减治法

本章学习目标

- 熟练掌握减治法的设计思想
- 熟练掌握折半查找、插入排序、选择问题
- 掌握二叉树查找、堆排序、假币问题
- 理解淘汰赛冠军问题

1.概述

- 减治法的设计思想
- 把一个复杂问题，分解为若干个相互独立的子问题。原问题的解与子问题的解之间存在某种确定的关系。通过求解一个子问题的解，就可以得到原问题的解
- 这种关系通常表现为：
- 原问题的解只存在于一个子问题中
- 原问题的解与子问题的解存在某种对应关系

1.概述

- 注意：分治法与减治法的区别
- 分治法有两个子问题，而减治法只有一个子问题

两个序列的中位数

- 描述：一个长度为n的升序序列S，处在第n/2个位置的数，称为序列S的中位数。
- 两个序列的中位数，是它们所有元素的升序序列的中位数。
- 例如：S1={11,13,15,17,19}，S2={2,4,6,8,20}。序列S1的中位数是15；序列S1+S2的中位数是11。

两个序列的中位数算法

- 算法：
- 1、分别求出两个序列的中位数，记为a和b。
- 2、比较a和b的大小：
  - 2.1、若a=b，则a为答案；
  - 2.2、若a<b，则中位数只能出现在a与b之间（为什么？）。在A序列中舍弃a之前的元素，得到序列A1。在B序列中舍弃b之后的元素，得到序列B1。

两个序列的中位数算法

- 算法：
  - 2.3、若a>b，则中位数只能出现在b与a之间。在A序列中舍弃a之后的元素，得到序列A1。在B序列中舍弃b之前的元素，得到序列B1。
- 3、在A1和B1中分别求出中位数，重复步骤2，直到两个序列各只有一个元素；较小者即为答案。

两个序列的中位数伪代码

- 伪代码（非递归调用）：
- 输入：两个长度为n的有序序列A和B
- 输出：A和B的中位数
- 1、循环直到A和B都只有一个元素
  - 1.1、a=序列A的中位数
  - 1.2、b=序列B的中位数
  - 1.3、比较a和b，执行下列操作之一
    - 1.3.1、若a=b，则返回a，算法结束

两个序列的中位数伪代码

- 伪代码（非递归调用）：
  - 1.3.2、若a<b，则在序列A中舍弃a之前的元素，得到新序列A。在序列B中舍弃b之后的元素，得到新序列B。
  - 1.3.3、若a>b，则在序列A中舍弃a之后的元素，得到新序列A。在序列B中舍弃b之前的元素，得到新序列B。
- 2、序列A和B均只有一个元素，返回最小者

两个序列的中位数的算法分析

- 1、找一个升序序列的中位数，时间O(1)
- 2、求子序列。子序列的长度是原来的一半，所以共循环O(logn)次
- 3、最终，算法的时间复杂度O(logn)

- 因为每个序列，只求一个子序列，丢弃另一个子序列。所以是减治法。

2.查找问题中的减治法

- 折半查找
- 只需要一个子问题的解

- 二叉查找树
  - 左子树所有结点的值均小于根结点的值
  - 右子树所有结点的值均大于根结点的值
  - 左右子树均是二叉排序树

- 选择问题

2.1折半查找

- **问题：** 在一个有序的序列中查找某个元素
- 思想：每次比较后消除一半的元素
  - 找到序列的中间
  - 将中间位置的值与搜索元素进行比较
  - 如果它们相等-完成！否则，请确定序列的哪一半包含搜索关键字，在数组的另一半上重复搜索，而忽略另一半
  - 继续搜索，直到匹配关键字或没有要搜索的元素为止

折半查找的例子

序列

1
5
15
19
25
27
29
31
33
45
55
88
100

搜索元素 = 19

序列的中间

比较29和19

因为19小于29

所以下一个
搜索将使用序列的上半部分

折半查找的例子

序列

1
5
15
19
25
27

搜索元素 = 19

使用它作为序列
的中间

比较15与19

15小于19,
使用下半部分进行搜索

折半查找的例子

序列

搜索元素 = 19

19
25
27

使用它作为序列的中间
比较25与19

25 大于19，所以使用上半部分进行搜索

折半查找的例子

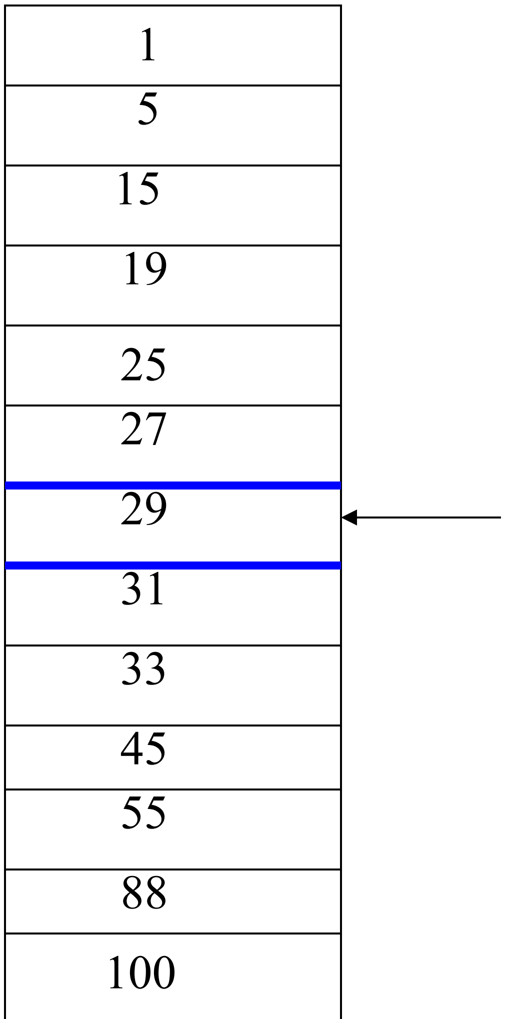
搜索元素 = 19

序列

使用它作序列的中间
比较它与19
找到了!

折半查找的例子2

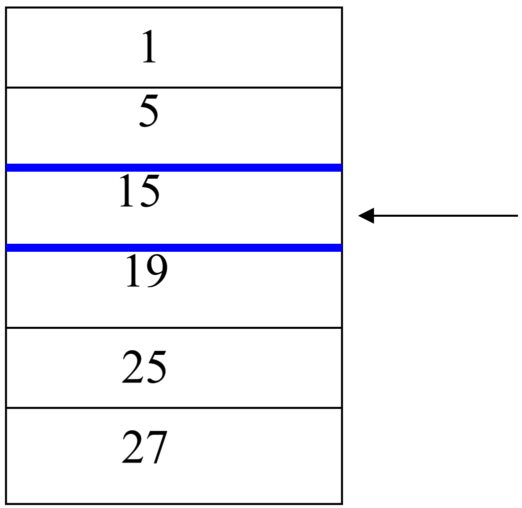
搜索元素 = 18



数组的中间

18小于29，所以下一个
搜索将使用数组的上半部分

折半查找的例子2



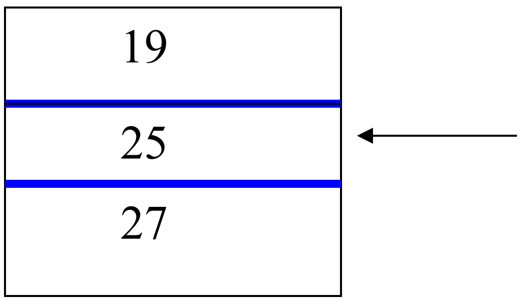
搜索元素 = 18

使用它作为数组的中间

15小于18，继续15的下半部分的搜索

折半查找的例子2

搜索元素 = 18



使用它作为数组的中间

25比18大，继续进行上半部分的搜索

折半查找的例子2

搜索元素 = 18

使用它作为数组的中间
比较它与18
不匹配，没有更多元素
比较。
未找到!

