分而治之篇：堆排序与线性时间排序

盛 浩

shenghao@buaa.edu.cn

北京航空航天大学

计算机学院

北航《算法设计与分析》

提纲

优先队列

（二叉）堆

堆排序

排序算法的下界

计数排序

提纲

- 优先队列
- （二叉）堆
- 堆排序
- 排序算法的下界
- 计数排序

优先队列：示例

- 现有3个作业A，B，C按顺序提交到一台打印机，考虑打印机完成这三个作业所需的时间。

作业量：

作业A：100页

作业B：10页

作业C：1页

优先队列：示例

- 现有3个作业A，B，C按顺序提交到一台打印机，考虑打印机完成这三个作业所需的时间。

作业量：

作业A：100页

作业B：10页

作业C：1页

- 平均完成时间（先来先服务，FIFO）
  - $(100 + 110 + 111) / 3 = 107\ time\ units$

优先队列：示例

- 现有3个作业A，B，C按顺序提交到一台打印机，考虑打印机完成这三个作业所需的时间。

作业量：

作业A：100页

作业B：10页

作业C：1页

- 平均完成时间（先来先服务，FIFO）
  - (100 + 110 + 111) / 3 = 107 *time units*
- 平均完成时间（短作业优先，SJF）
  - (1 + 11 + 111) / 3 = 41 *time units*

优先队列：示例

- 队列中的元素是待打印的作业，每个作业将各自的页数作为其优先级
- 短作业优先的方式相当于从队列中提取最小的元素 **(Extract-Min)**
- 当有新的作业到来时需进行入队 **(Insert)**

优先队列：示例

- 队列中的元素是待打印的作业，每个作业将各自的页数作为其优先级
- 短作业优先的方式相当于从队列中提取最小的元素 **(Extract-Min)**
- 当有新的作业到来时需进行入队 **(Insert)**

问题：是否有一个队列能够支持两种操作：**Insert**和**Extract-Min**?

优先队列

优先队列是一个抽象的数据结构，支持下述两种操作：

- **Insert**： 将新元素插入队列中
- **Extract-Min**：删除并返回队列中最小的元素。

可能的实现方式

- 无序列表 + 一个指向最小元素的指针
  - **Insert**：$\boldsymbol{O(1)}$
  - **Extract-Min**：$\boldsymbol{O(n)}$，需要线性时间来找到最小元素

可能的实现方式

- 无序列表 + 一个指向最小元素的指针
  - Insert：$O(1)$
  - Extract-Min：$O(n)$，需要线性时间来找到最小元素
- 有序数组
  - Insert：$O(n)$
  - Extract-Min：$O(1)$

可能的实现方式

- 无序列表 + 一个指向最小元素的指针
  - Insert： $\boldsymbol{O(1)}$
  - Extract-Min： $\boldsymbol{O(n)}$，需要线性时间来找到最小元素
- 有序数组
  - Insert： $\boldsymbol{O(n)}$
  - Extract-Min： $\boldsymbol{O(1)}$
- 双向有序链表
  - Insert： $\boldsymbol{O(n)}$
  - Extract-Min： $\boldsymbol{O(1)}$

可能的实现方式

- 无序列表 + 一个指向最小元素的指针
  - Insert：$\boldsymbol{O}(\mathbf{1})$
  - Extract-Min：$\boldsymbol{O}(\boldsymbol{n})$，需要线性时间来找到最小元素
- 有序数组
  - Insert：$\boldsymbol{O}(\boldsymbol{n})$
  - Extract-Min：$\boldsymbol{O}(\mathbf{1})$
- 双向有序链表
  - Insert：$\boldsymbol{O}(\boldsymbol{n})$
  - Extract-Min：$\boldsymbol{O}(\mathbf{1})$

问题: 是否有一种数据结构使得**Insert**和**Extract-Min**的时间复杂度均为$\boldsymbol{O}(\mathbf{log}\,\boldsymbol{n})$？

提纲

优先队列

（二叉）堆

堆排序

排序算法的下界

计数排序

(二叉)堆

堆的定义

1. n个元素的序列(k1, k2, … , kn)，当且仅当满足

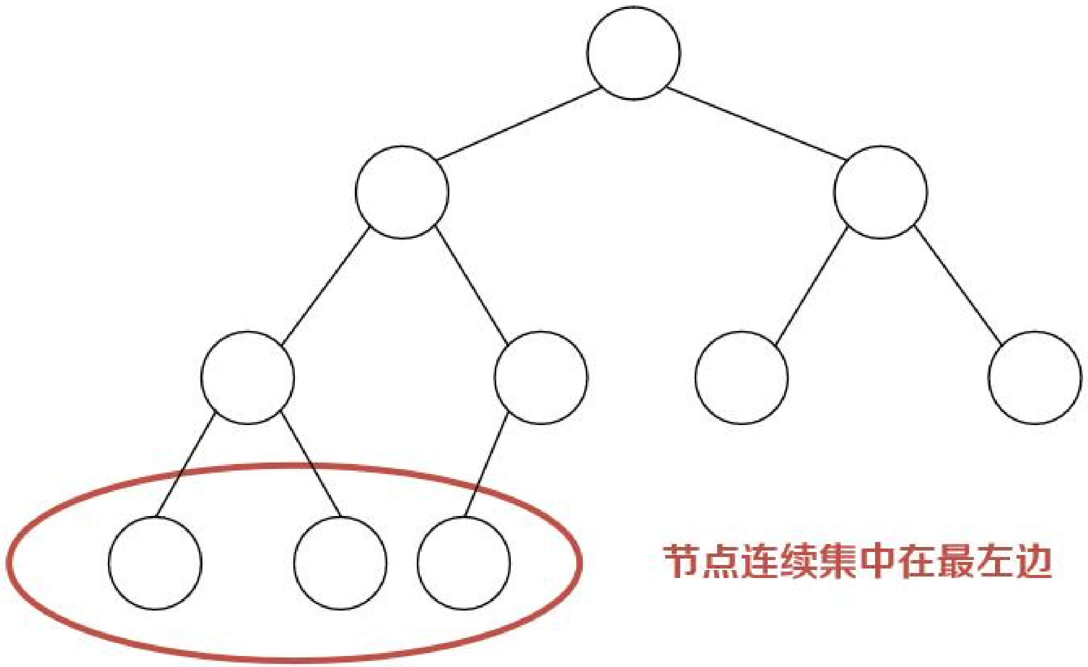
(1) $\begin{cases} ki \geq k2i \\ ki \geq k2i+1 \end{cases}$ 或者 **(2)** $\begin{cases} ki \leq k2i \\ ki \leq k2i+1 \end{cases}$

