

# 极客时间算法训练营

## 第六课

### 二叉堆、二叉搜索树

李煜东

《算法竞赛进阶指南》作者





# 目录

1. 二叉堆的原理、实现与应用
2. 二叉搜索树的原理、实现与应用

# 二叉堆

# 堆 Heap

堆（Heap）是一种高效维护集合中最大或最小元素的数据结构。

大根堆：根节点最大的堆，用于维护和查询max。

小根堆：根节点最小的堆，用于维护和查询min。

堆是一棵树，并且满足堆性质（Heap property）

- 大根堆任意结点的关键码  $\geq$  它所有子结点的关键码（父  $\geq$  子）
- 小根堆任意结点的关键码  $\leq$  它所有子结点的关键码（父  $\leq$  子）

# 二叉堆 Binary Heap

二叉堆是堆的一种简易实现

- 本质上是一棵满足堆性质的完全二叉树

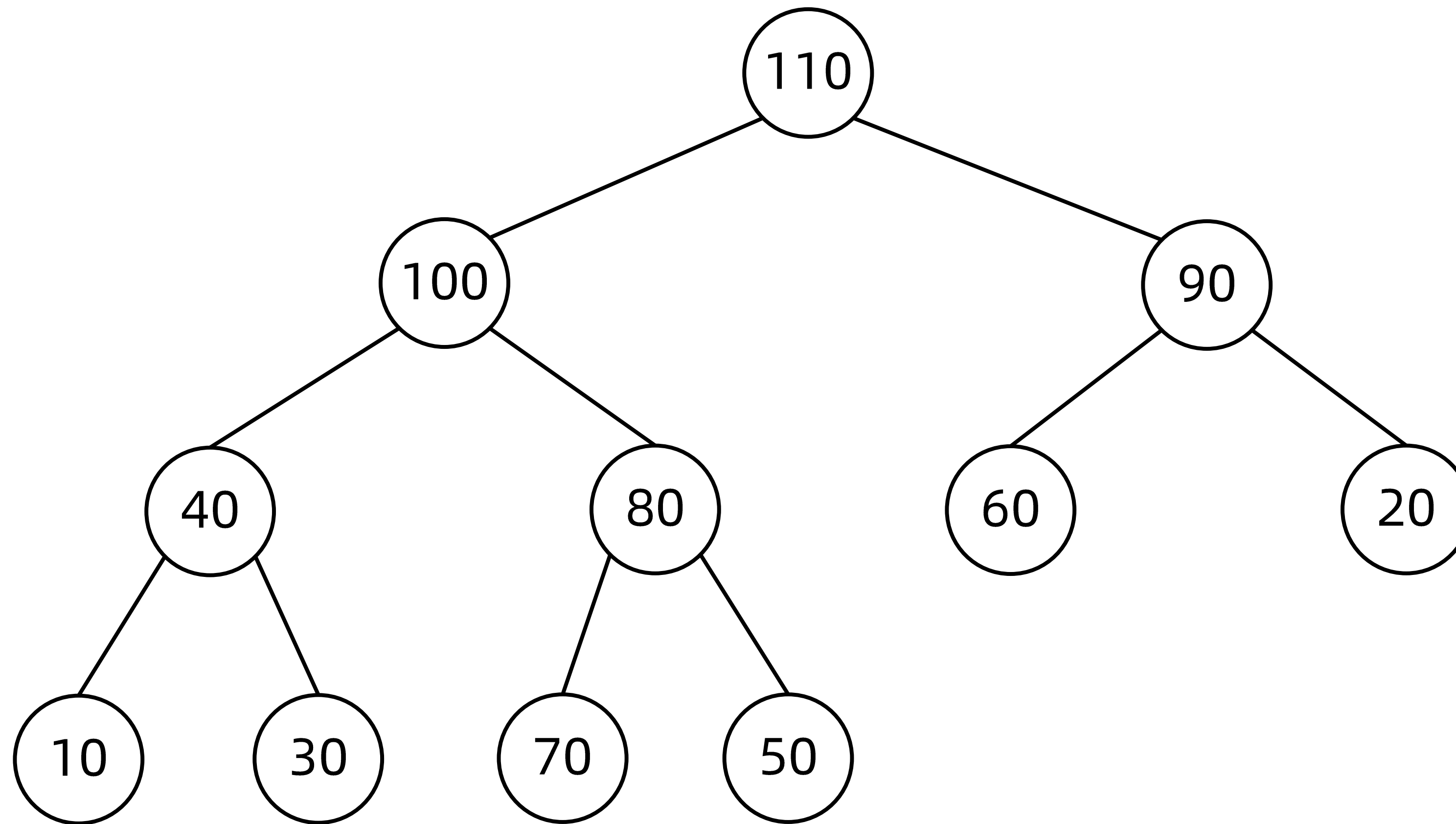
常见操作

- 建堆 (build) :  $O(N)$
- 查询最值 (get max/min) :  $O(1)$
- 插入 (insert) :  $O(\log N)$
- 取出最值 (delete max/min) :  $O(\log N)$

斐波那契堆、配对堆等可以做到插入  $O(1)$ ，左偏树、斜堆等可以支持合并

这些高级数据结构就不再讲解了

# 二叉堆



# 二叉堆的实现

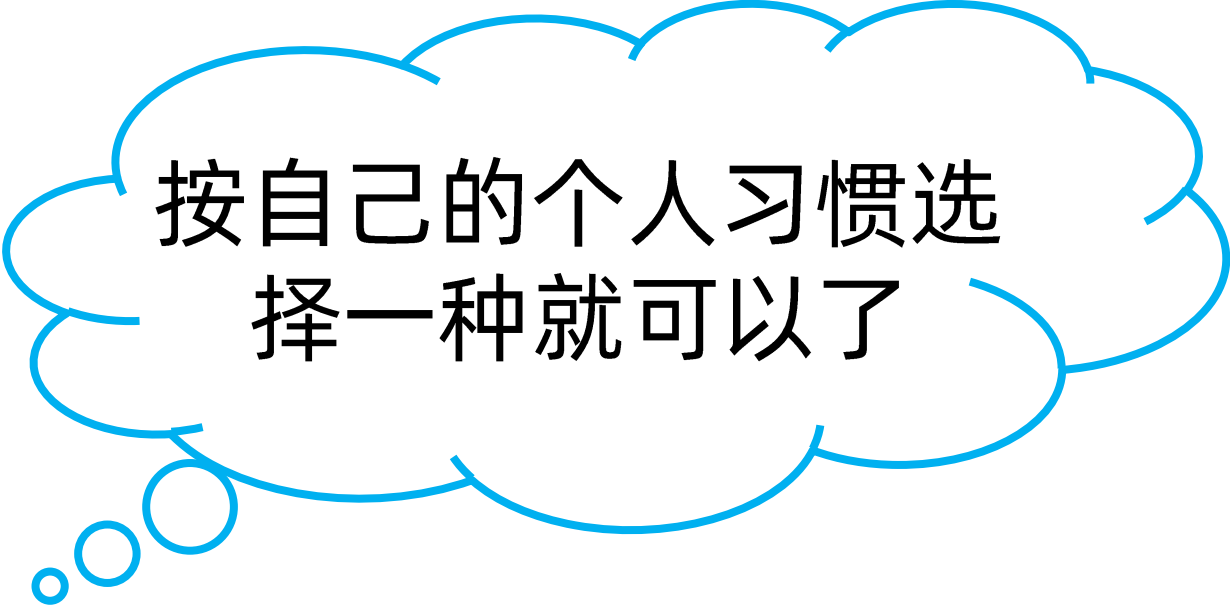
二叉堆一般使用一个一维数组来存储，利用完全二叉树的结点编号特性

假设第一个元素存储在索引（下标）为 1 的位置的话

- 索引为  $p$  的结点的左孩子的索引为  $p * 2$
- 索引为  $p$  的结点的右孩子的索引为  $p * 2 + 1$
- 索引为  $p$  的结点的父亲的索引为  $p / 2$ （下取整）

