, 5, 7, 12\rangle$ 的数组是一个最大堆吗?

**解:** 完全二叉树结构如下图所示.

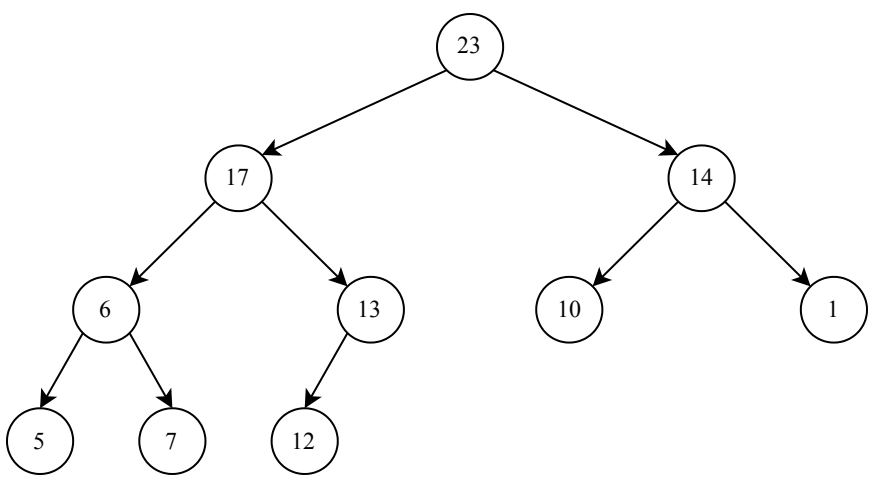


Figure 1. A complete binary tree with values $\langle 23, 17, 14, 6, 13, 10, 1, 5, 7, 12\rangle$.

存在子节点的值 7 大于其父节点的值 6, 因此不是一个最大堆.

**6.2-5** Max-Heapify 的代码效率较高, 但第 10 行中的递归调用可能例外, 他可能是某些编译器产生低效的代码. 请用循环控制结构取代递归, 重写 Max-Heapify 代码 (写出伪代码).

**解:** 伪代码如下:

**Algorithm 1:** 6.2-5

**Input:** a complete binary tree stored in an array $A$ and an index $i$ into the array such that the subtrees rooted at $\textsc{Left}(i)$ and $\textsc{Right}(i)$ are max-heaps.
**Output:** a max-heap rooted at index $i$.
**function** $\textsc{Max-Heapify}(A, i)$:

```
largest ← i
repeat:
    i ← largest
    l ← Left(i)
    r ← Right(i)
    if l ≤ Heap-Size(A) and A[l] > A[largest]:
        largest ← l
    if r ≤ Heap-Size(A) and A[r] > A[largest]:
        largest ← r
    Swap(A[i], A[largest])
until i ≠ largest
```