$i=1, 2, 3, \ldots, \lfloor n/2 \rfloor$

称该序列为一个**堆积(heap)**，简称**堆**。

称满足条件(1)的堆为 大顶堆 ，称满足条件(2)的堆为 小顶堆 。

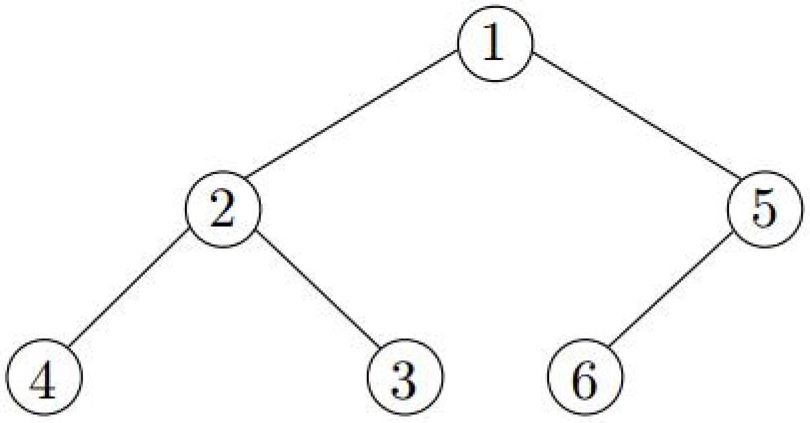
- 堆可以视为一颗完全二叉树
  - 除最后一层节点数可能不满外，其余层数的节点数目均达到最大个数
  - 最后一层的节点都连续集中在最左边



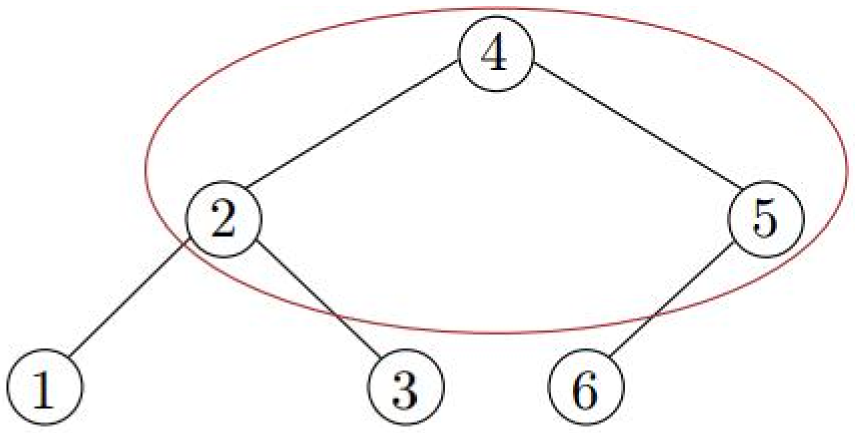
堆的特性

- 堆顺序特性（小根/顶堆）
  - 父节点的值比每一个子节点的值都要小。
  - 且每一棵子树也满足堆的特性。

$$A[Parent(i)] \le A[i]$$



小根堆



不属于堆

堆的特性

- 如果保持了堆的顺序特性，堆可以高效地支持以下操作（假设堆中有$\boldsymbol{n}$个元素）
  - 在$\boldsymbol{O(log\ n)}$时间内**Insert**
  - 在$\boldsymbol{O(log\ n)}$时间内**Extract-Min**

堆的特性

- 如果保持了堆的顺序特性，堆可以高效地支持以下操作（假设堆中有$\boldsymbol{n}$个元素）
  - 在$\boldsymbol{O(\log n)}$时间内**Insert**
  - 在$\boldsymbol{O(\log n)}$时间内**Extract-Min**

- 结构属性
  - 一高度为$\boldsymbol{h}$的堆有$\boldsymbol{2^h}$到$\boldsymbol{2^{h+1}-1}$个节点。因此，一包含$\boldsymbol{n}$个节点的的堆的高度为$\boldsymbol{\Theta(\log n)}$。

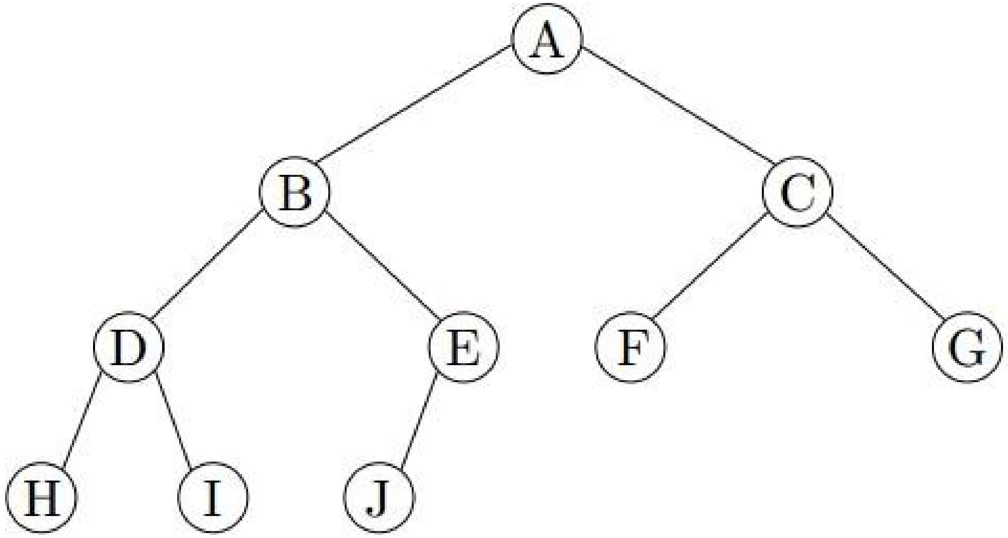
堆的特性

- 如果保持了堆的顺序特性，堆可以高效地支持以下操作（假设堆中有$\boldsymbol{n}$个元素）
  - 在$\boldsymbol{O(\log n)}$时间内**Insert**
  - 在$\boldsymbol{O(\log n)}$时间内**Extract-Min**