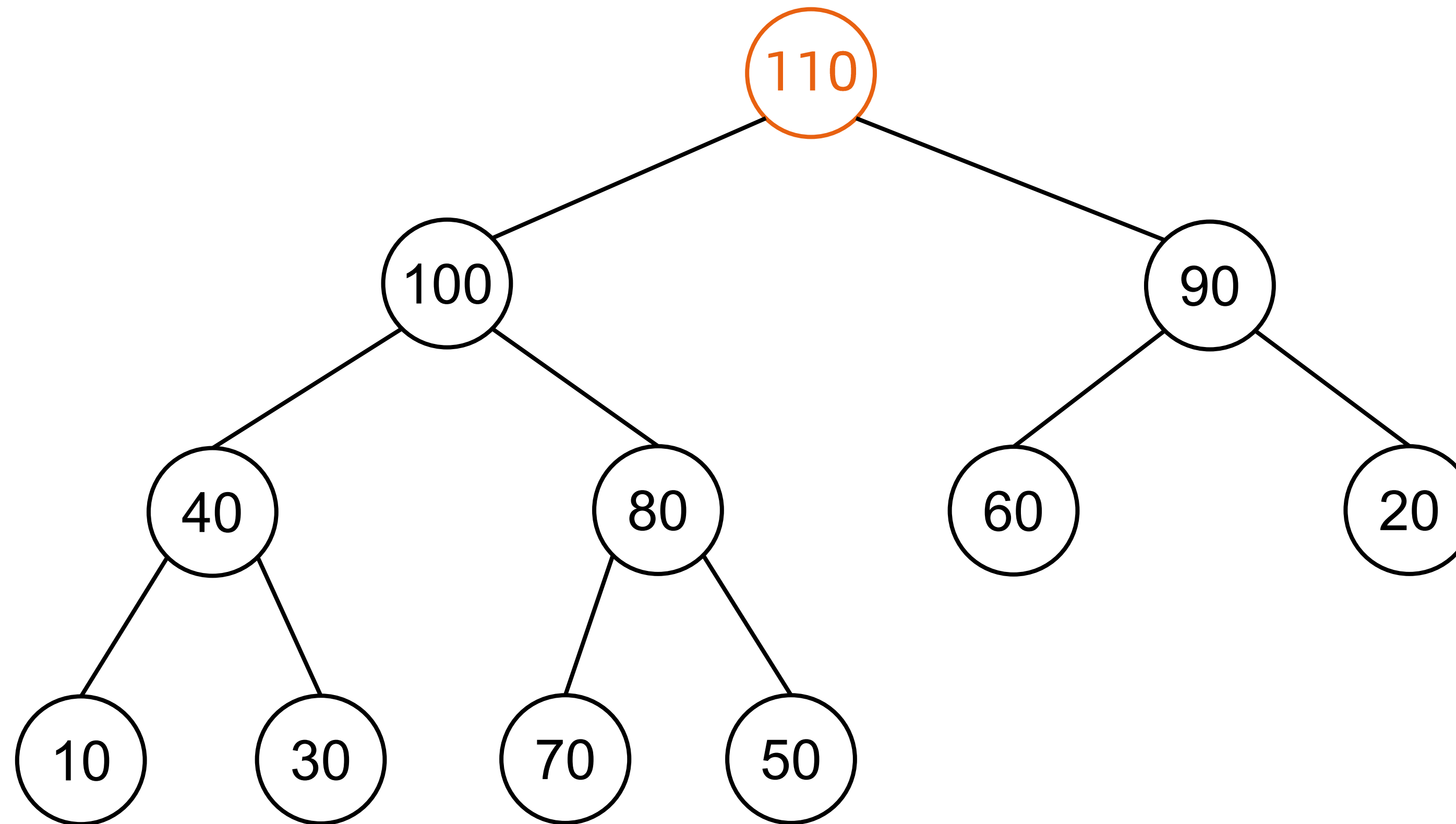
假设第一个元素存储在索引（下标）为 0 的位置的话

- 索引为  $p$  的结点的左孩子的索引为  $p * 2 + 1$
- 索引为  $p$  的结点的右孩子的索引为  $p * 2 + 2$
- 索引为  $p$  的结点的父亲的索引为  $(p - 1) / 2$ （下取整）



按自己的个人习惯选择一种就可以了

# 二叉堆



一维数组 [0, 110, 100, 90, 40, 80, 60, 20, 10, 30, 70, 50]



# 插入 (insert)

新元素一律插入到数组 heap 的尾部

- 设插入到了索引  $p$  的位置

然后向上进行一次调整 (Heapify Up)

- 若已到达根, 停止
- 若满足堆性质 ( $\text{heap}[p] \leq \text{heap}[p / 2]$ ), 停止
- 否则交换  $\text{heap}[p]$  和  $\text{heap}[p / 2]$ , 令  $p = p / 2$ , 继续调整

$O(\log N)$

# 取出堆顶 (extract / delete max)

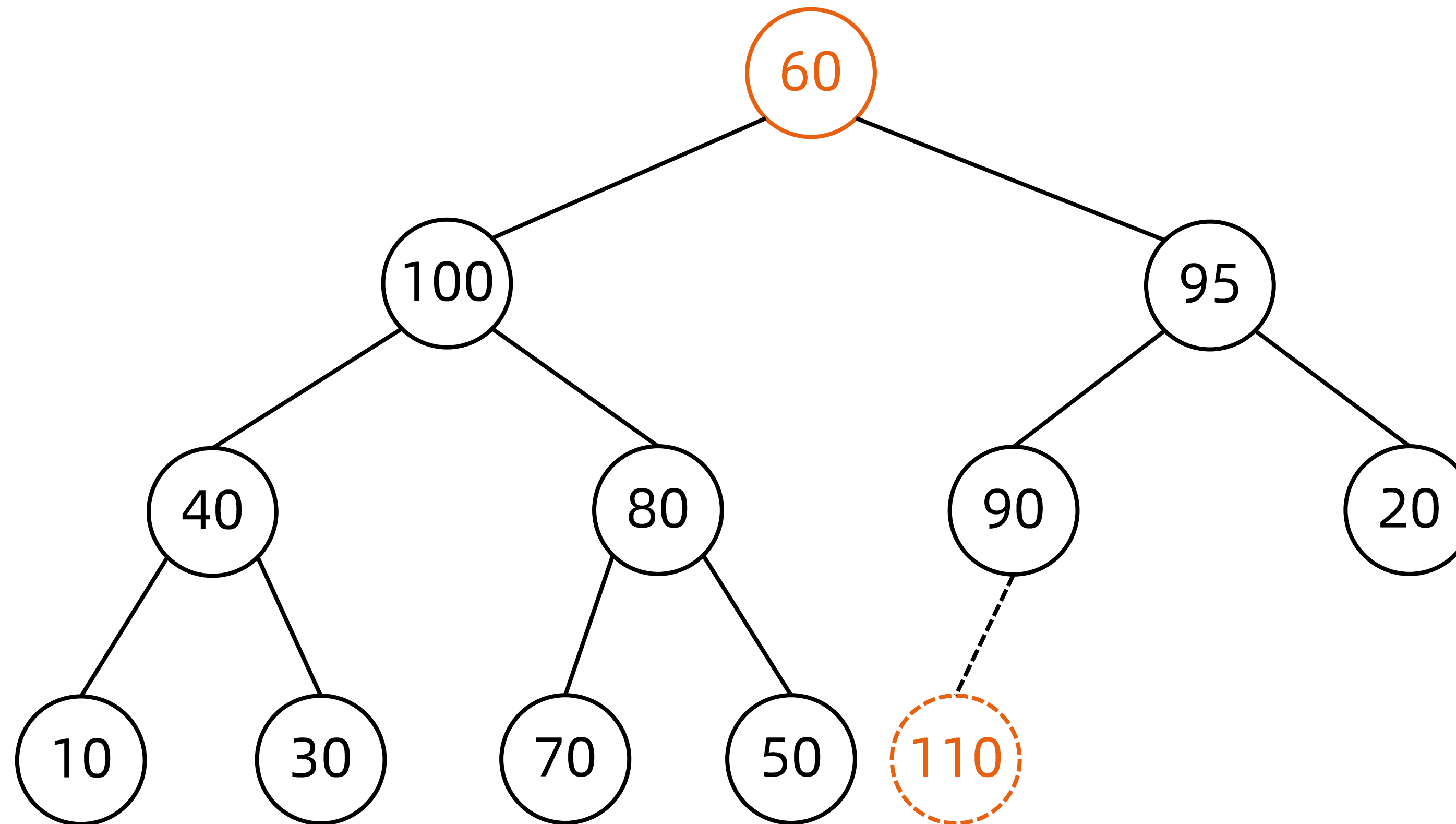
把堆顶 (heap[1]) 与堆尾 (heap[n]) 交换, 删除堆尾 (数组最后一个元素)

然后从根向下进行一次调整 (Heapify Down)