**6.3-1** 参照图 6-3 的方法,说明 Build-Max-Heap 在数组 $A = \langle 5, 3, 17, 10, 84, 19, 6, 22, 9\rangle$ 上的操作过程.

**解:** 过程如下图所示.

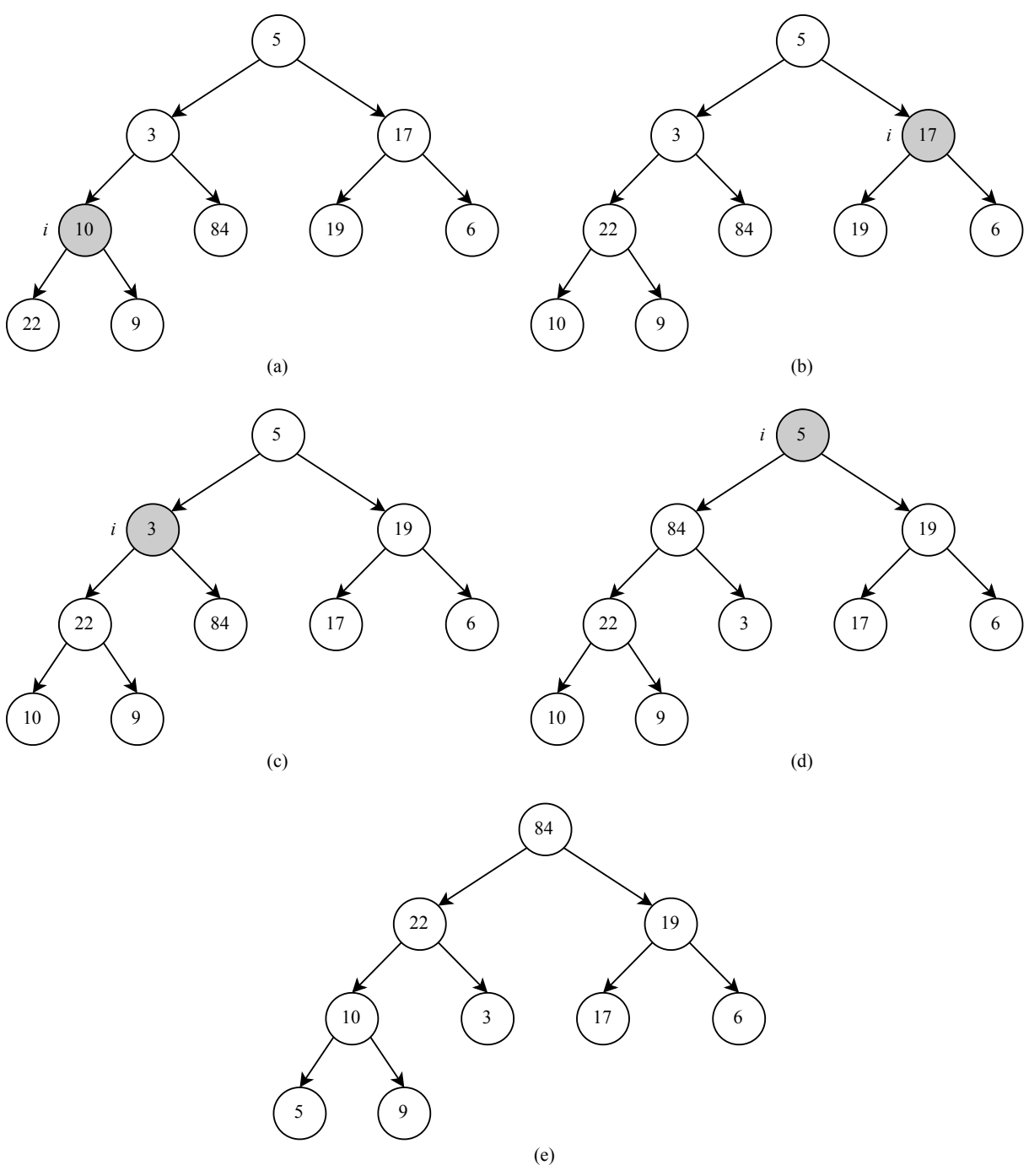


Figure 2. Build-Max-Heap on the array $\langle 5, 3, 17, 10, 84, 19, 6, 22, 9\rangle$. (a) Before the first call to Max-Heapify, the variable $i$ points to the node 4. (b) In the second iteration, $i$ points to the node 3. (c) In the third iteration, $i$ points to the node 2. (d) In the fourth iteration, $i$ points to the node 1. (e) After the execution of Build-Max-Heap, the array $A$ is a max-heap.

**6.5-9** 请设计一个时间复杂度为 $O(n \lg k)$ 的算法, 它能够将 $k$ 个有序链表合并为一个有序链表, 这里 $n$ 是所有输入链表包含的总的元素个数. (提示: 使用最小堆来完成 $k$ 路归并)

**解:** 伪代码如下:

**Algorithm 2:** 6.5-9

**Input:** $k$ sorted linked lists $L_1, L_2, \ldots, L_k$.
**Output:** a linked list $L$ containing all elements of $L_1, L_2, \ldots, L_k$ in ascending order.

```
Create an empty list L.
Create an empty min-heap H, which stores list nodes and compares their values.
for i ← 1 to k:
    Insert(H, Head(L_i))
while H is not empty:
    x ← Extract-Min(H)
    Insert(L, Value(x))
    if Next(x) exists:
        Insert(H, Next(x))
```

其中, 建堆过程可视为调用 Build-Min-Heap, 时间复杂度为 $O(k)$; **while** 循环共执行 $n$ 次, 每次迭代, 删除并返回堆中最小元素的时间复杂度为 $O(\lg k)$, 向链表、堆中插入元素的时间复杂度分别为 $O(1)$、$O(\lg k)$, 因此总时间复杂度为:

$$O(k) + n(O(1) + O(\lg k)) = O(n \lg k).$$

**7.1-2** 当数组 $A[p..r]$ 中的元素都相同时, Partition 返回的 $q$ 值是什么? 修改 Partition , 使得当数组 $A[p..r]$ 中所有元素的值都相同时, $q = \left\lfloor \frac{p+r}{2} \right\rfloor$. 请给出修改后的 Partition 伪代码.

**解:** 修改前, 返回的 $q$ 值是 $r$; 修改后的伪代码如下:

**Algorithm 3:** 7.1-2

**Input:** an array $A$ and two indices $p$ and $r$ such that $p \le r$.
**Output:** an index $q$ such that $p \le q \le r$ and $A[p..q-1] \le A[q] \le A[q+1..r]$.
**function** $\textsc{Partition}(A, p, r)$:

```
x ← A[r]
i ← p − 1
for j ← p to r − 1:
    if A[j] < x or (A[j] = x and j ≡ p + 1 (mod 2)):
        i ← i + 1
        Swap(A[i], A[j])
Swap(A[i + 1], A[r])
return i + 1
```

当数组 $A[p..r]$ 中的元素都相同时, 满足 $j \equiv p + 1 \pmod 2$ 的元素个数为 $\left\lfloor \frac{r-p}{2} \right\rfloor$, 因此返回的 $q$ 值为

$$q = p - 1 + \left\lfloor \frac{r-p}{2} \right\rfloor + 1 = \left\lfloor \frac{p+r}{2} \right\rfloor.$$

**7.2-5** 假设快速排序的每一层所做的划分的比例都是 $1 - a : a$, 其中 $0 < a \le \frac{1}{2}$ 且是一个常数. 试证明: 在相应的递归树中, 叶子节点的最小深度大约是 $-\frac{\lg n}{\lg a}$, 最大深度大约是 $-\frac{\lg n}{\lg(1-a)}$ (无需考虑整数舍入问题).

**解:** 每层划分, 两个子问题的数组长度大约为原问题的 $1 - a$ 和 $a$ 倍, 其中 $1 - a \ge \frac{1}{2} \ge a$. 当进行第 $k$ 层划分时, 子问题的最小数组长度大约为 $na^k$, 最大数组长度大约为 $n(1-a)^k$, 当数组长度小于等于 1 时, 子问题对应递归树的叶子节点. 设叶子节点的深度为 $h$, 则有

$$na^h \le 1 \le n(1-a)^h \implies -\frac{\lg n}{\lg a} \le h \le -\frac{\lg n}{\lg(1-a)}.$$

等号可以取到. 故叶子节点的最小深度大约是 $-\frac{\lg n}{\lg a}$, 最大深度大约是 $-\frac{\lg n}{\lg(1-a)}$.