- 结构属性
  - 一高度为$\boldsymbol{h}$的堆有$\mathbf{2}^{\boldsymbol{h}}$到$\mathbf{2}^{\boldsymbol{h}+\mathbf{1}}-\mathbf{1}$个节点。因此，一包含$\boldsymbol{n}$个节点的的堆的高度为$\boldsymbol{\Theta(\log n)}$。
  - 该结构非常有规律，可以使用数组来表示，且不需要任何链接！

堆的数组实现

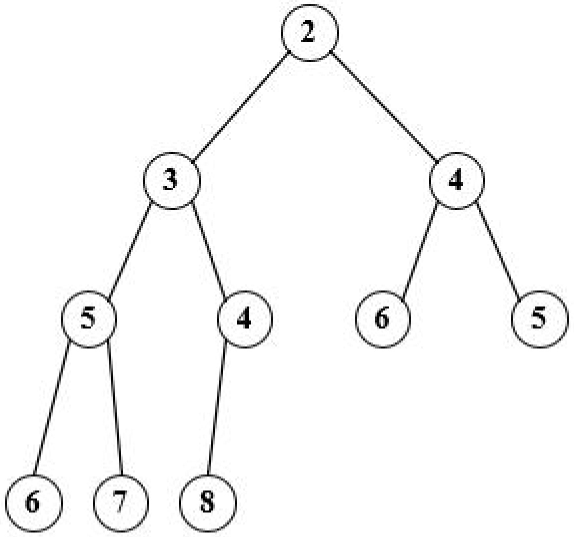
- 数组第一个元素表示根节点
- 对每个**a[i]**都有
  - 其左孩子的数组下标为**2i**
  - 其右孩子的数组下标为**2i+1**
  - 其父节点的数组下标为$\lfloor i/2 \rfloor$
- 我们将堆用树的形式表示，实际实现时使用更简单的数组



1	2	3	4	5	6	7	8	9	10
A	B	C	D	E	F	G	H	I	J

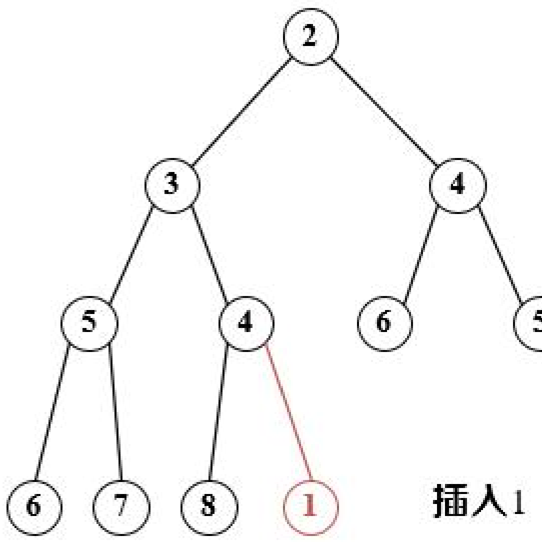
插入

- 首先将待插入的节点添加到最底层下一个可用位置
- 如果违反了小根堆的顺序特性，则调整节点位置，恢复小根堆的特性
  - 一般的策略是向上调整（或向上冒泡）：如果当前节点的父节点的值比当前节点的值大，那么交换当前节点与父节点的位置。



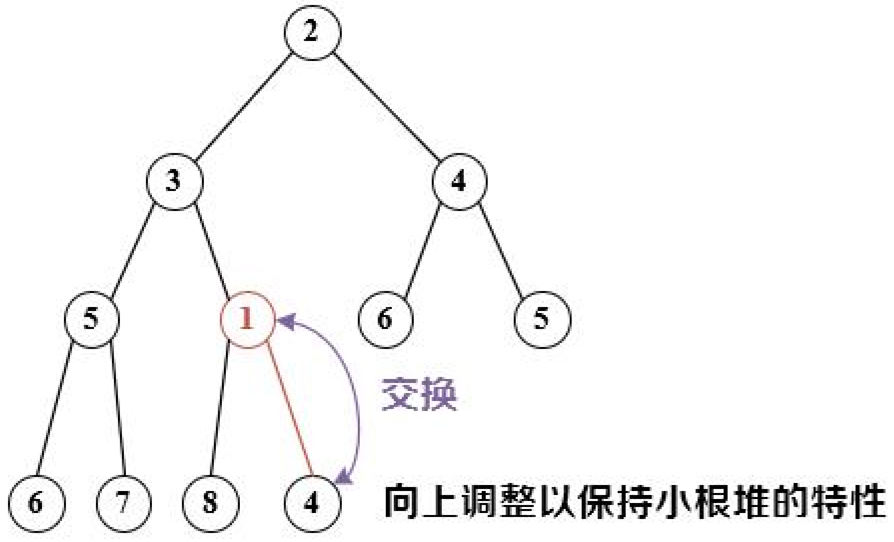
插入

- 首先将待插入的节点添加到最底层下一个可用位置
- 如果违反了小根堆的顺序特性，则调整节点位置，恢复小根堆的特性
  - 一般的策略是向上调整（或向上冒泡）：如果当前节点的父节点的值比当前节点的值大，那么交换当前节点与父节点的位置。