算法

- **输入：一个有序的数组a[1...n], 一个代找到的元素key**
- 算法:
- while (low <= high)
  - mid = (low + high) / 2;
  - if (key == a[mid]) 输出找到位置mid
  - elseif (key < a [mid]) high = mid -1;
  - else: low = mid + 1;

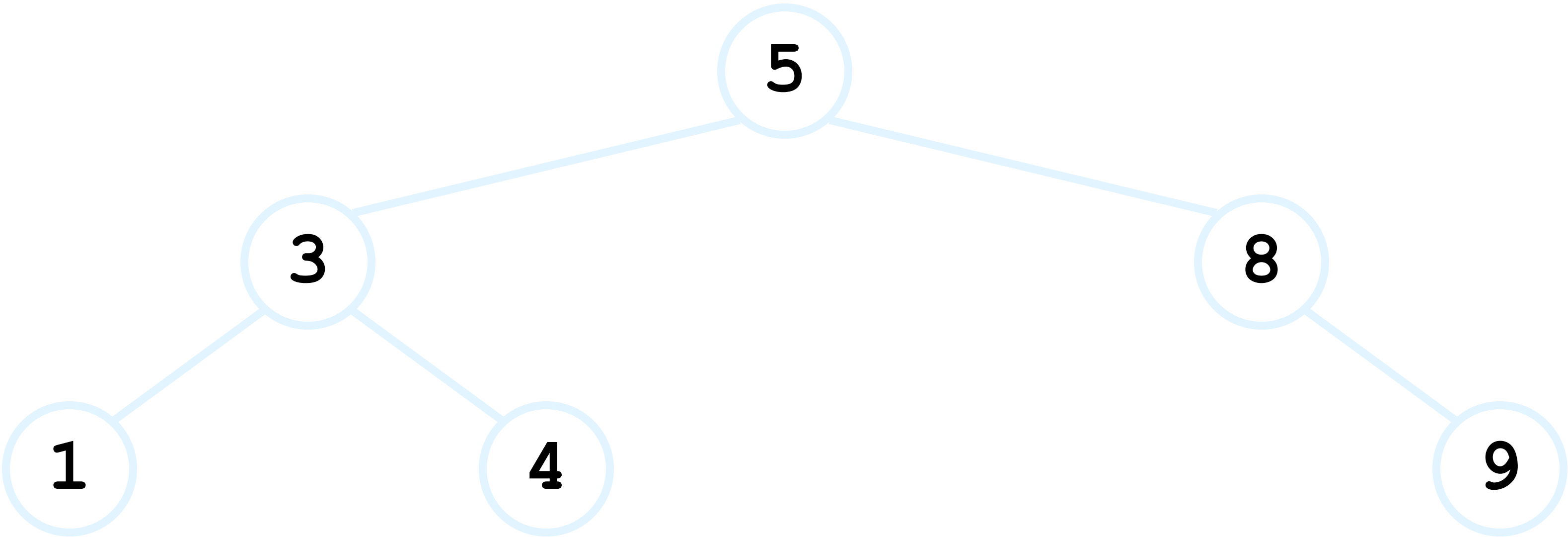
问题： 折半查找的递归算法

2.2 二叉查找树

- 问题：给定一棵二叉查找树T, 查找元素k
- 一个二叉树T的节点S包含元素：
  - *key*: 标识字段，指示总排序
  - *lchild*: 指向左孩子的指针（可以为NULL）
  - *rchild*: 指向正确子项的指针（可以为NULL）
  - *p*: 指向父节点的指针（root为NULL）

回顾: 二叉查找树BST

- BST的性质:
  key[leftSubtree(x)] ≤ key[x] ≤key[rightSubtree(x)]
- 例子:



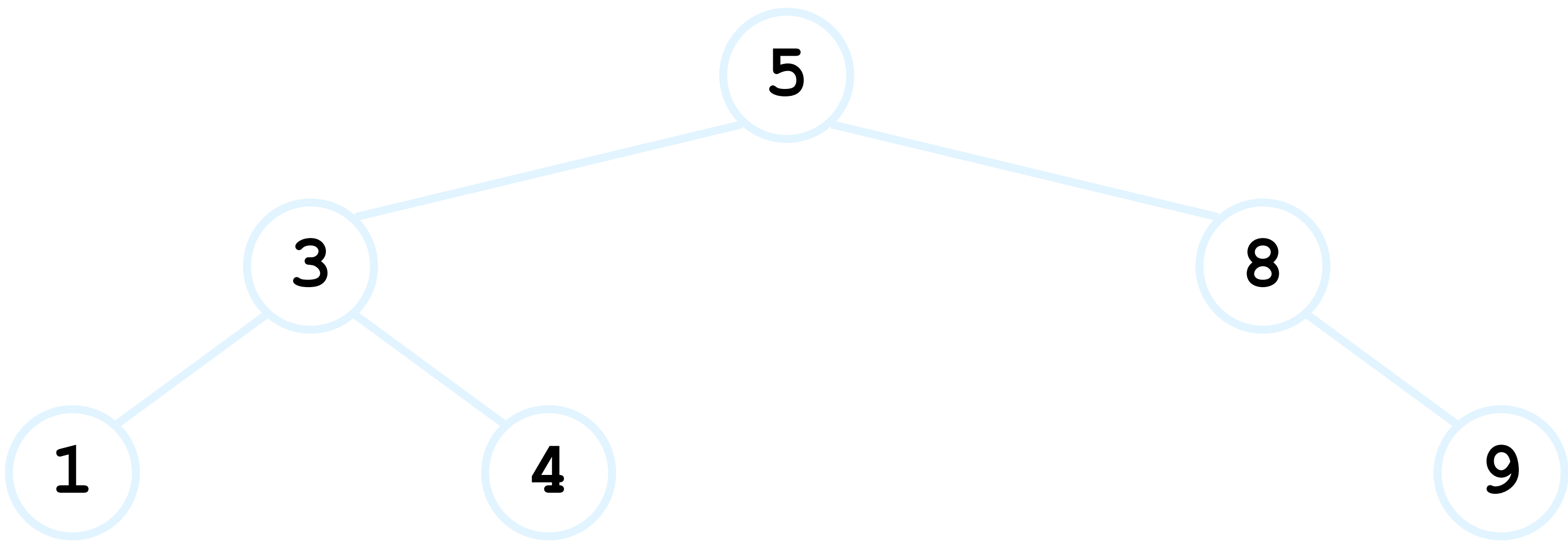
BST的操作: 搜素

- 给定值k和指向节点的指针x，则返回具有该值或NULL的节点
- **TreeSearch(x, k)**

```
➢ if (x = NULL  or  k = key[x])
    return x;
➢ if (k < key[x])
    return TreeSearch(left[x], k);
➢ else
    return TreeSearch(right[x], k);
```

二叉树搜索: 例子

- 搜索 *4* 和 2:

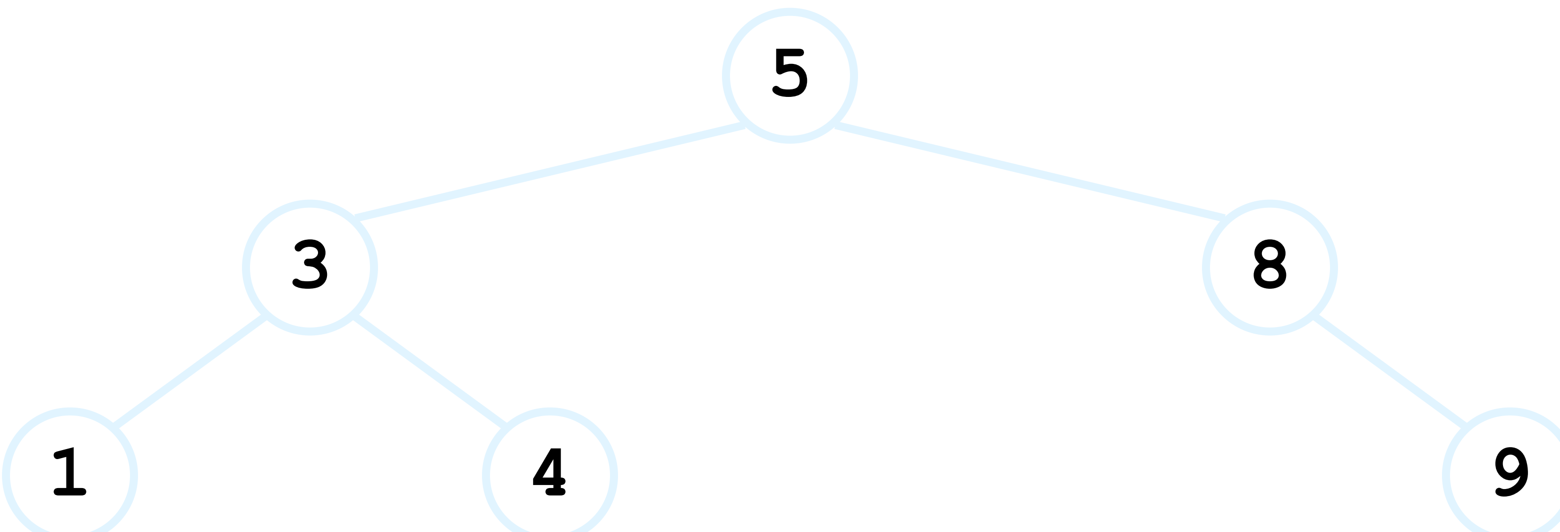


BST的操作: 插入

- 将元素x添加到树中， 使得BST的性质继续保持
- 基本算法
  - 像上面的搜索过程
  - 插入x代替NULL
- 新建BST即把数组的元素不断插入

二叉树插入: 例子

- 搜索 6



2.3 选择问题

- 设无序序列T=(r1,r2,…,rn)，T的第k（1≤k≤n）小元素定义为：
- T按升序排序后在第k个位置上的元素。
- 特别地，若k=n/2，则寻找第n/2小的元素的问题，称为中值问题

选择问题的减治法

- 选择问题的普通想法
- 先对序列T进行排序O(nlogn)，然后在一个有序的序列中查找第k个元素O(1)