- 每次与左、右子结点中较大的一个比较, 检查堆性质, 不满足则交换
- 注意判断子结点是否存在

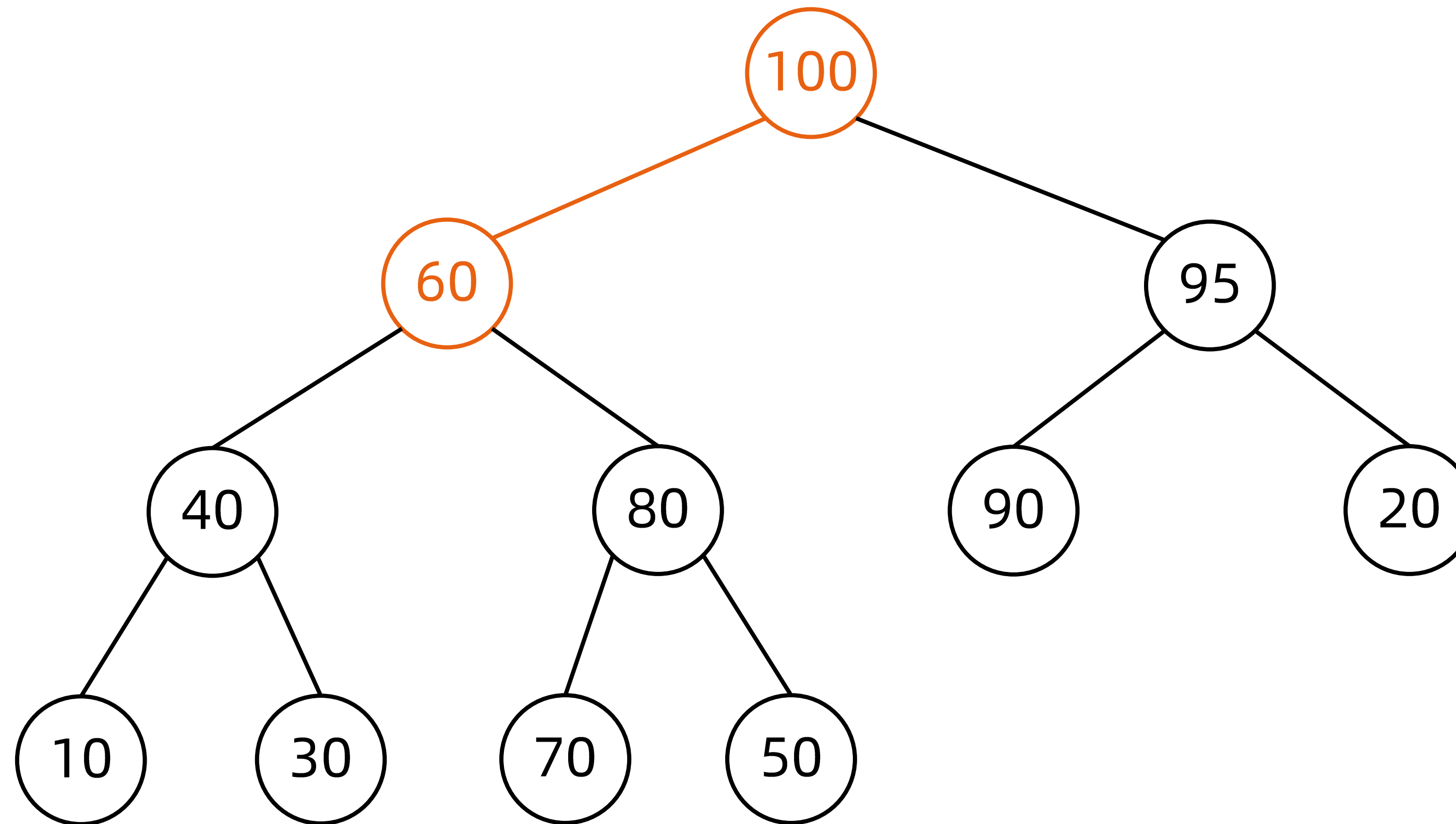
$O(\log N)$

# Step 1



一维数组 [0, 60, 100, 95, 40, 80, 90, 20, 10, 30, 70, 50, ~~110~~]