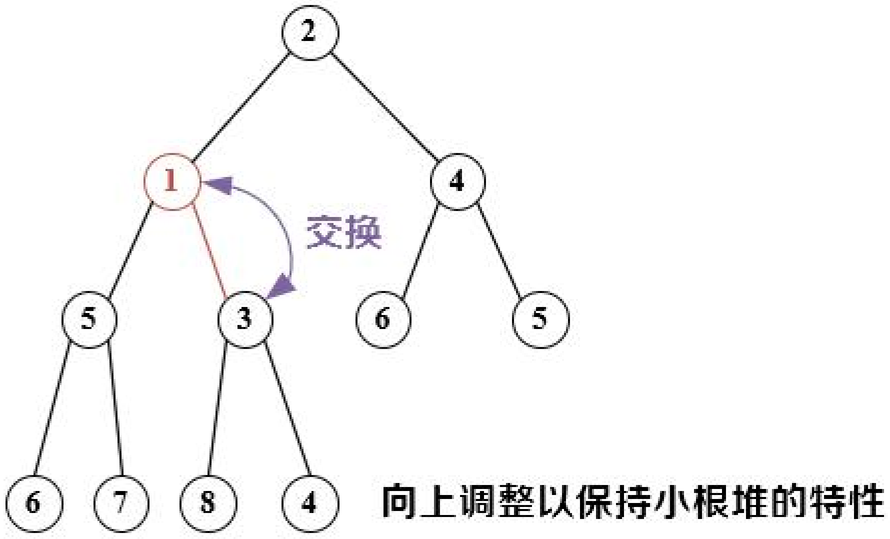
插入

- 首先将待插入的节点添加到最底层下一个可用位置
- 如果违反了小根堆的顺序特性，则调整节点位置，恢复小根堆的特性
    - 一般的策略是向上调整（或向上冒泡）：如果当前节点的父节点的值比当前节点的值大，那么交换当前节点与父节点的位置。



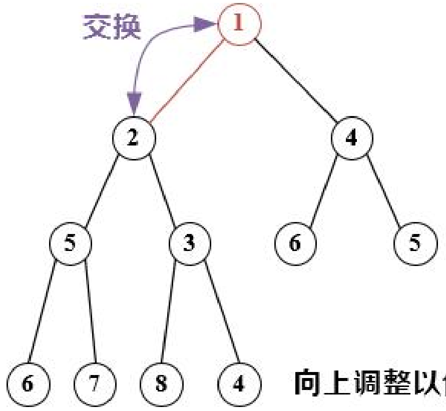
插入

- 首先将待插入的节点添加到最底层下一个可用位置
- 如果违反了小根堆的顺序特性，则调整节点位置，恢复小根堆的特性
  - 一般的策略是向上调整（或向上冒泡）：如果当前节点的父节点的值比当前节点的值大，那么交换当前节点与父节点的位置。



插入

- 首先将待插入的节点添加到最底层下一个可用位置
- 如果违反了小根堆的顺序特性，则调整节点位置，恢复小根堆的特性
  - 一般的策略是向上调整（或向上冒泡）：如果当前节点的父节点的值比当前节点的值大，那么交换当前节点与父节点的位置。