- 选择问题的减治法
- 借用快速排序法的思想
- 左子问题的所有元素均小于选定元素
- 右子问题的所有元素均大于选定元素

选择问题的减治法

- 选择问题的减治法的算法
- 1、设置初始查找区间：i=1，j=n
- 2、以ri为轴值，对序列ri~rj进行一次划分，得到轴值的位置s
- 3、比较s和k的值
  - 3.1、若k=s，则返回ri，结束
  - 3.2、若k<s，则j=s-1，转步骤2
  - 3.3、若k>s，则i=s+1，转步骤2

例子：选择问题的减治法

- 选择问题的减治法的算法计算

- 输入：
  - A[...]: 9,5,2,6,11, 8
  - k: 3

选择问题算法分析

- 算法分析
- 最好情况下，每次选择的ri都是中值，则为O(n)（比较次数占主导）
- 最坏情况下，每次选择的ri都是最大值或最小值，则为O(n^2)
- 平均情况下，使用随机发生器，选择ri，则为O(n)

3. 排序问题中的减治法

- 插入排序
- 最好情况O(n)，最坏情况O(n^2)，平均情况O(n^2)
- 存在记录中元素的位置移动的现象

- 堆排序
- 存在动态维护堆的代价
- 锦标赛排序思想。时间复杂度O(nlogn)

3.1. 插入排序

- 方法
  - 选择任意未排序元素并将其插入已排序的子列表中的适当位置
  - 重复上述过程直到所有元素都被插入

插入排序例子

- Iteration i. 如果较小，则反复将元素i与左边的元素交换

- 性质. 在第i次迭代之后，a[0]至a[i]包含升序的前i + 1个元素

Array index	0	1	2	3	4	5	6	7	8	9
Value	2.78	7.42	0.56	1.12	1.17	0.32	6.21	4.42	3.14	7.71

Iteration 0: step 0.

*

插入排序例子

- Iteration i. 如果较小，则反复将元素i与左边的元素交换
- 性质. 在第i次迭代之后，a[0]至a[i]包含升序的前i + 1个元素

Array index	0	1	2	3	4	5	6	7	8	9
Value	2.78	7.42	0.56	1.12	1.17	0.32	6.21	4.42	3.14	7.71

Iteration 1: step 0.

*

插入排序例子

- Iteration i. 如果较小，则反复将元素i与左边的元素交换

- 性质. 在第i次迭代之后，a[0]至a[i]包含升序的前i + 1个元素

-

Array index	0	1	2	3	4	5	6	7	8	9
Value	2.78	0.56	7.42	1.12	1.17	0.32	6.21	4.42	3.14	7.71

Iteration 2: step 0.

*

插入排序例子

- Iteration i. 如果较小，则反复将元素i与左边的元素交换
- 性质. 在第i次迭代之后，a[0]至a[i]包含升序的前i + 1个元素

Array index	0	1	2	3	4	5	6	7	8	9
Value	0.56	2.78	7.42	1.12	1.17	0.32	6.21	4.42	3.14	7.71

Iteration 2: step 1.

*

插入排序例子

- Iteration i. 如果较小，则反复将元素i与左边的元素交换
- 性质. 在第i次迭代之后，a[0]至a[i]包含升序的前i + 1个元素

Array index	0	1	2	3	4	5	6	7	8	9
Value	0.56	2.78	7.42	1.12	1.17	0.32	6.21	4.42	3.14	7.71

Iteration 2: step 2.

*

插入排序例子

- Iteration i. 如果较小， 则反复将元素i与左边的元素交换

- 性质. 在第i次迭代之后， a[0]至a[i]包含升序的前i + 1个元素

Array index	0	1	2	3	4	5	6	7	8	9
Value	0.56	2.78	1.12	7.42	1.17	0.32	6.21	4.42	3.14	7.71

Iteration 3: step 0.

*

插入排序例子

- Iteration i. 如果较小，则反复将元素i与左边的元素交换

- 性质. 在第i次迭代之后，a[0]至a[i]包含升序的前i + 1个元素

Array index	0	1	2	3	4	5	6	7	8	9
Value	0.56	1.12	2.78	7.42	1.17	0.32	6.21	4.42	3.14	7.71

Iteration 3: step 1.

*

插入排序例子

- Iteration i. 如果较小，则反复将元素i与左边的元素交换

- 性质. 在第i次迭代之后，a[0]至a[i]包含升序的前i + 1个元素

Array index	0	1	2	3	4	5	6	7	8	9
Value	0.56	1.12	2.78	7.42	1.17	0.32	6.21	4.42	3.14	7.71

Iteration 3: step 2.

*

插入排序例子

- Iteration i. 如果较小，则反复将元素i与左边的元素交换
- 性质. 在第i次迭代之后，a[0]至a[i]包含升序的前i + 1个元素

Array index	0	1	2	3	4	5	6	7	8	9
Value	0.56	1.12	2.78	1.17	7.42	0.32	6.21	4.42	3.14	7.71

Iteration 4: step 0.

*

插入排序例子

- Iteration i. 如果较小，则反复将元素i与左边的元素交换

- 性质. 在第i次迭代之后，a[0]至a[i]包含升序的前i + 1个元素

Array index	0	1	2	3	4	5	6	7	8	9
Value	0.56	1.12	1.17	2.78	7.42	0.32	6.21	4.42	3.14	7.71

Iteration 4: step 1.

*

插入排序例子

- Iteration i. 如果较小，则反复将元素i与左边的元素交换

- 性质. 在第i次迭代之后，a[0]至a[i]包含升序的前i + 1个元素

Array index	0	1	2	3	4	5	6	7	8	9
Value	0.56	1.12	1.17	2.78	7.42	0.32	6.21	4.42	3.14	7.71

Iteration 4: step 2.

*

插入排序例子

- Iteration i. 如果较小，则反复将元素i与左边的元素交换

- 性质. 在第i次迭代之后，a[0]至a[i]包含升序的前i + 1个元素

Array index	0	1	2	3	4	5	6	7	8	9
Value	0.56	1.12	1.17	2.78	0.32	7.42	6.21	4.42	3.14	7.71

Iteration 5: step 0.

*

插入排序例子

- Iteration i. 如果较小，则反复将元素i与左边的元素交换

- 性质. 在第i次迭代之后，a[0]至a[i]包含升序的前i + 1个元素

Array index	0	1	2	3	4	5	6	7	8	9
Value	0.56	1.12	1.17	0.32	2.78	7.42	6.21	4.42	3.14	7.71

Iteration 5: step 1.

*

插入排序例子

- Iteration i. 如果较小，则反复将元素i与左边的元素交换

- 性质. 在第i次迭代之后，a[0]至a[i]包含升序的前i + 1个元素

Array index	0	1	2	3	4	5	6	7	8	9
Value	0.56	1.12	0.32	1.17	2.78	7.42	6.21	4.42	3.14	7.71

Iteration 5: step 2.

*

插入排序例子

- Iteration i. 如果较小，则反复将元素i与左边的元素交换

- 性质. 在第i次迭代之后，a[0]至a[i]包含升序的前i + 1个元素

Array index	0	1	2	3	4	5	6	7	8	9
Value	0.56	0.32	1.12	1.17	2.78	7.42	6.21	4.42	3.14	7.71

Iteration 5: step 3.

*

插入排序例子

- Iteration i. 如果较小，则反复将元素i与左边的元素交换

- 性质. 在第i次迭代之后，a[0]至a[i]包含升序的前i + 1个元素

Array index	0	1	2	3	4	5	6	7	8	9
Value	0.32	0.56	1.12	1.17	2.78	7.42	6.21	4.42	3.14	7.71

Iteration 5: step 4.

*

插入排序例子

- Iteration i. 如果较小，则反复将元素i与左边的元素交换

- 性质. 在第i次迭代之后，a[0]至a[i]包含升序的前i + 1个元素

Array index	0	1	2	3	4	5	6	7	8	9
Value	0.32	0.56	1.12	1.17	2.78	7.42	6.21	4.42	3.14	7.71

Iteration 5: step 5.

*

插入排序例子

- Iteration i. 如果较小，则反复将元素i与左边的元素交换

- 性质. 在第i次迭代之后，a[0]至a[i]包含升序的前i + 1个元素

Array index	0	1	2	3	4	5	6	7	8	9
Value	0.32	0.56	1.12	1.17	2.78	6.21	7.42	4.42	3.14	7.71

Iteration 6: step 0.

*

插入排序例子

- Iteration i. 如果较小，则反复将元素i与左边的元素交换
- 性质. 在第i次迭代之后，a[0]至a[i]包含升序的前i + 1个元素

Array index	0	1	2	3	4	5	6	7	8	9
Value	0.32	0.56	1.12	1.17	2.78	6.21	7.42	4.42	3.14	7.71

Iteration 6: step 1.

*

插入排序例子

- Iteration i. 如果较小，则反复将元素i与左边的元素交换

- 性质. 在第i次迭代之后，a[0]至a[i]包含升序的前i + 1个元素

Array index	0	1	2	3	4	5	6	7	8	9
Value	0.32	0.56	1.12	1.17	2.78	6.21	4.42	7.42	3.14	7.71

Iteration 7: step 0.