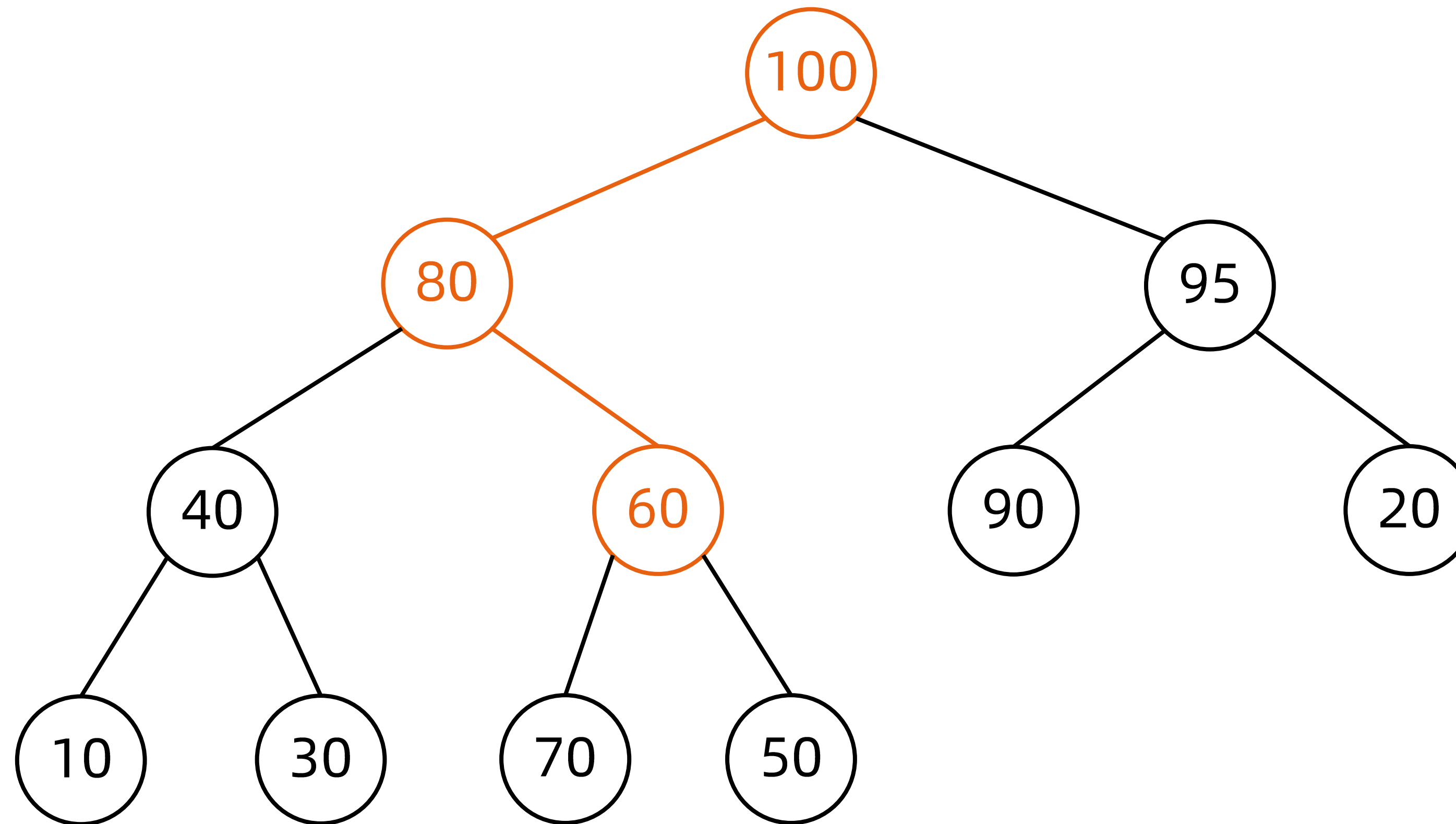
## Step 2



一维数组 [0, 100, 60, 95, 40, 80, 90, 20, 10, 30, 70, 50]  
Index      1    2    3

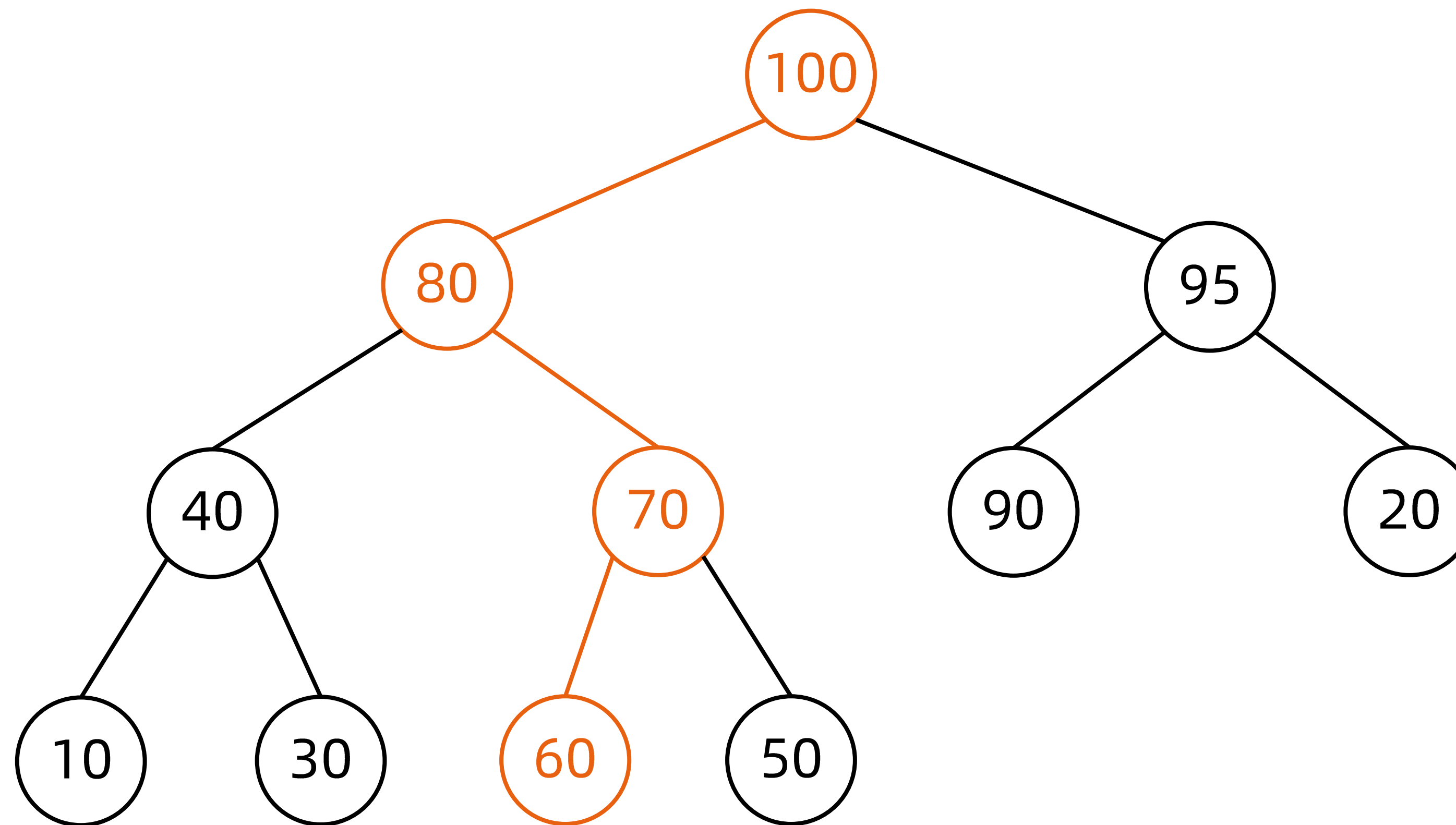


# Step 3



一维数组 [0, 100, 80, 95, 40, 60, 90, 20, 10, 30, 70, 50]  
Index                      2            4    5

## Step 4



一维数组 [0, 100, 80, 95, 40, 70, 90, 20, 10, 30, 60, 50]  
Index 5 10 11

# 优先队列（Priority Queue）

二叉堆是优先队列（Priority Queue）一种简单、常见的实现，但不是最优实现

理论上二叉堆可以支持  $O(\log N)$  删除任意元素，只需要

- 定位该元素在堆中的结点  $p$ （可以通过在数值与索引之间建立映射得到）
- 与堆尾交换，删除堆尾
- 从  $p$  向上、向下各进行一次调整

不过优先队列并没有提供这个方法

在各语言内置的库中，需要支持删除任意元素时，一般使用有序集合等基于平衡二叉搜索树的实现

# 实战

合并K个升序链表

<https://leetcode-cn.com/problems/merge-k-sorted-lists/>