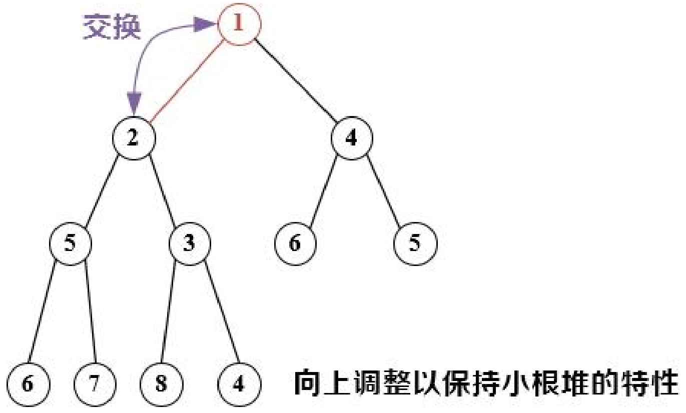


向上调整以保持小根堆的特性

- 正确性：每次交换后对于以新节点为根节点的子树来说，小根堆的特性均得以满足。

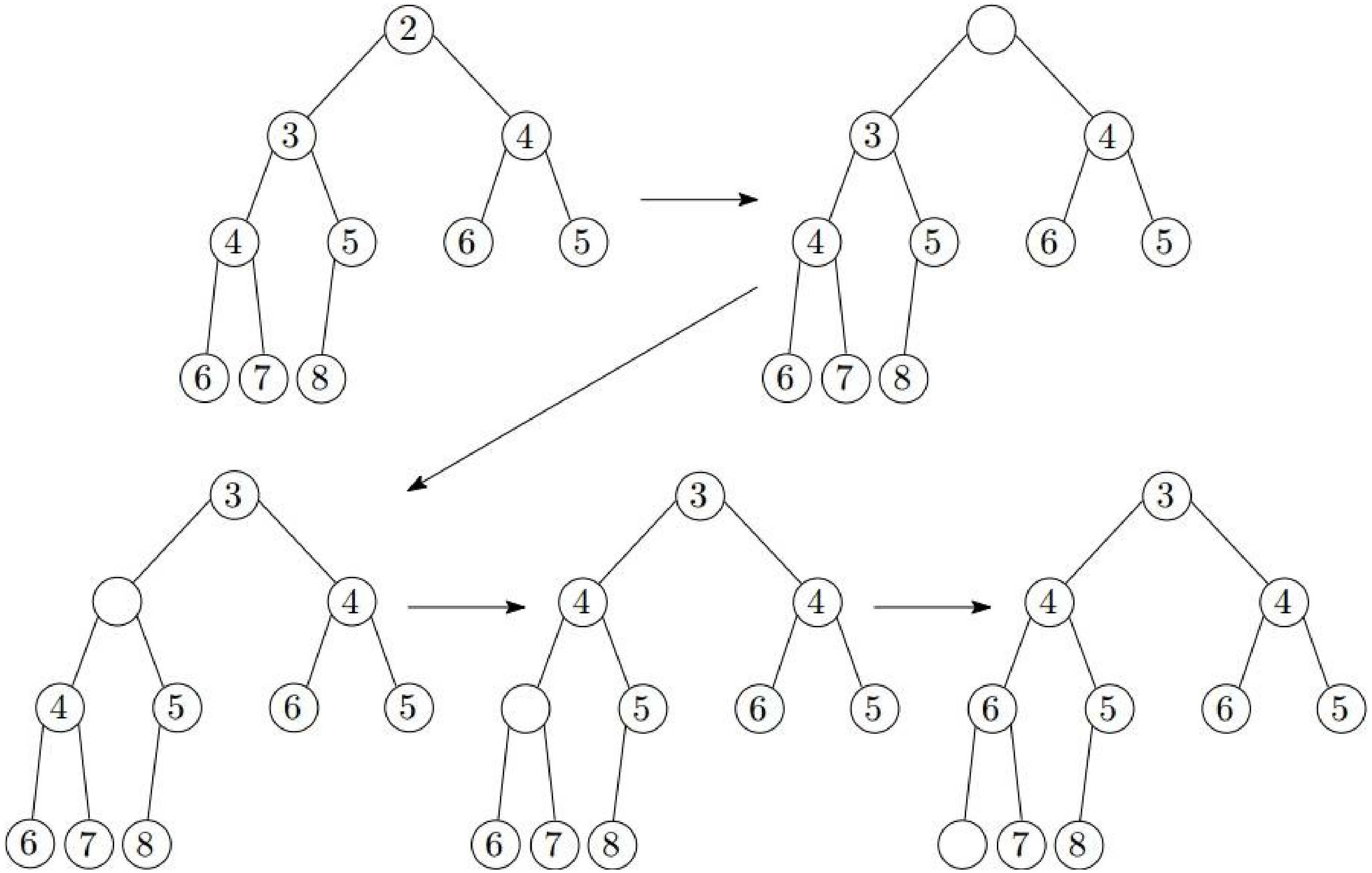
插入

- 首先将待插入的节点添加到最底层下一个可用位置
- 如果违反了小根堆的顺序特性，则调整节点位置，恢复小根堆的特性
  - 一般的策略是向上调整（或向上冒泡）：如果当前节点的父节点的值比当前节点的值大，那么交换当前节点与父节点的位置。