*

插入排序例子

- Iteration i. 如果较小, 则反复将元素i与左边的元素交换

- 性质. 在第i次迭代之后, a[0]至a[i]包含升序的前i + 1个元素

Array index	0	1	2	3	4	5	6	7	8	9
Value	0.32	0.56	1.12	1.17	2.78	4.42	6.21	7.42	3.14	7.71

Iteration 7: step 1.

*

插入排序例子

- Iteration i. 如果较小，则反复将元素i与左边的元素交换

- 性质. 在第i次迭代之后，a[0]至a[i]包含升序的前i + 1个元素

Array index	0	1	2	3	4	5	6	7	8	9
Value	0.32	0.56	1.12	1.17	2.78	4.42	6.21	7.42	3.14	7.71

Iteration 7: step 2.

*

插入排序例子

- Iteration i. 如果较小，则反复将元素i与左边的元素交换

- 性质. 在第i次迭代之后，a[0]至a[i]包含升序的前i + 1个元素

Array index	0	1	2	3	4	5	6	7	8	9
Value	0.32	0.56	1.12	1.17	2.78	4.42	6.21	3.14	7.42	7.71

Iteration 8: step 0.

*

插入排序例子

- Iteration i．如果较小，则反复将元素i与左边的元素交换
- 性质．在第i次迭代之后，a[0]至a[i]包含升序的前i + 1个元素

Array index	0	1	2	3	4	5	6	7	8	9
Value	0.32	0.56	1.12	1.17	2.78	4.42	3.14	6.21	7.42	7.71

Iteration 8: step 1.

*

插入排序例子

- Iteration i. 如果较小，则反复将元素i与左边的元素交换
- 性质. 在第i次迭代之后，a[0]至a[i]包含升序的前i + 1个元素

Array index	0	1	2	3	4	5	6	7	8	9
Value	0.32	0.56	1.12	1.17	2.78	3.14	4.42	6.21	7.42	7.71

Iteration 8: step 2.

*

插入排序例子

- Iteration i. 如果较小，则反复将元素i与左边的元素交换

- 性质. 在第i次迭代之后，a[0]至a[i]包含升序的前i + 1个元素

Array index	0	1	2	3	4	5	6	7	8	9
Value	0.32	0.56	1.12	1.17	2.78	3.14	4.42	6.21	7.42	7.71

Iteration 8: step 3.

*

插入排序例子

- Iteration i. 如果较小，则反复将元素i与左边的元素交换

- 性质. 在第i次迭代之后，a[0]至a[i]包含升序的前i + 1个元素

Array index	0	1	2	3	4	5	6	7	8	9
Value	0.32	0.56	1.12	1.17	2.78	3.14	4.42	6.21	7.42	7.71

Iteration 9: step 0.

*

插入排序例子

- Iteration i. 如果较小，则反复将元素i与左边的元素交换

- 性质. 在第i次迭代之后，a[0]至a[i]包含升序的前i + 1个元素

Array index	0	1	2	3	4	5	6	7	8	9
Value	0.32	0.56	1.12	1.17	2.78	3.14	4.42	6.21	7.42	7.71

Iteration 10: DONE.

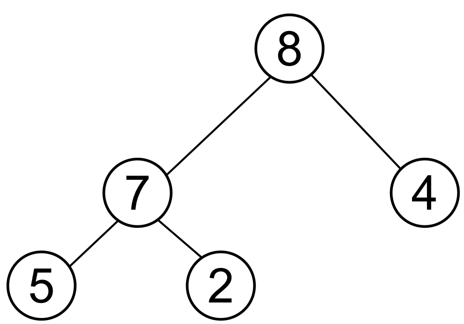
插入排序算法

算法: insertion-sort(*A*)

```
for j ← 2 to n
    do key ← A[ j ]
        // 把A[ j ]插入到已排序数组A[1 . . j -1]
        i ← j - 1
        while i > 0 and A[i] > key
            do A[i + 1] ← A[i]
                i ← i - 1
        A[i + 1] ← key
```

3.2 堆排序

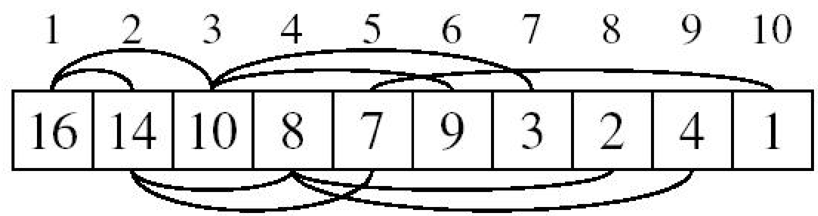
- 给定一个组数，构造数据结构堆来进行排序
- 定义: 堆是完全二叉树，有以下两个属性:
  - **结构属性:** 除了最后一个层级外（从左到右填充），所有层级都已满，
  - **序属性:** 对于任何节点x，parent(x)≥x

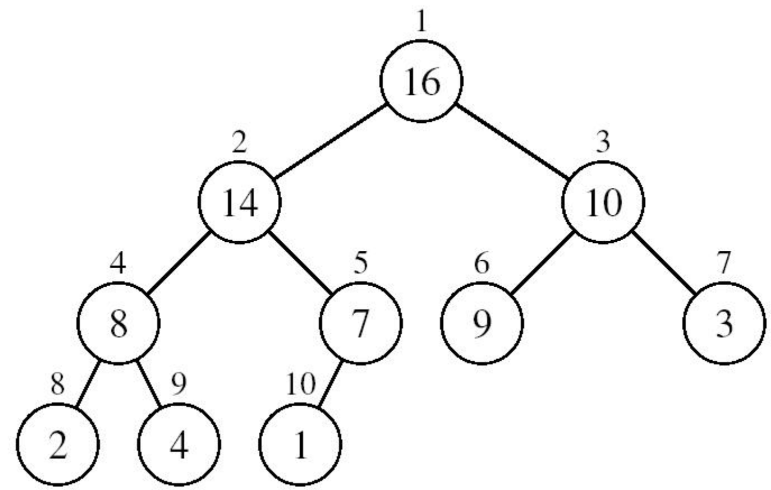


Heap（堆）

堆的数组表示

- 堆可以存储为数组A
  - 树的根是A[1]
  - A[i]的左孩子= A[2i]
  - A[i]的右孩子= A[2i + 1]
  - A[i]的父节点 = A[$\lfloor i/2 \rfloor$]
- 最大堆，最小堆