对比分治合并和堆的解法

Homework: 学习自己所用语言中重载比较运算符的方法

- C++: `bool operator <(const Node& a, const Node& b)`
- Java: `class Node implements Comparable<Node>`
- Python: `def __lt__(self, other)`



# 实战

滑动窗口最大值

<https://leetcode-cn.com/problems/sliding-window-maximum/>

对比单调队列和堆的解法

懒惰删除法

- 指的是需要从堆中删除一个元素（不一定是最大值）时，不直接删除，而是打上删除标记（soft delete）
- 等未来它作为最大值被取出时，再判断它是否已经被标记，若是则抛弃它，取下一个最大值

“懒惰”的含义——只要要删除的元素还不是最大值，在堆里多待一会儿也无妨，未来等它会影响最大值正确性的时候再说吧

# Homework

设计推特

<https://leetcode-cn.com/problems/design-twitter/>

模块化：检索最近的十条推文，其实就是合并若干个有序链表  
（自己的链表，以及关注的每个人的链表）

# 二叉搜索树

# 二叉搜索树 Binary Search Tree

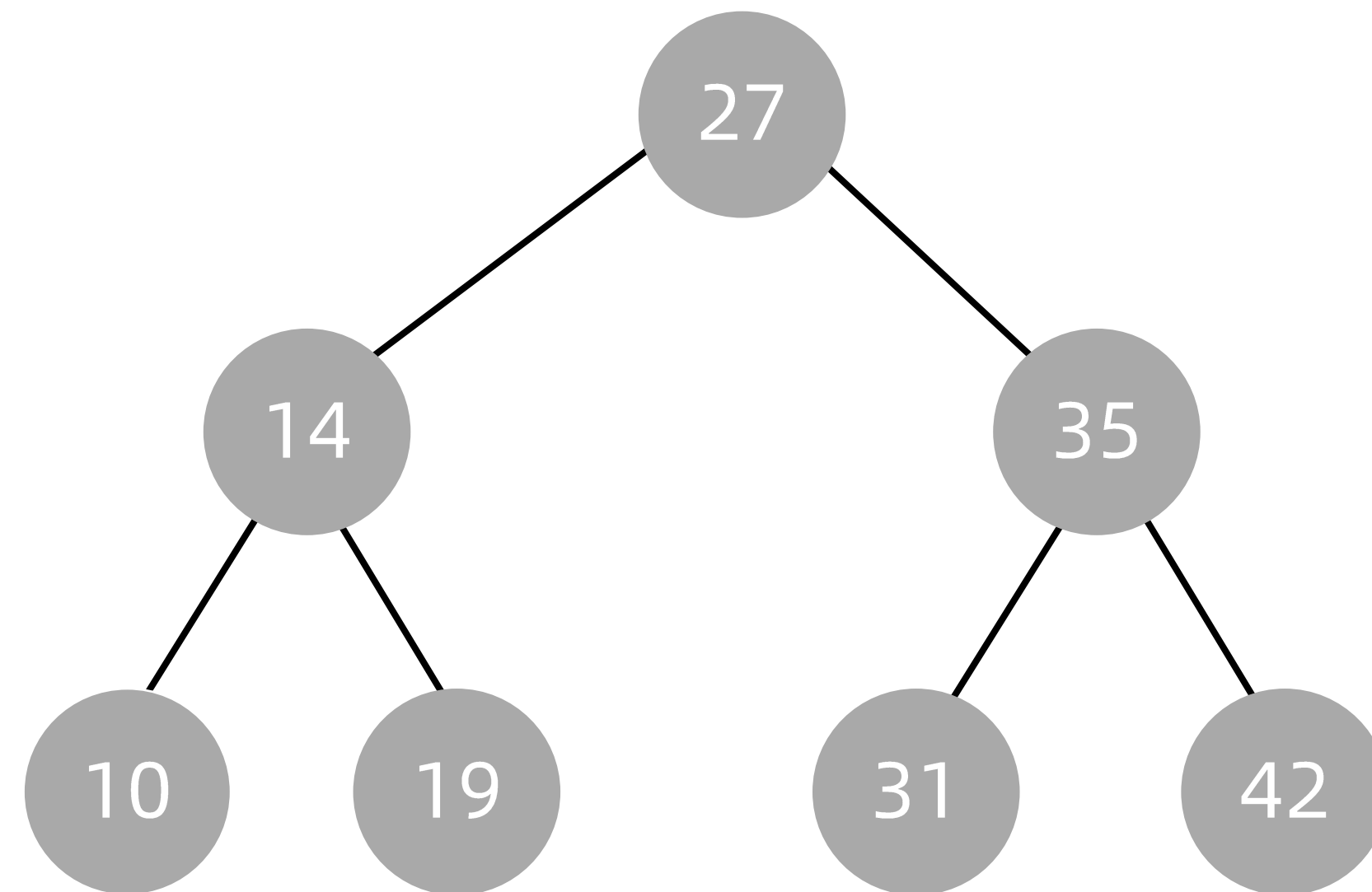
二叉搜索树（Binary Search Tree）是一棵满足如下性质（BST性质）的二叉树：