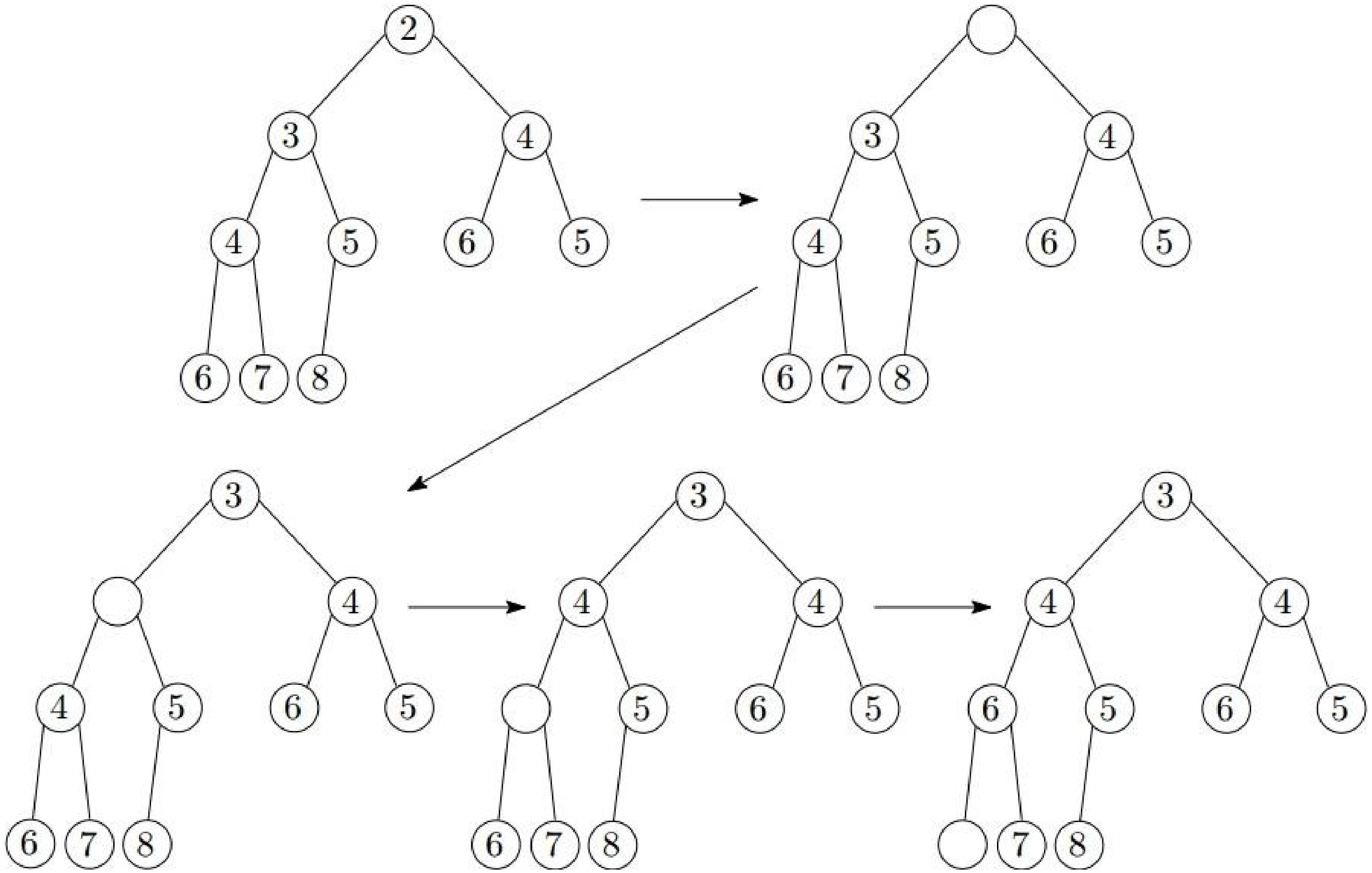


- 正确性：每次交换后对于以新节点为根节点的子树来说，小根堆的特性均得以满足。
- 时间复杂度 $= O(height) = O(\log n)$

获取最小值：首次尝试



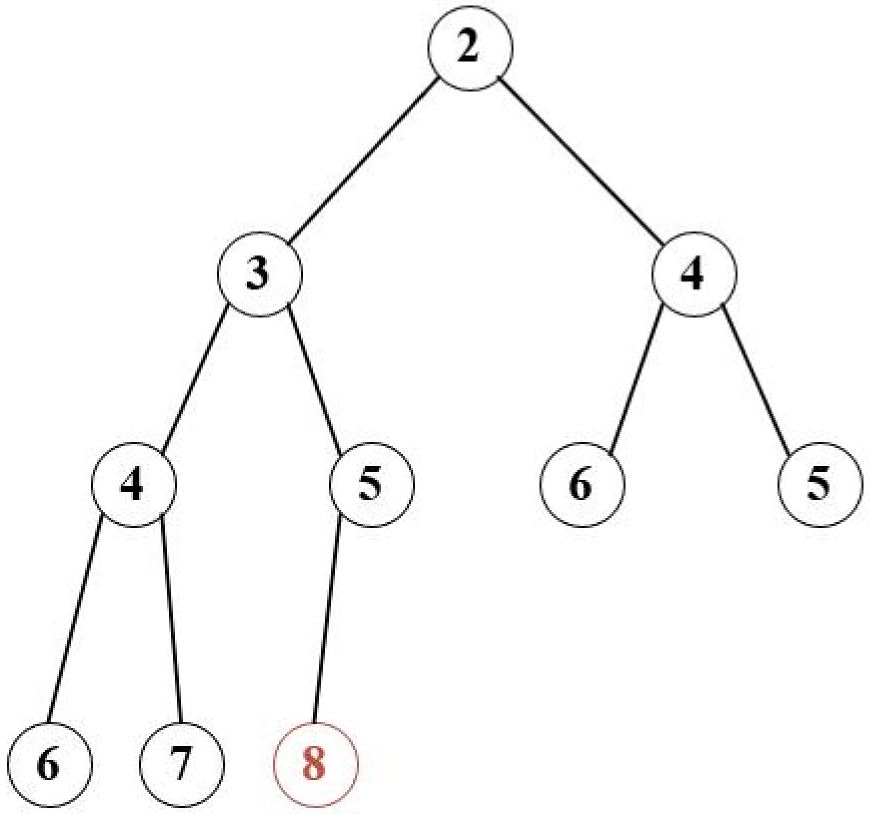
获取最小值：首次尝试



满足小根堆的顺序特性，但完全性被破坏！

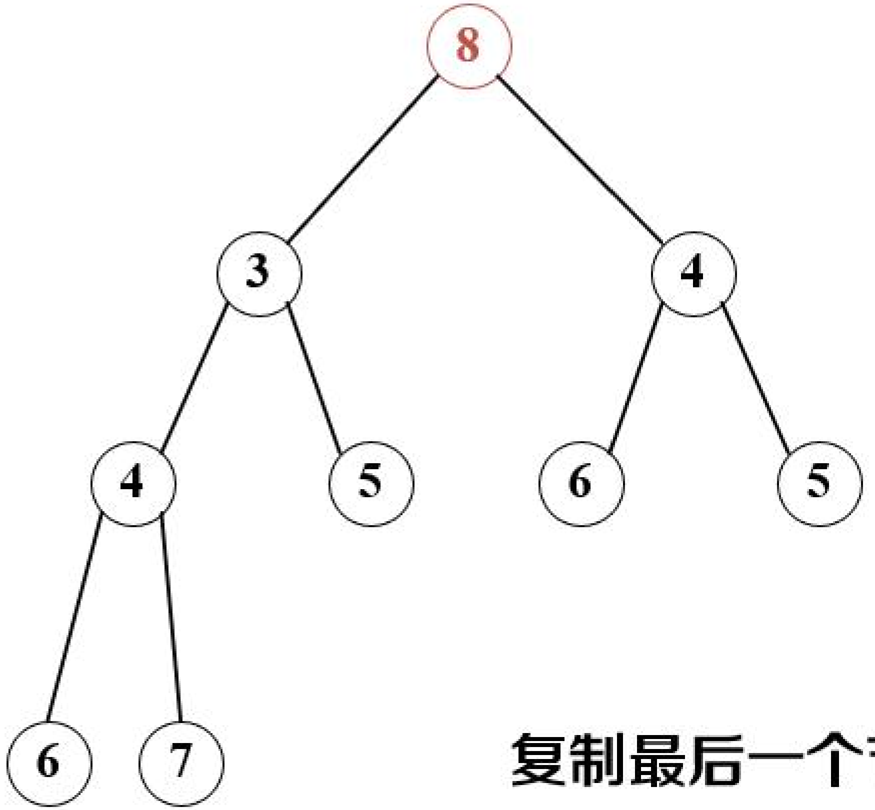
获取最小值

- 将最后一个节点复制到根节点（即覆盖存储在根节点处的最小元素）
- 通过向下调整（或向下冒泡）恢复小根堆的特性：如果该节点的值比它任何一个子节点的值都大，那么就将它和子节点中较小的那个节点进行交换。