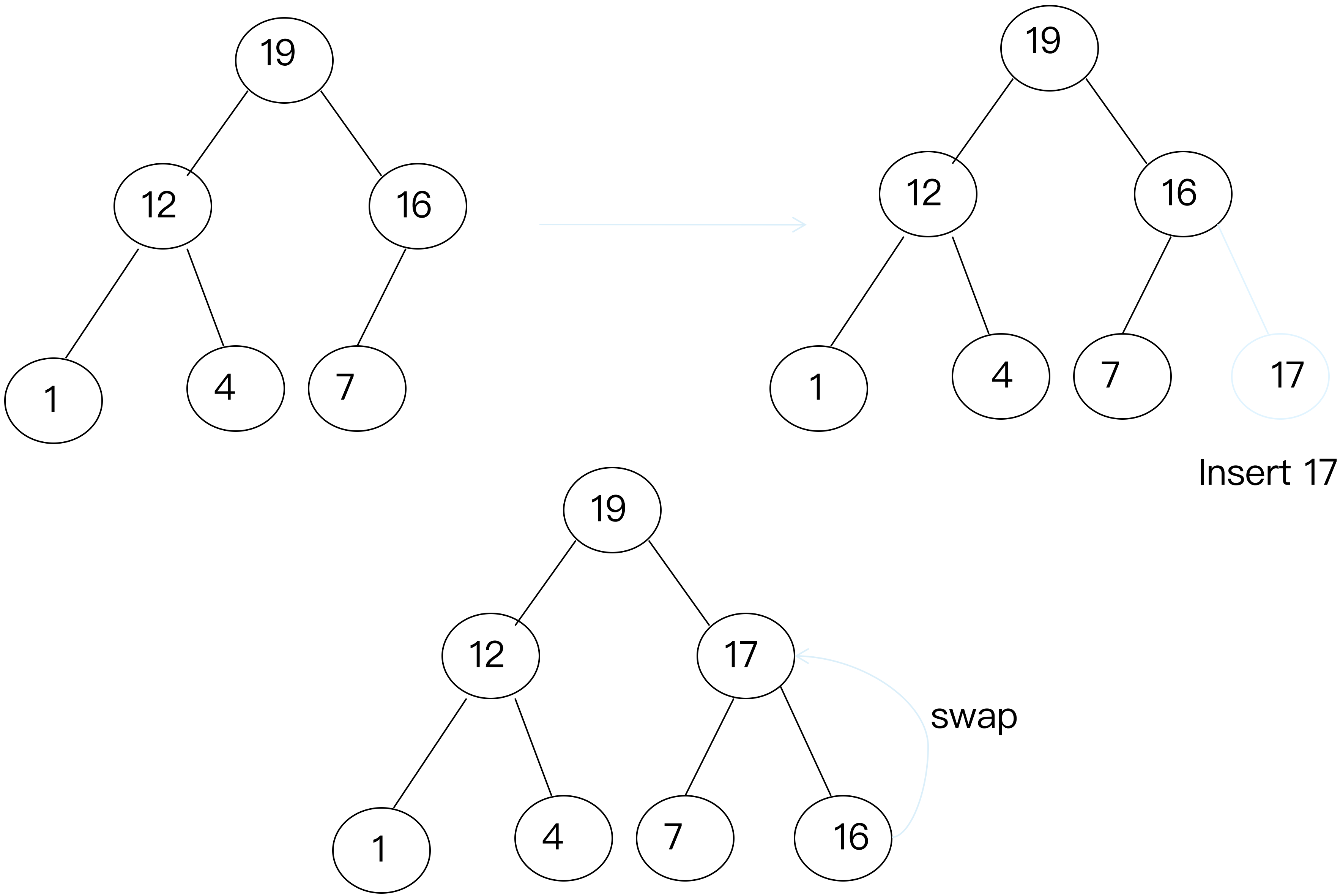




堆的插入操作

- 将新元素添加到最低层的下一个可用位置

- 如果违反了最大堆的属性，请还原：
  - 常规策略是渗透（或冒泡）：如果该元素的父节点元素小于该元素，则交换父元素和子元素



冒泡使得最大堆性质保持

堆的删除操作

- 删除最大元
  - 将最后一个数字复制到根（覆盖存储在那里的最大元素）。
  - 通过向下渗透来恢复最大堆属性。

关于堆排序的程序有：

- 筛选法Heapify
- 堆的生成Build Heap
- 堆排序Heap Sort

Heapify

- Heapify选择最大的子键并将其与父键进行比较。如果父键大于最大的子键，则退出；否则将其与最大的子键交换。使得父级比其子级更大。

```
Heapify(A, i)
{
    l ← left(i)
    R ← right(i)
    if l在堆的范围并且A[l] > A[i]
        then largest ←l
        else largest ← i
    if R在堆的范围并且 A[r] > A[largest]
        then largest ← R
    if largest != i
        then交换 A[i] ←→ A[largest]
            Heapify(A, largest)
}
```

堆的生成

- 我们可以用自下而上的方式使用过程'Heapify'来转换数组A[1...n]变成堆（因为子组数A中的元素[n/2+1...n] 都是叶子）

```
 Buildheap(A)
{
      for i 从 length[A]/2 下降到1
         do Heapify(A, i)
 }
```

堆排序算法

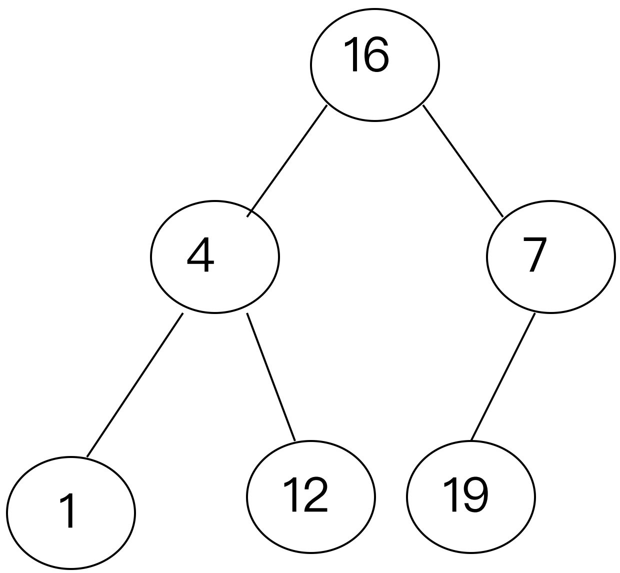
- 堆排序算法首先使用BUILD-heap过程在输入数组A[1...n] 中构造堆。由于数组的最大元素存储在根节点A[1]上，因此可以通过与A[n]（A中的最后一个元素）交换，并通过**Heapify**将其放入其正确的最终位置。

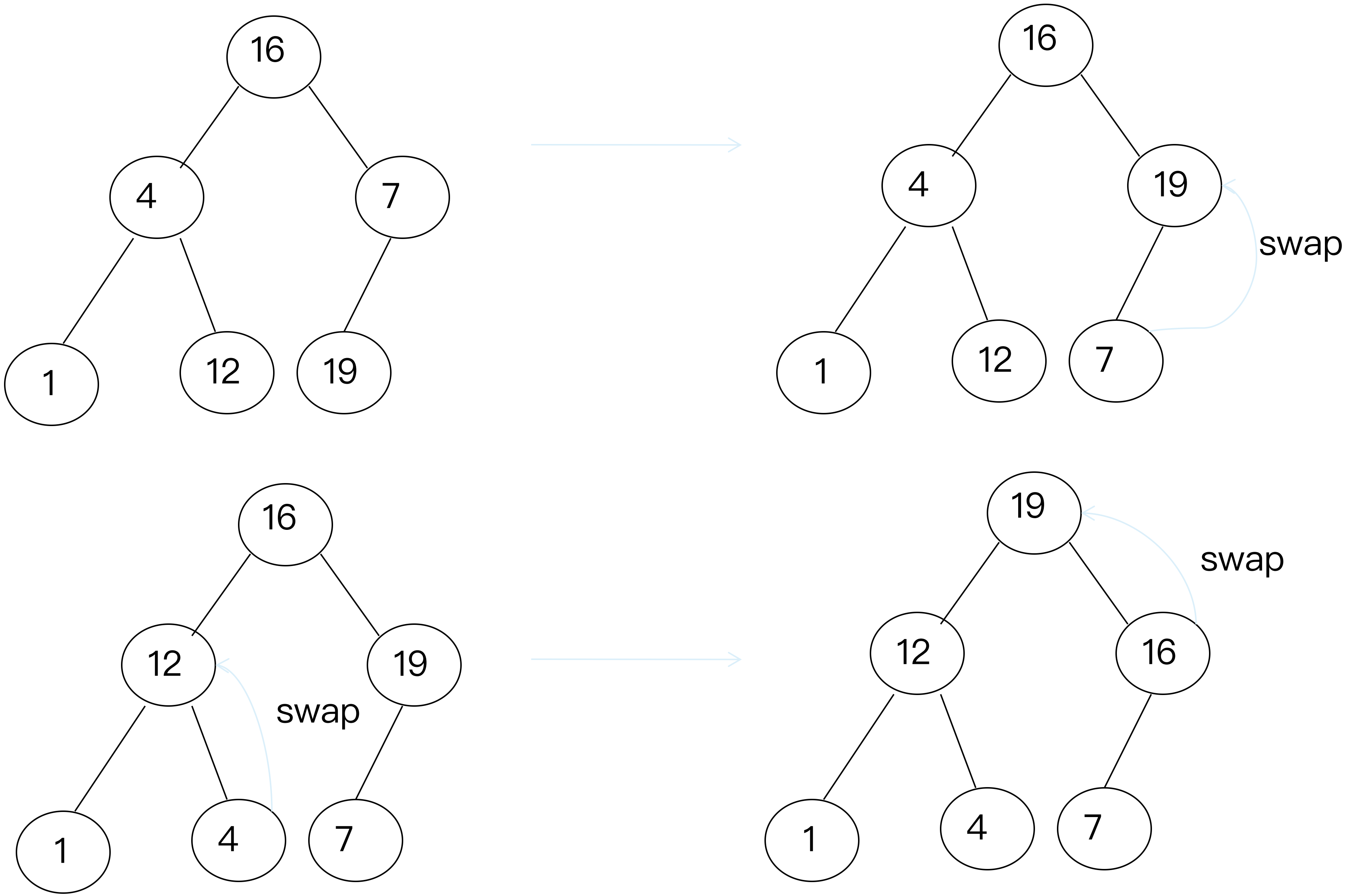
```
Heapsort(A)
{
      Buildheap(A)
      for i 从 length[A] 下降为 2
         交换 A[1] <-> A[i]
         heapsize[A] <- heapsize[A] - 1
         Heapify(A, 1)
}
```