- 任意结点的关键码  $\geq$  它左子树中所有结点的关键码
- 任意结点的关键码  $\leq$  它右子树中所有结点的关键码

根据以上性质，二叉搜索树的中序遍历必然为一个有序序列



# 二叉搜索树 Binary Search Tree

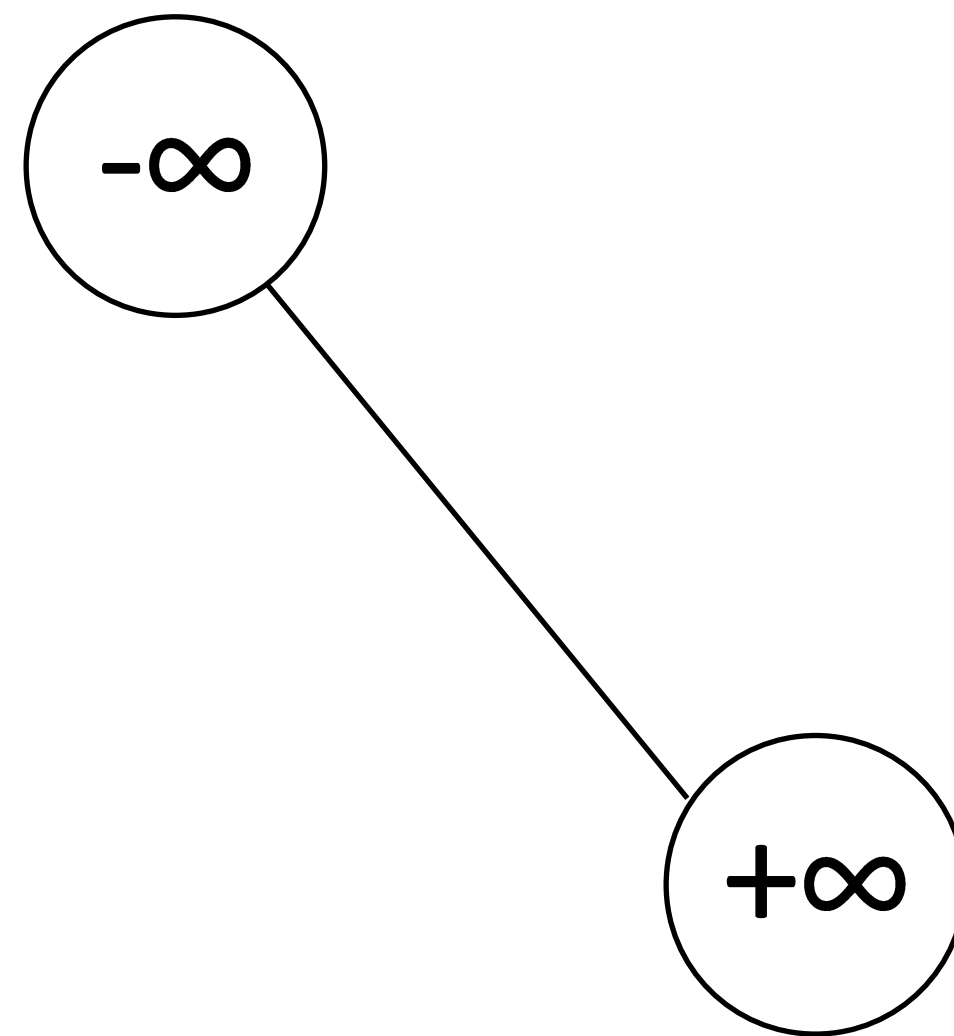


# BST - 建立

为了避免越界，减少边界情况的特殊判断，一般在 BST 中额外插入两个保护结点

- 一个关键码为正无穷（一个很大的正整数）
- 一个关键码为负无穷

仅由这两个结点构成的 BST 就是一棵初始的空 BST



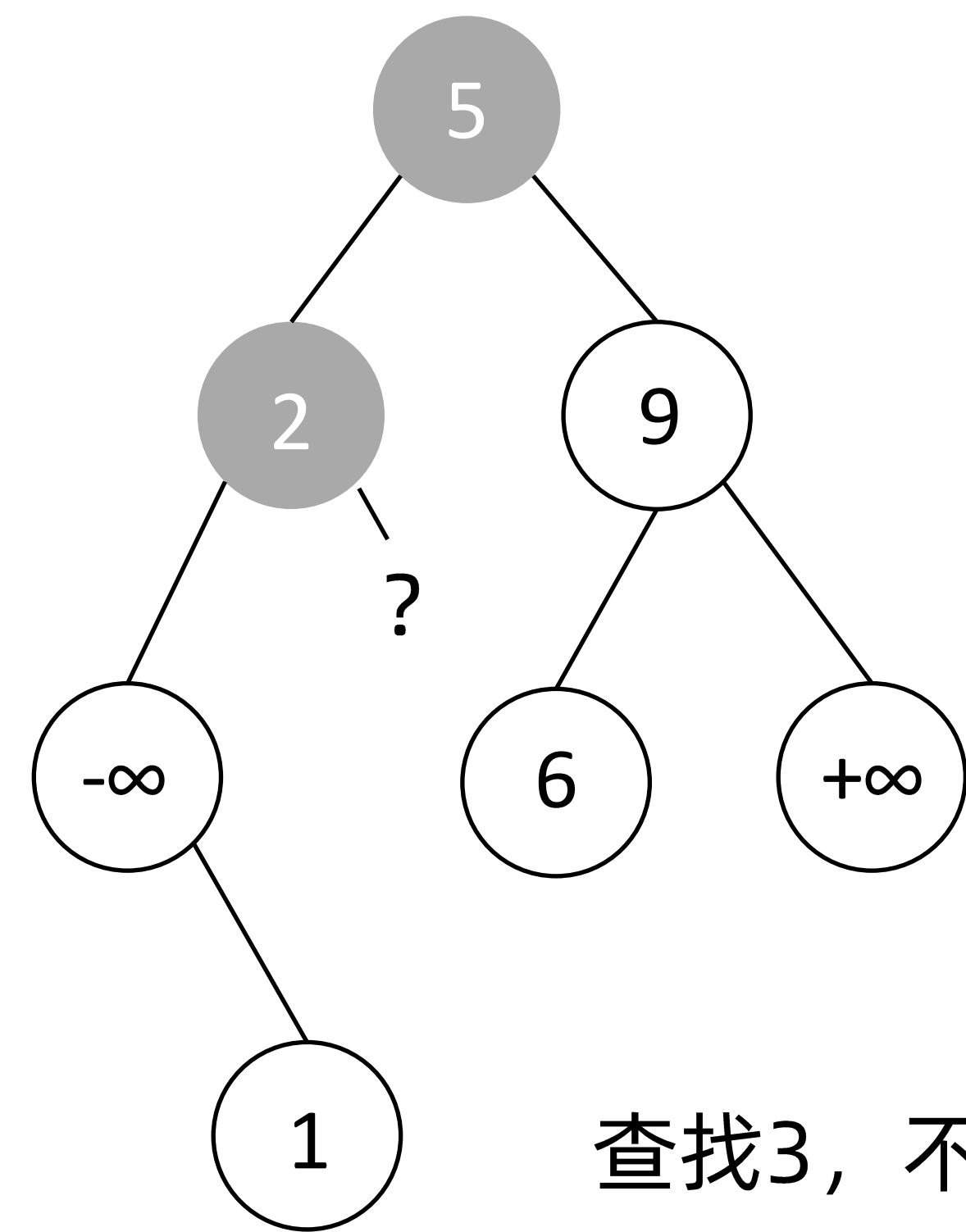
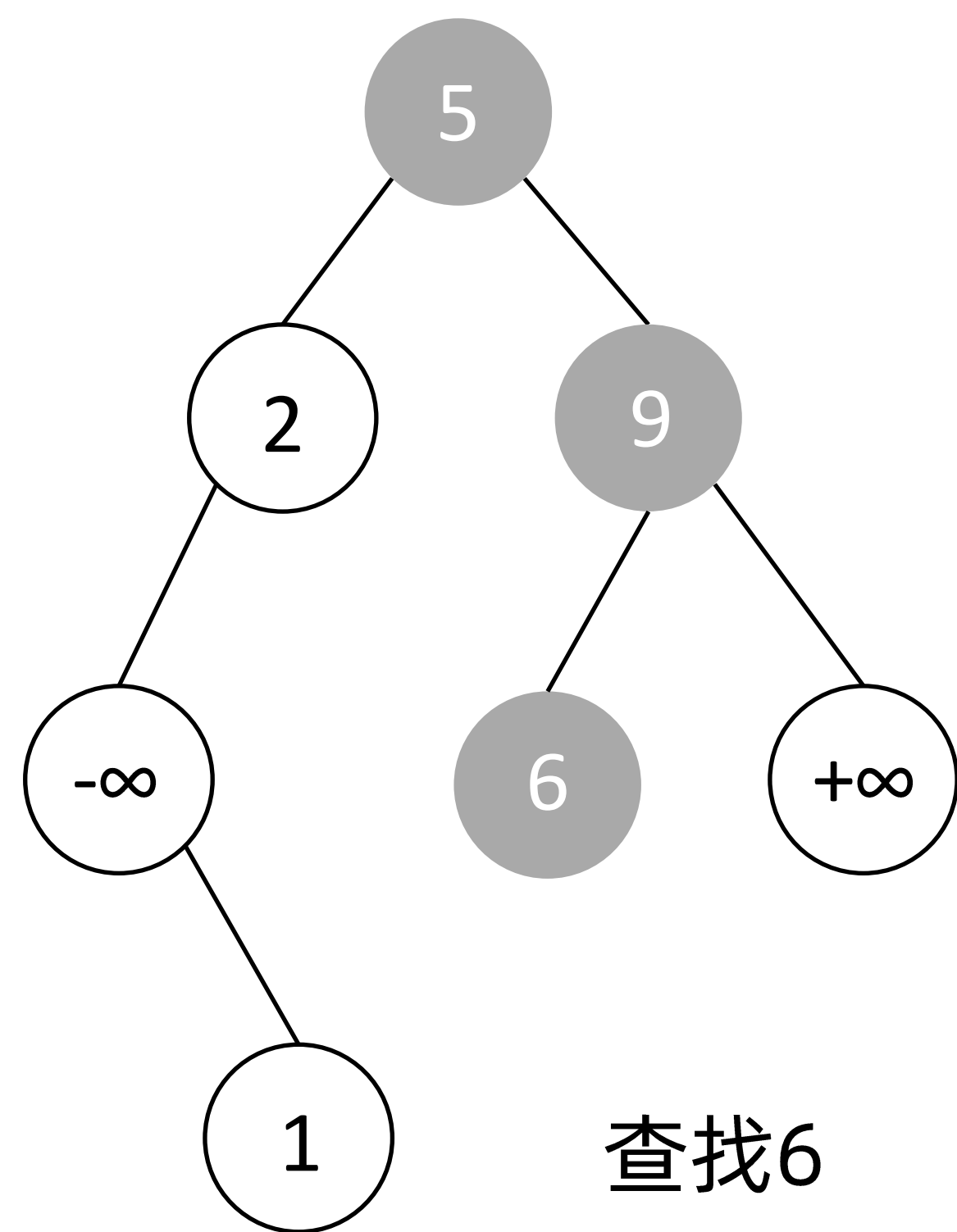
# BST – 检索

检索关键码 `val` 是否存在

从根开始递归查找

- 若当前结点的关键码等于 `val`，则已经找到
- 若关键码大于 `val`，递归检索左子树（或不存在）
- 若关键码小于 `val`，递归检索右子树（或不存在）