获取最小值

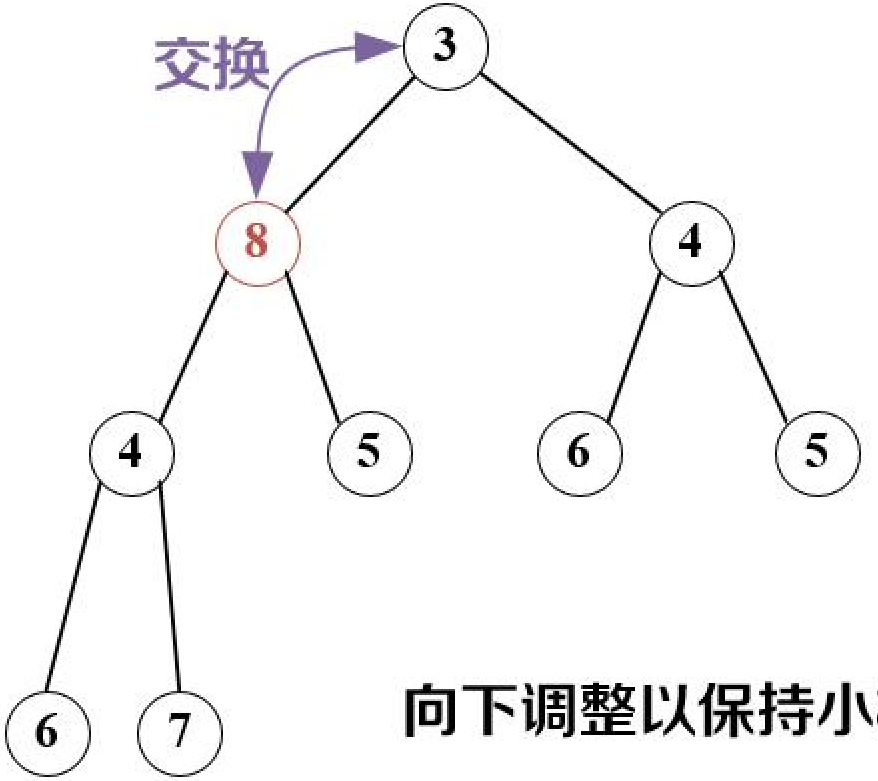
- 将最后一个节点复制到根节点（即覆盖存储在根节点处的最小元素）
- 通过向下调整（或向下冒泡）恢复小根堆的特性：如果该节点的值比它任何一个子节点的值都大，那么就将它和子节点中较小的那个节点进行交换。



复制最后一个节点到根节点

获取最小值

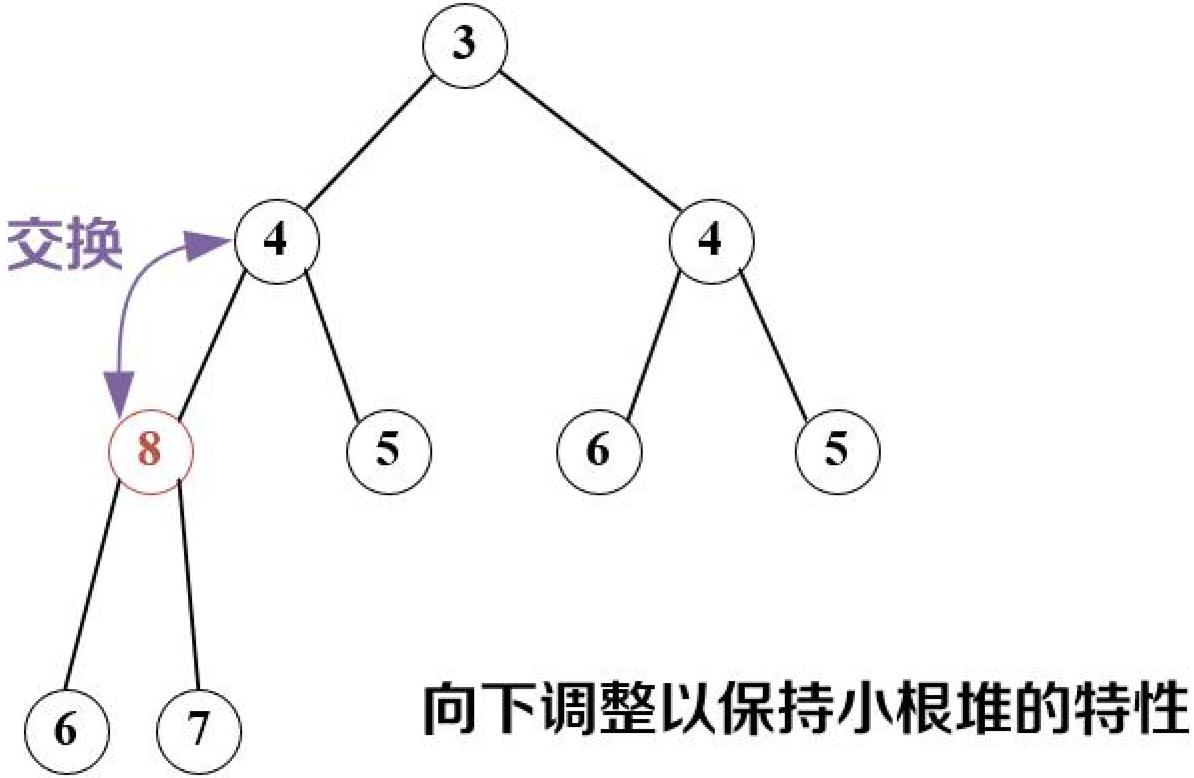
- 将最后一个节点复制到根节点（即覆盖存储在根节点处的最小元素）
- 通过向下调整（或向下冒泡）恢复小根堆的特性：如果该节点的值比它任何一个子节点的值都大，那么就将它和子节点中较小的那个节点进行交换。



向下调整以保持小根堆的特性

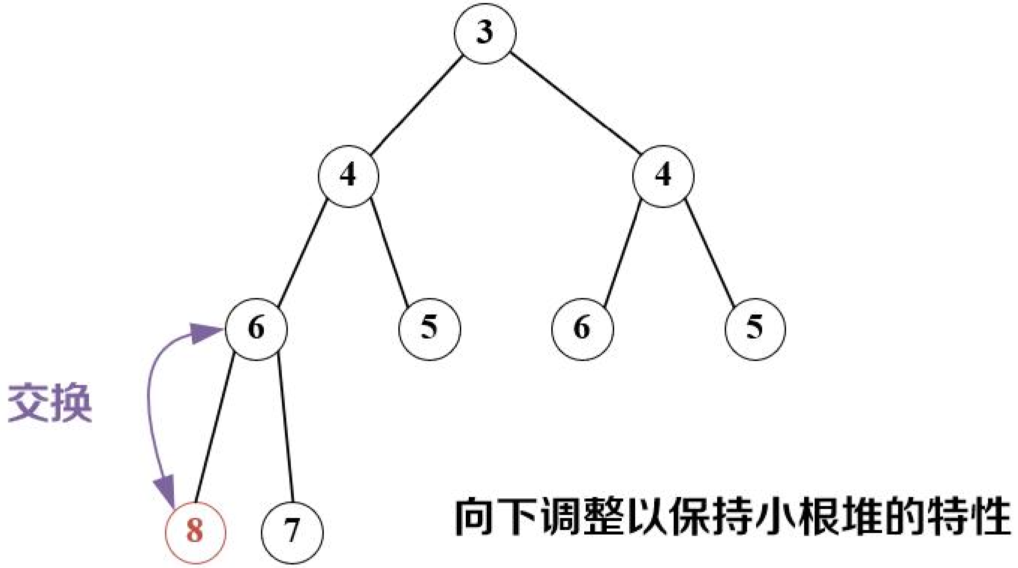
获取最小值

- 将最后一个节点复制到根节点（即覆盖存储在根节点处的最小元素）
- 通过向下调整（或向下冒泡）恢复小根堆的特性：如果该节点的值比它任何一个子节点的值都大，那么就将它和子节点中较小的那个节点进行交换。