例子: 转化如下数组为堆

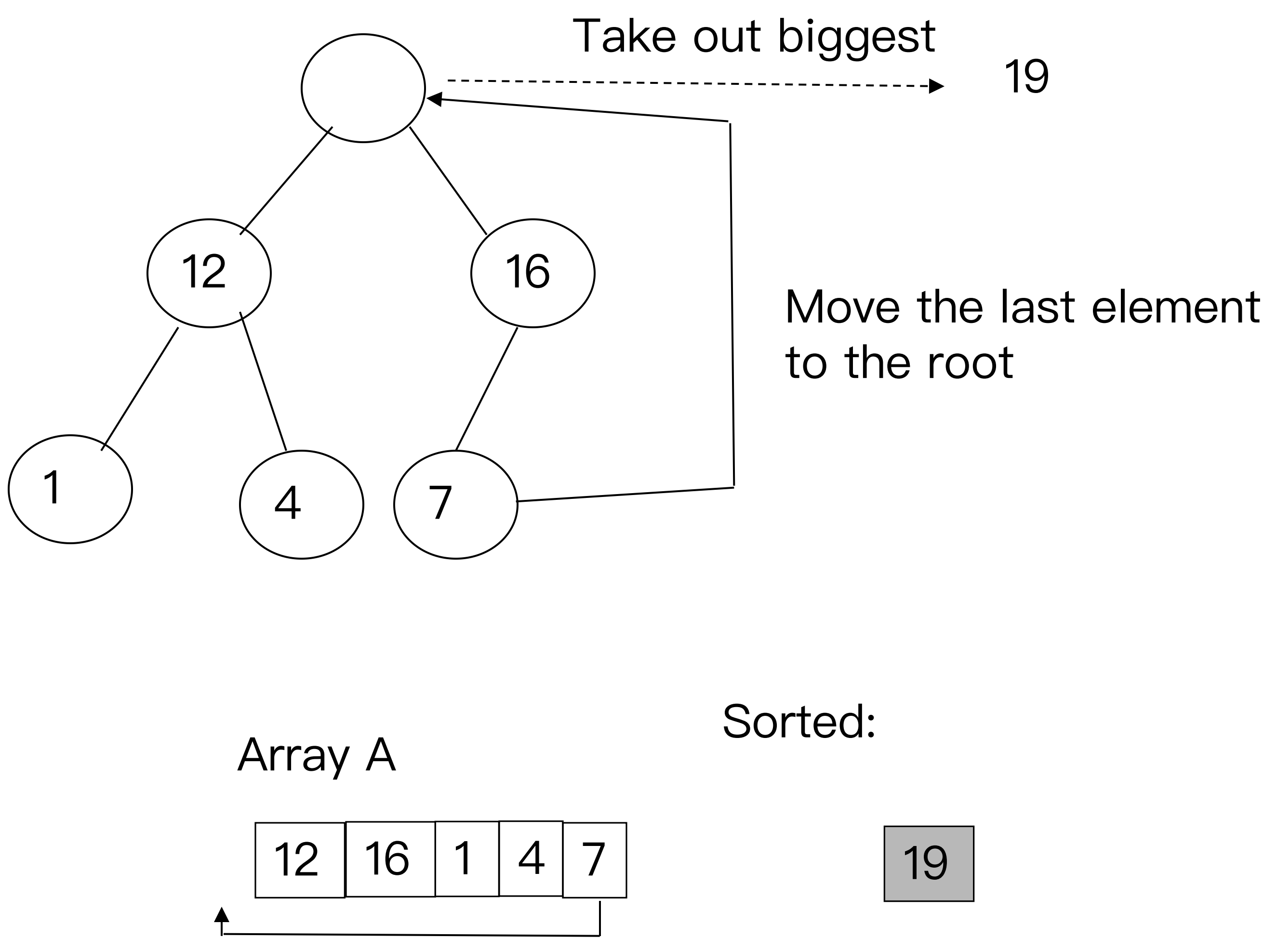
16	4	7	1	12	19

它是如下堆完全二叉树:

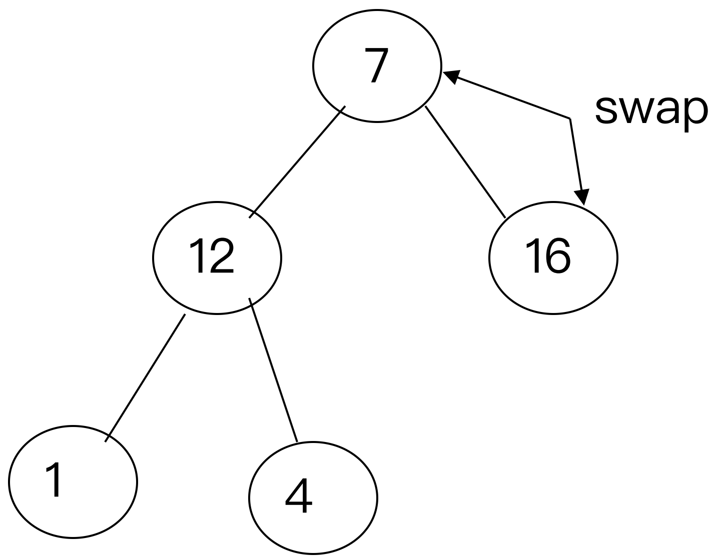



16
4
7
1
12
19
16
4
19
swap
1
12
7
16
12
19
swap
1
4
7
19
swap
12
16
1
4
7

例子：堆排序


Take out biggest
19
12
16
Move the last element
to the root
1
4
7
Sorted:
Array A
12
16
1
4
7
19

HEAPIFY()

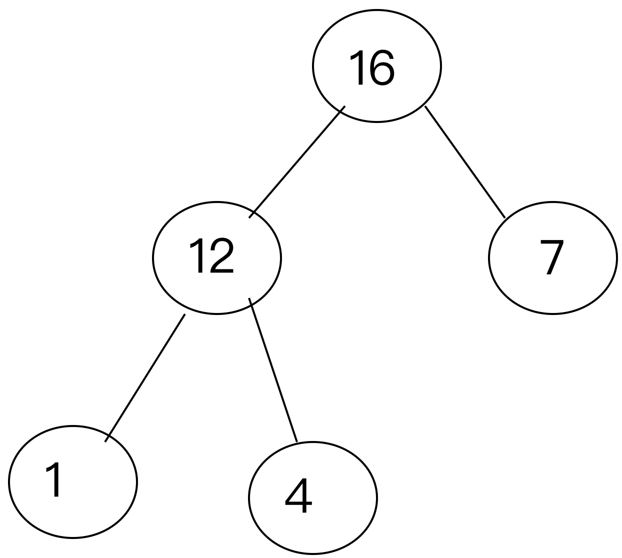


Array A

7	12	16	1	4

Sorted:

19

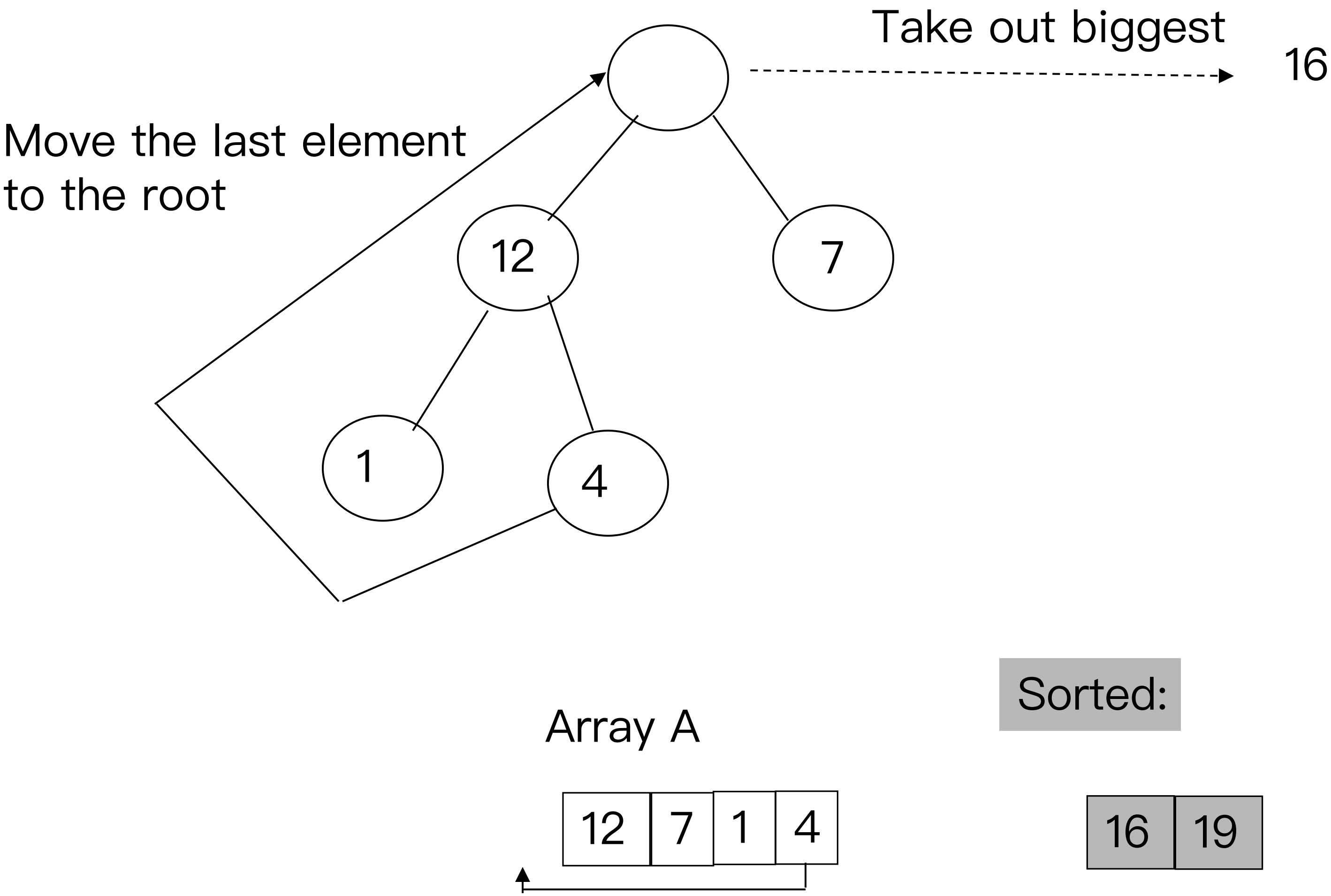


Array A

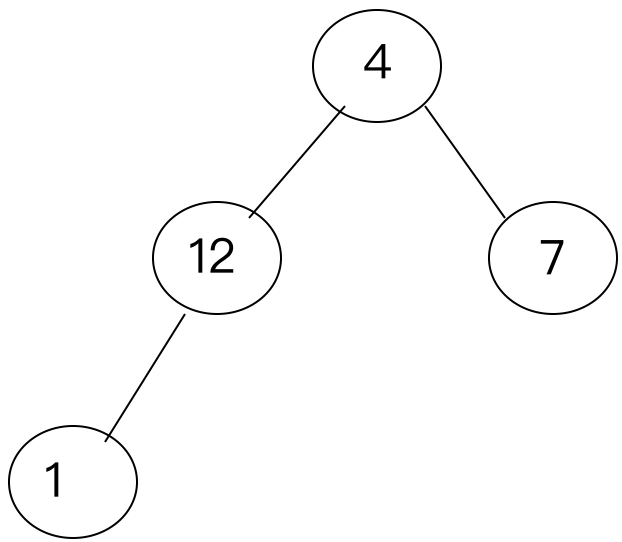
16	12	7	1	4

Sorted:

19



Take out biggest
16
Move the last element
to the root
12
7
1
4
Sorted:
Array A
12
7
1
4
16
19



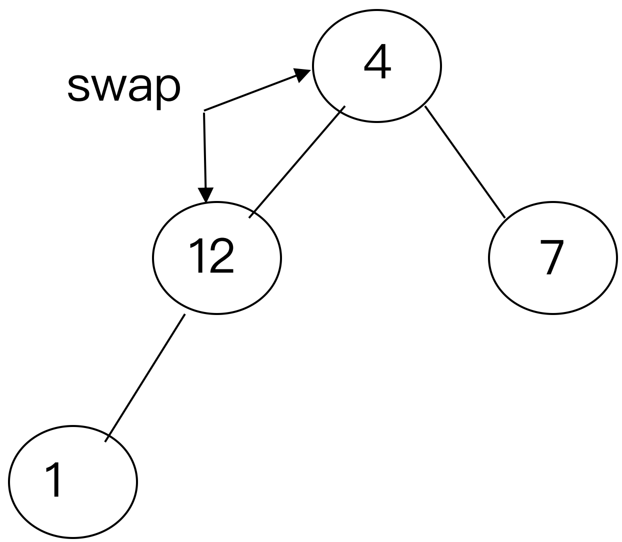
Sorted:

Array A

4	12	7	1

16	19

HEAPIFY()

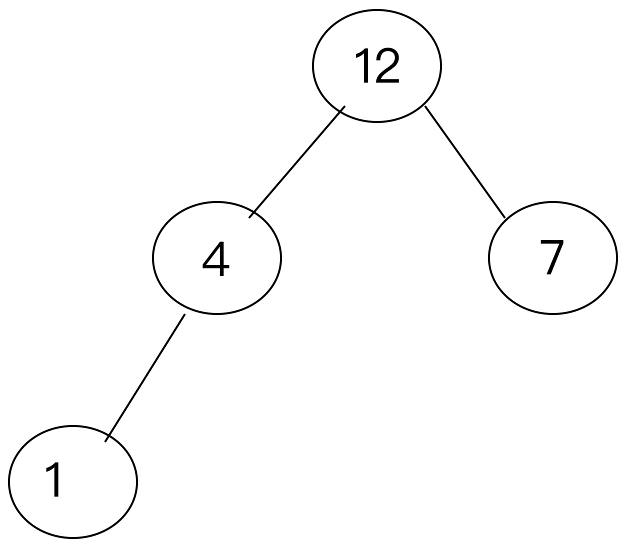


Array A

4	12	7	1

Sorted:

16	19

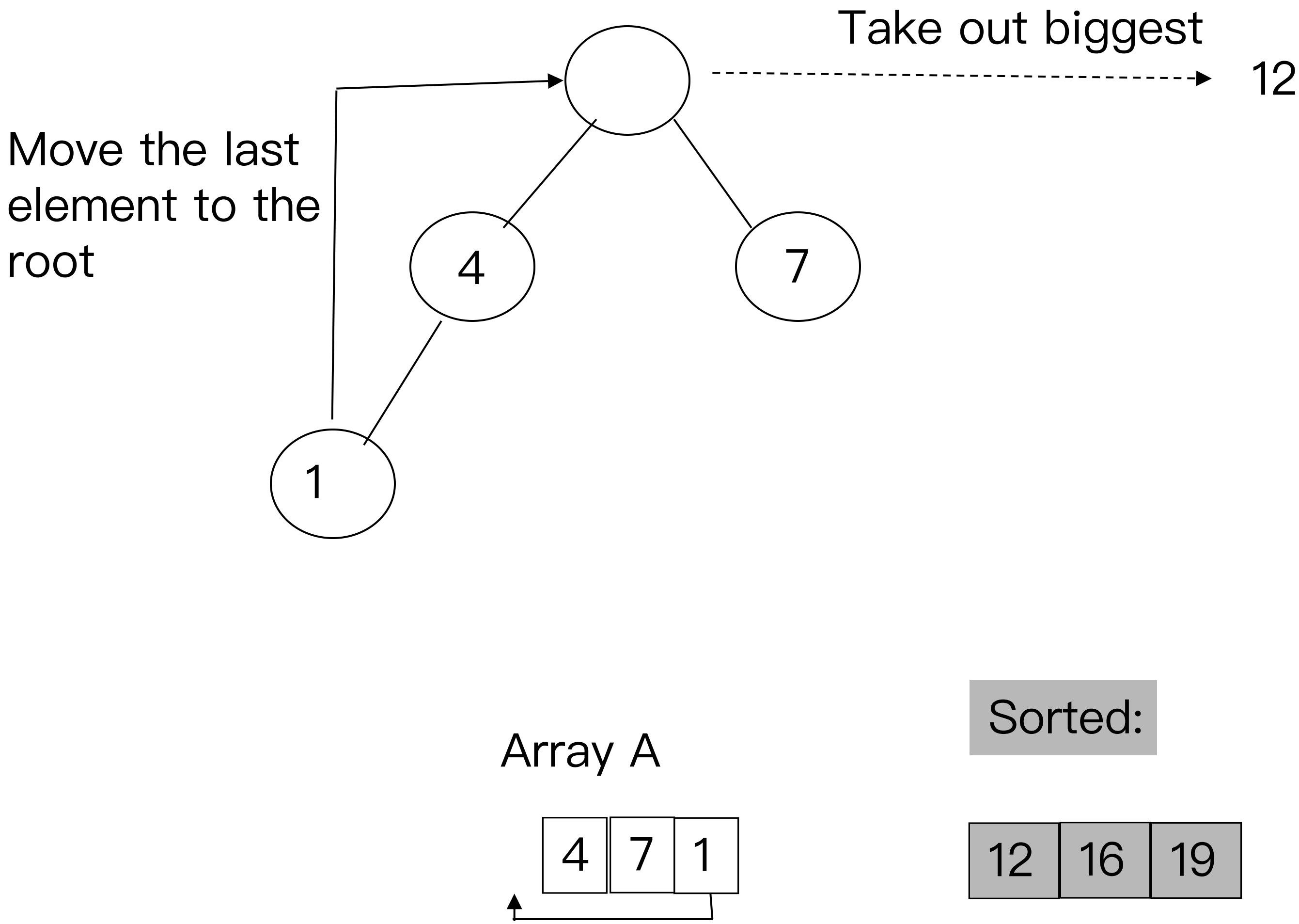


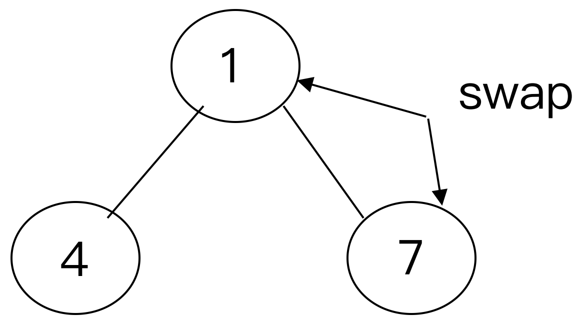
Array A

12	4	7	1

Sorted:

16	19


Take out biggest
12
Move the last element to the root
4
7
1
Array A
4 7 1
Sorted:
12 16 19