# BST - 检索

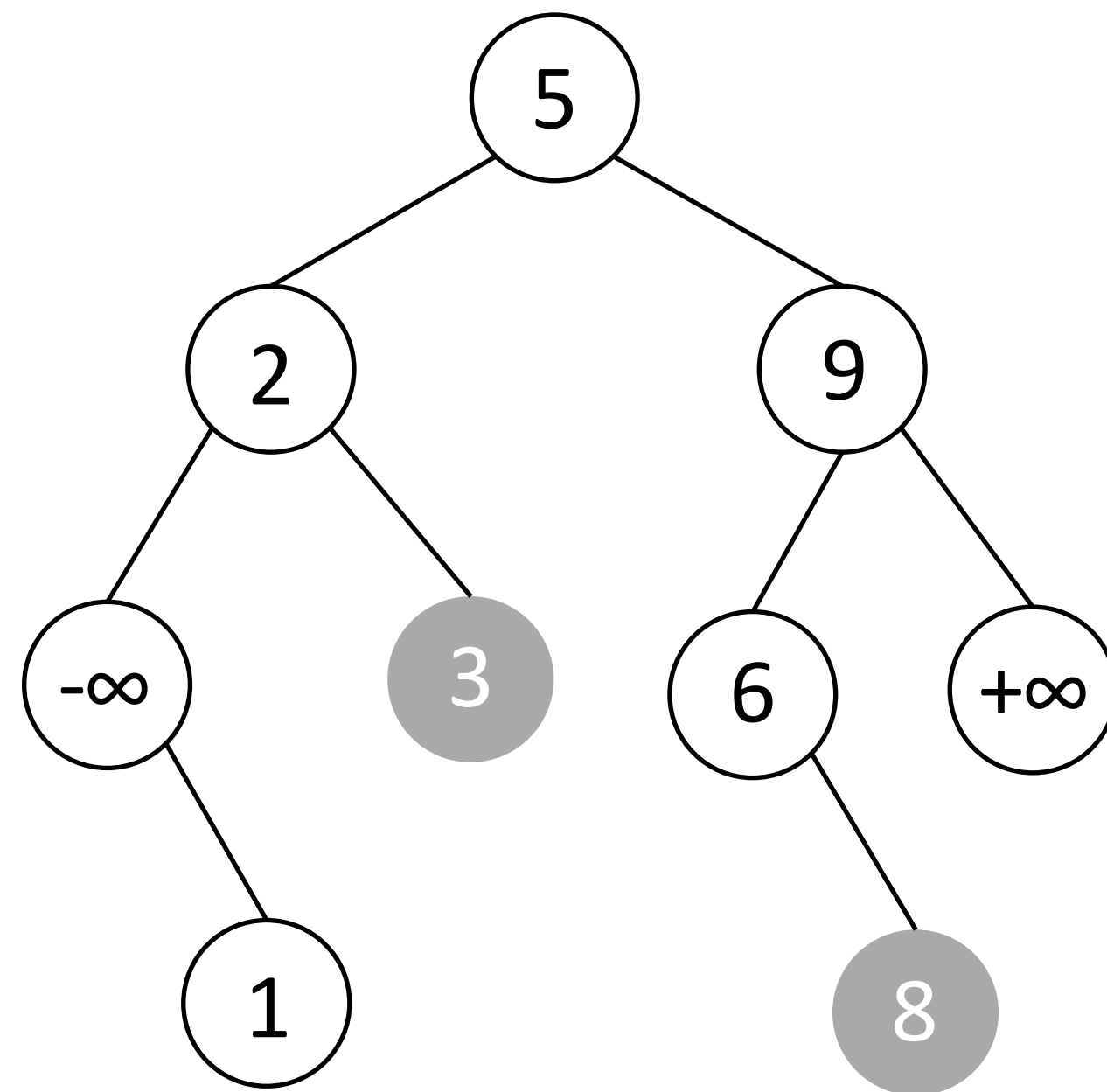




# BST - 插入

插入 val 与检索 val 的过程类似

- 若检索发现存在，则放弃插入（或者把 val 对应结点的计数 +1，视要求而定）
- 若检索发现不存在（子树为空），直接在对应位置新建关键码为 val 的结点



插入3和8

# BST - 求前驱/后继

前驱：BST 中小于 val 的最大结点

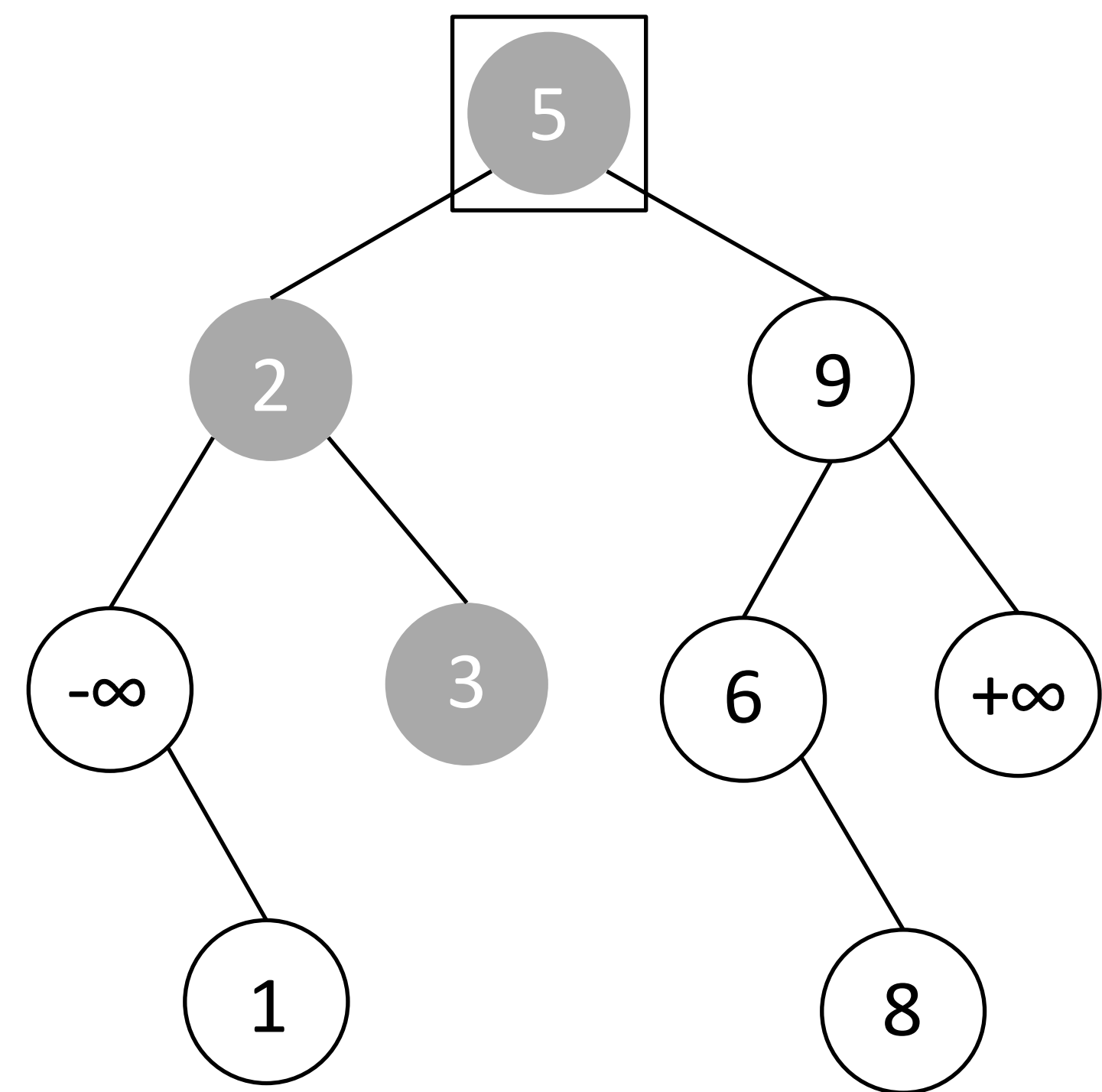
后继：BST 中大于 val 的最小结点

求前驱和后继也是基于检索的，先检索 val

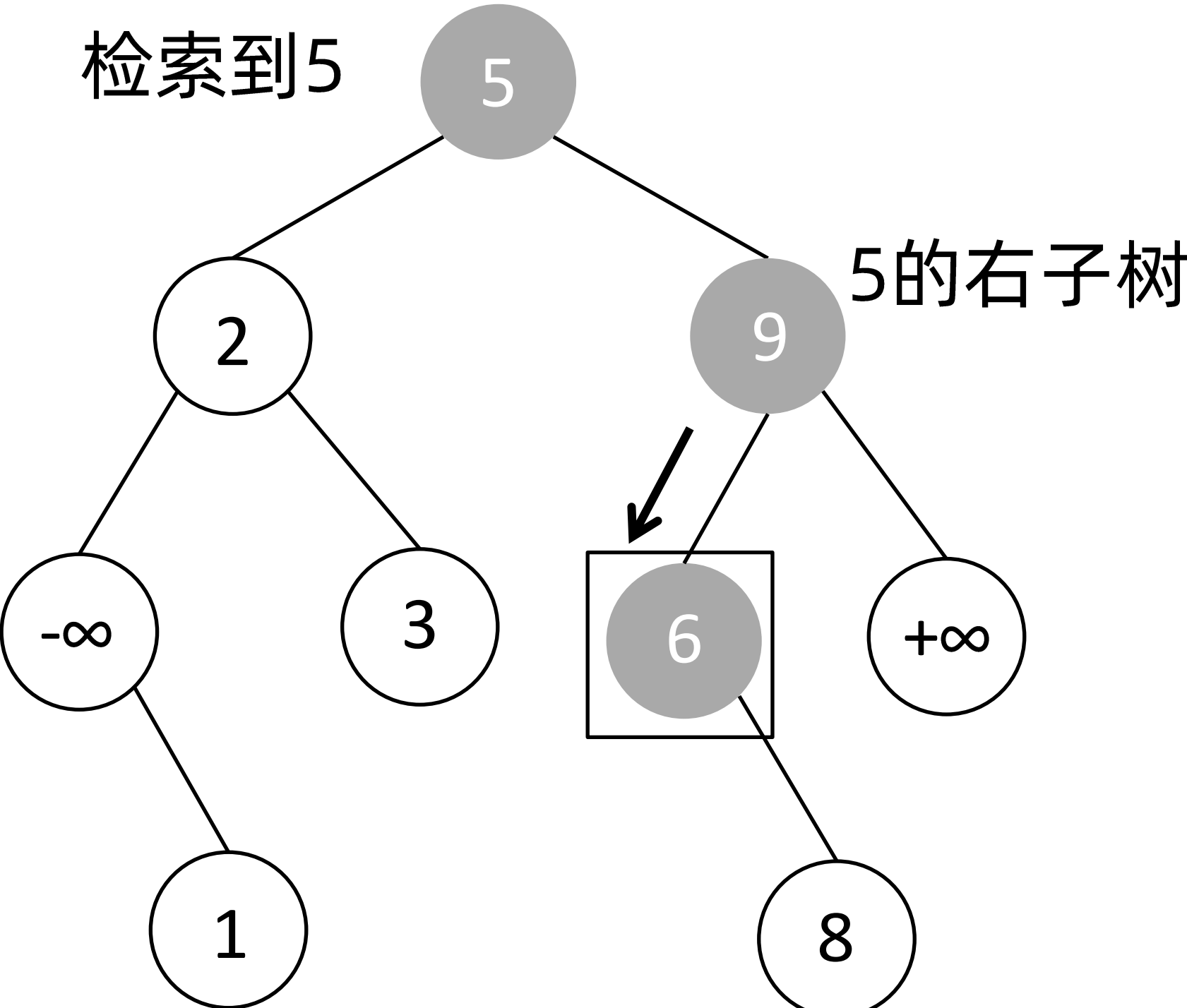
以后继为例：

- 如果检索到了 val，并且 val 存在右子树，则在右子树中一直往左走到底
- 否则说明没找到 val 或者 val 没有右子树，此时前驱就在检索过程经过的结点中（即当前结点的所有祖先节点，可以拿一个变量顺便求一下）

# BST - 求前驱/后继



求3的后继



求5的后继

# BST - 删除

从 BST 中删除关键码为 val 的结点，可以基于检索 + 求后继实现

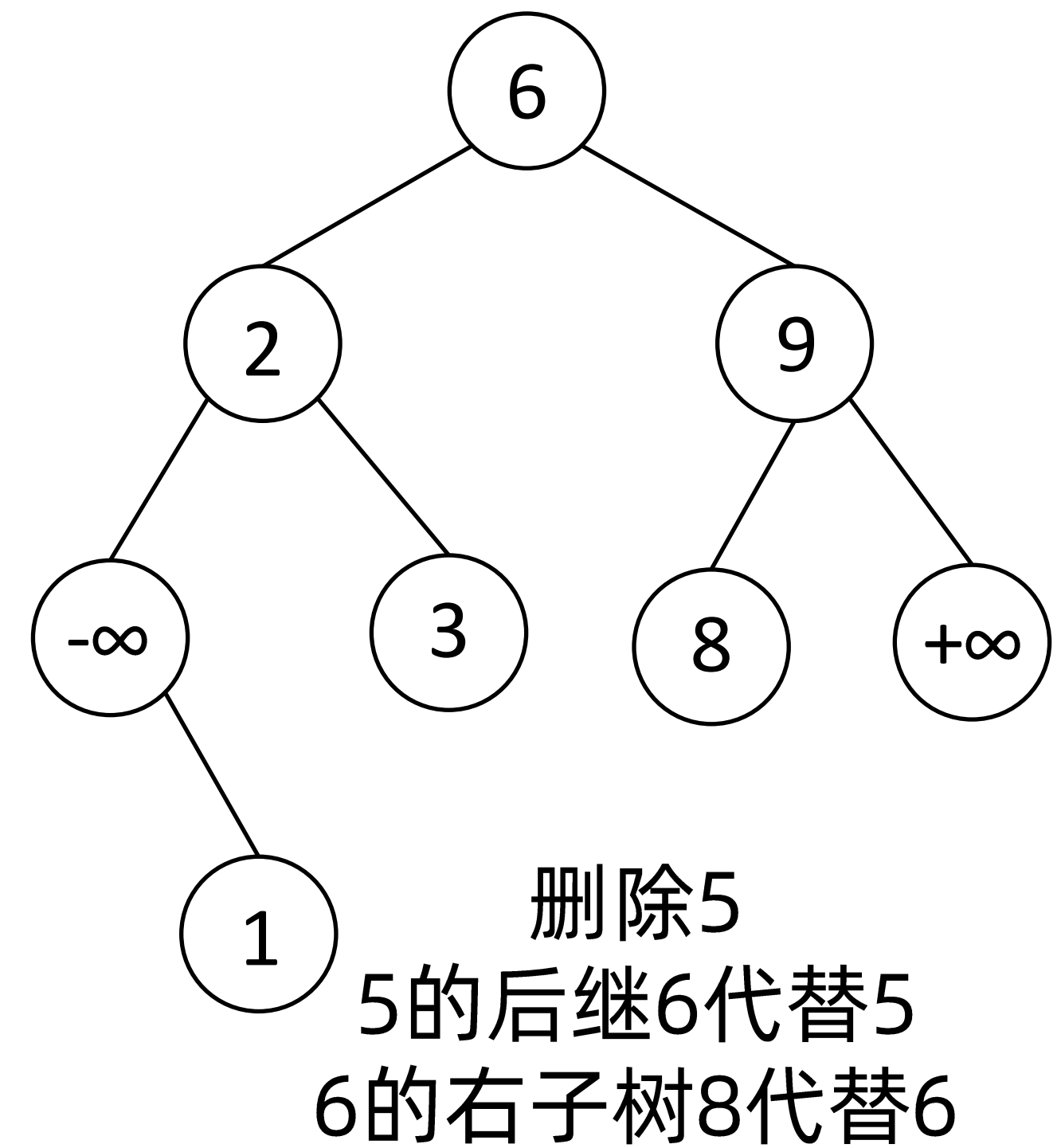
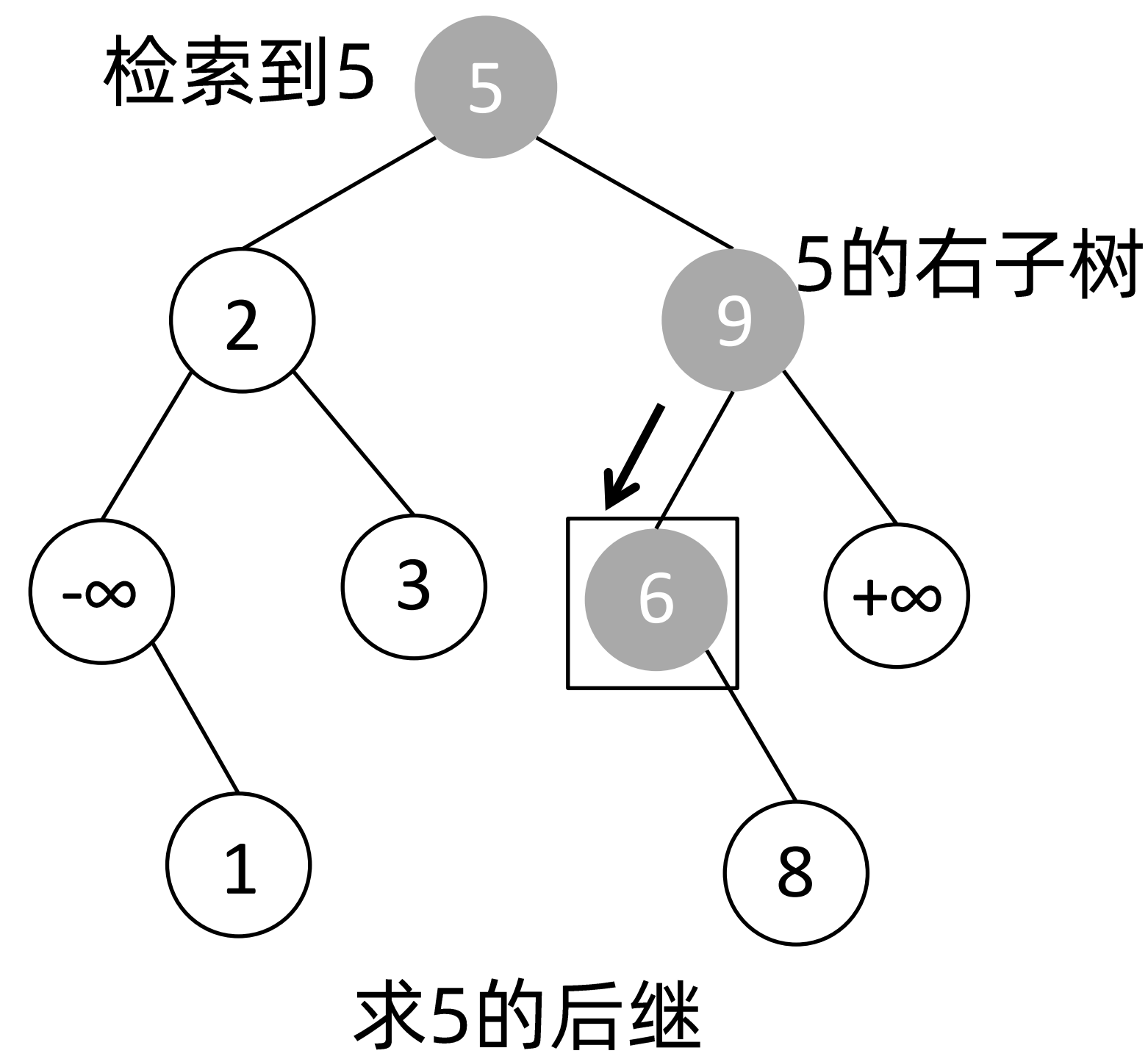
首先检索 val

如果 val 只有一棵子树，直接删除 val，把子树和父结点相连就行了

如果有两棵子树，需要找到后继，先删除后继，再用后继结点代替 val 的位置

（因为后继是右子树一直往左走到底，所以它最多只会有一棵子树）

# BST - 删除



# 二叉搜索树 Binary Search Tree

查询/插入/求前驱/求后继/删除操作的时间复杂度:

随机数据期望  $O(\log N)$

在非随机数据上, BST容易退化为  $O(N)$ , 一般都要结合旋转操作来进行平衡

平衡二叉树将在后面的课程中讲解

# 实战

二叉搜索树中的插入操作

<https://leetcode-cn.com/problems/insert-into-a-binary-search-tree/>

后继者

<https://leetcode-cn.com/problems/successor-lcci/>

删除二叉搜索树中的节点

<https://leetcode-cn.com/problems/delete-node-in-a-bst/>

把二叉搜索树转换为累加树（Homework）

<https://leetcode-cn.com/problems/convert-bst-to-greater-tree/>



# THANKS

 极客时间 | 训练营