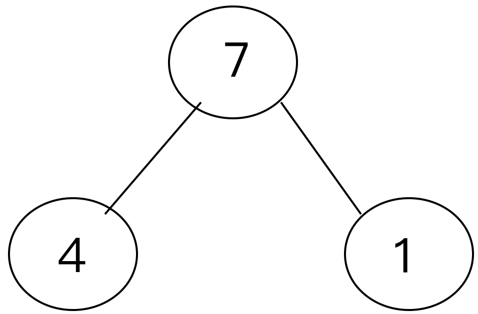


Array A

1	4	7

Sorted:

12	16	19

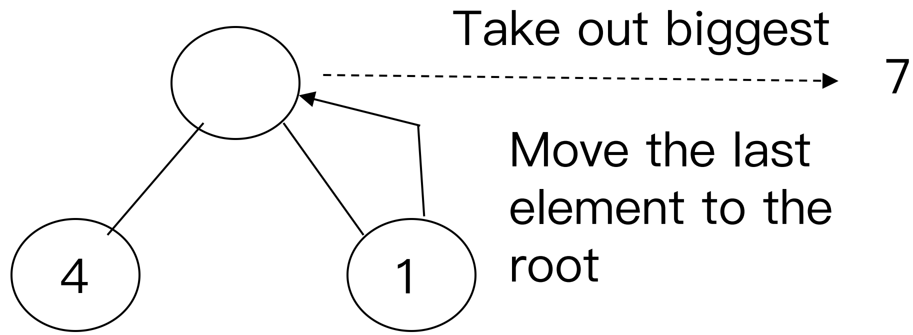


Array A

7	4	1

Sorted:

12	16	19



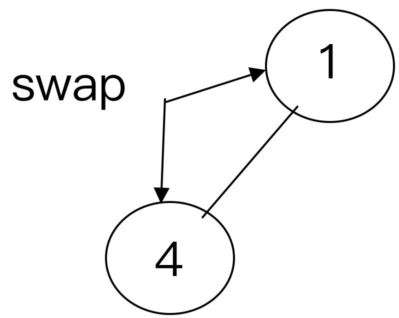
Array A

1	4

Sorted:

7	12	16	19

HEAPIFY()

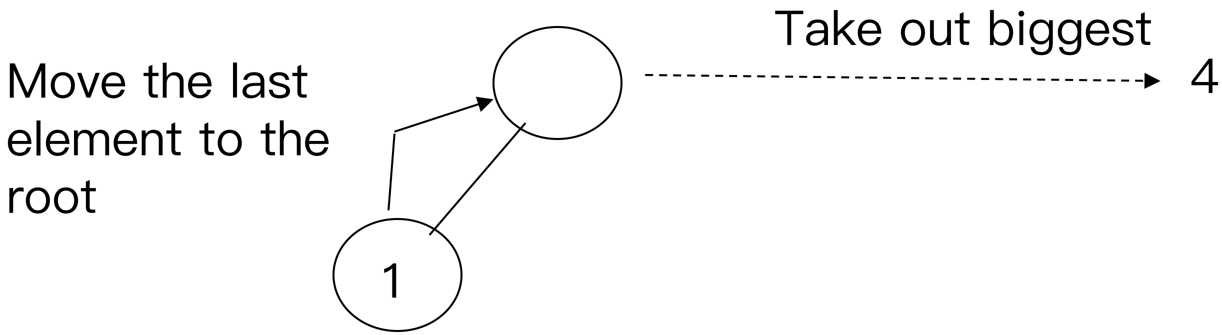


Array A

4	1

Sorted:

7	12	16	19

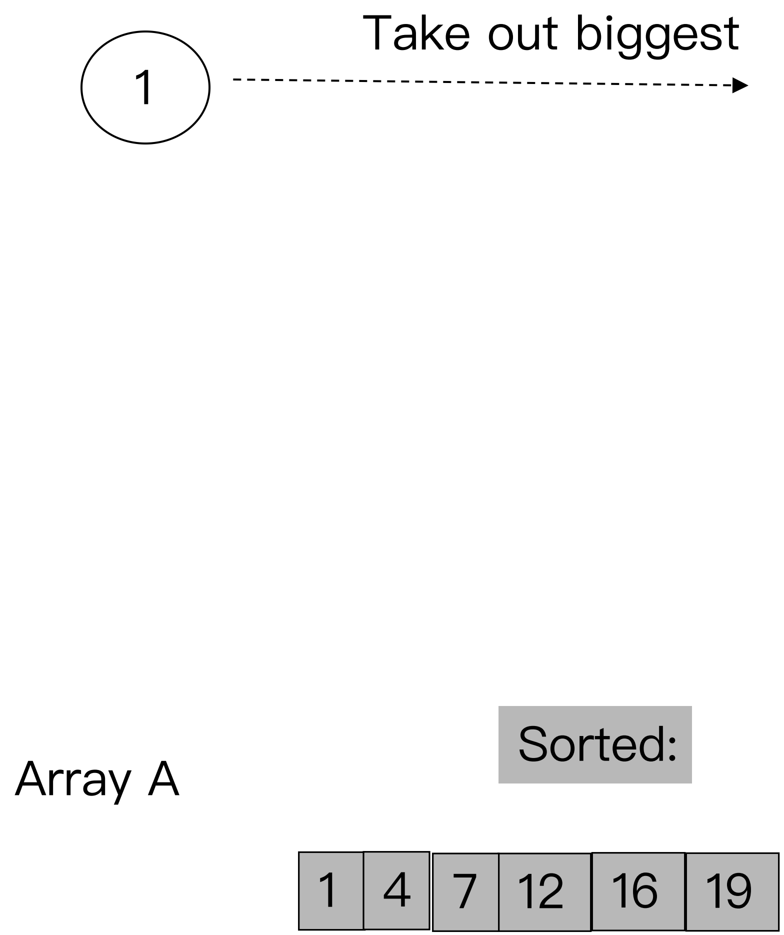


Array A

1

Sorted:

4	7	12	16	19


Take out biggest
1
Sorted:
Array A
1
4
7
12
16
19

Sorted:

1	4	7	12	16	19

4. 组合问题中的减治法

- 淘汰赛冠军问题
- 假币问题

4.1淘汰赛冠军问题

- 设有n=2^k个选手进行淘汰赛，最后决出冠军。问如何设计淘汰比赛的过程。
- 分治法的思想
- n个选手分成2组。每组决出胜者后，这两个胜者再角逐。
  - 每组又分成2个小组，各自决出胜者后，再角逐。
- O(n)

淘汰赛冠军问题

- 减治法的思想
- n个选手分成n/2组，每组2名选手。
- 每组胜者，构成待比赛的全部选手。再将这些选手分成n/2/2=n/4组，每组2名选手。
- 重复上述过程，直至只有2名选手为止。他们两人最后决出冠军。
- O(n)

淘汰赛冠军算法

- 减治法的算法
- 1、i=n；//变量i记录当前的全部选手个数
- 2、当i>1时，循环以下操作
  - 2.1、i=i/2；//折半分组
  - 2.2、for j=0 to j<i；j++
    - 2.2.1、如果r[j+i]>r[j]，则r[j]=r[j+i]
- 3、返回r[0]

4.2 假币问题

- 在n枚外观相同的硬币中，有一枚是假币，且较轻。只能通过天平来比较两组硬币。
- 问；如何找出这枚假币。

假币问题

- 减治法的思想
- 硬币分两组，每组有$\lfloor n/2 \rfloor$个硬币。若n为奇数，则留一枚在外面。
- 判断两组是否一样重。若是，则预留者是假币。若不是，则假币在较轻的那组中
- 对较轻的那组重复上述过程，直至找到假币。
- $O(\log_2 n)$

假币问题

- 减治法的思想

- 若将硬币分成三组，则$O(\log_3 n)$

假币问题

- 分三组的减治法的算法
- 输入：硬币所在组的下标low和high，硬币个数n
- 1、如果n=1，则该硬币为假币，结束；
- 2、计算3组的硬币个数num1、num2、num3
- 3、add1=第1组硬币重量，add2=第2组硬币重量

假币问题

- 分三组的减治法的算法
- 输入：硬币所在组的下标low和high，硬币个数n
- 4、根据下列情况，分别执行一种操作：
  - 4.1、若add1<add2，则在第1组硬币中继续查找
  - 4.2、若add1>add2，则在第2组硬币中继续查找
  - 4.3、若add1=add2，则在第3组硬币中继续查找

练习1

- 设计算法实现在大根堆中删除一个元素，要求算法堆时间复杂度为O(log2 n)

- 拿子游戏。考虑下面这个游戏：桌子上有一堆火柴，游戏开始时共有n根火柴，两个玩家轮流拿走1根、2根或3根，拿走最后一根的玩家为获胜方。请为先走的玩家设计一个制胜的策略（如果存在的话)

- P94-95：
- T3，T10
- T3：修改折半查找算法，使之能够进行范围查找。即找出给定值a和b之间的所有元素。
- T10：在120枚外观相同的硬币中，有一枚假币。但不是它是轻的还是重的。使用一个天平来任意比较两组硬币。最坏情况下，能否只比较5次，就检测出这枚假币？

End