获取最小值

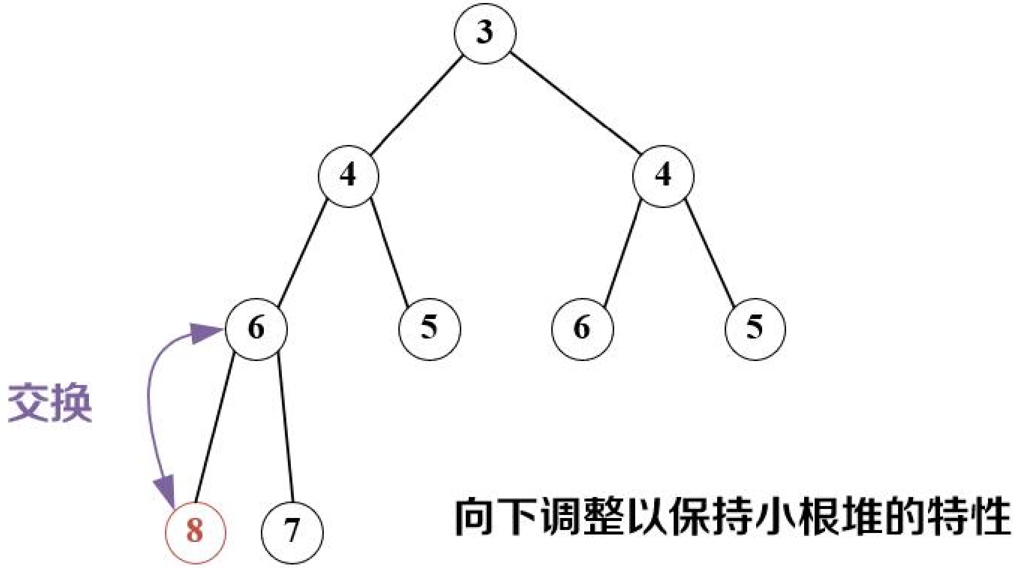
- 将最后一个节点复制到根节点（即覆盖存储在根节点处的最小元素）
- 通过向下调整（或向下冒泡）恢复小根堆的特性：如果该节点的值比它任何一个子节点的值都大，那么就将它和子节点中较小的那个节点进行交换。



- 正确性：在每次交换之后，除了包含该元素的节点之外，所有的节点都满足最小堆属性（相对于其子节点而言）

获取最小值

- 将最后一个节点复制到根节点（即覆盖存储在根节点处的最小元素）
- 通过向下调整（或向下冒泡）恢复小根堆的特性：如果该节点的值比它任何一个子节点的值都大，那么就将它和子节点中较小的那个节点进行交换。



- 正确性：在每次交换之后，除了包含该元素的节点之外，所有的节点都满足最小堆属性（相对于其子节点而言）
- 时间复杂度 $= O(height) = O(\log n)$

提纲

- 优先队列
- （二叉）堆
- 堆排序
- 排序算法的下界
- 计数排序

堆排序

- 核心思想
  - 第i趟排序将序列的前n-i+1个元素组成的子序列 **转换** 为一个堆积，然后将堆的第一个元素与堆的最后那个元素交换位置。

- 排序步骤

建初始堆积

  - 1. 将原始序列**转换**为第一个堆。
  - 2. 将堆的第一个元素与堆积的最后那个元素交换位置。(即“去掉”最大值元素）
  - 3. 将“去掉”最大值元素后剩下的元素组成的子序列重新**转换**一个新的堆。
  - 4. 重复上述过程的第2至第3步n-1次。

堆排序

- 建立一个有$\boldsymbol{n}$个节点的二叉堆
  - 最小的节点在堆的顶部

堆排序

- 建立一个有$\boldsymbol{n}$个节点的二叉堆
  - 最小的节点在堆的顶部
- 执行$\boldsymbol{n}$次**Extract-Min**操作
  - 节点是按顺序提取的

堆排序

- 建立一个有$\boldsymbol{n}$个节点的二叉堆
  - 最小的节点在堆的顶部
  - 逐一插入$\boldsymbol{n}$个元素 $\rightarrow \boldsymbol{O(n\ log\ n)}$
- 执行$\boldsymbol{n}$次**Extract-Min**操作
  - 节点是按顺序提取的

堆排序

- 建立一个有$\boldsymbol{n}$个节点的二叉堆
  - 最小的节点在堆的顶部
  - 逐一插入$\boldsymbol{n}$个元素 $\rightarrow \boldsymbol{O(n\log n)}$

- 执行$\boldsymbol{n}$次**Extract-Min**操作
  - 节点是按顺序提取的
  - 每次**Extract-Min**操作的时间复杂度为$\boldsymbol{O(\log n)} \rightarrow$ n次为$\boldsymbol{O(n\log n)}$

堆排序

- 建立一个有$\boldsymbol{n}$个节点的二叉堆
  - 最小的节点在堆的顶部
  - 逐一插入$\boldsymbol{n}$个元素 $\rightarrow \boldsymbol{O(n\ log\ n)}$
- 执行$\boldsymbol{n}$次**Extract-Min**操作
  - 节点是按顺序提取的
  - 每次**Extract-Min**操作的时间复杂度为$\boldsymbol{O(log\ n)}$ → **n**次为$\boldsymbol{O(n\ log\ n)}$
- 总的时间复杂度:$\boldsymbol{O(n\ log\ n)}$

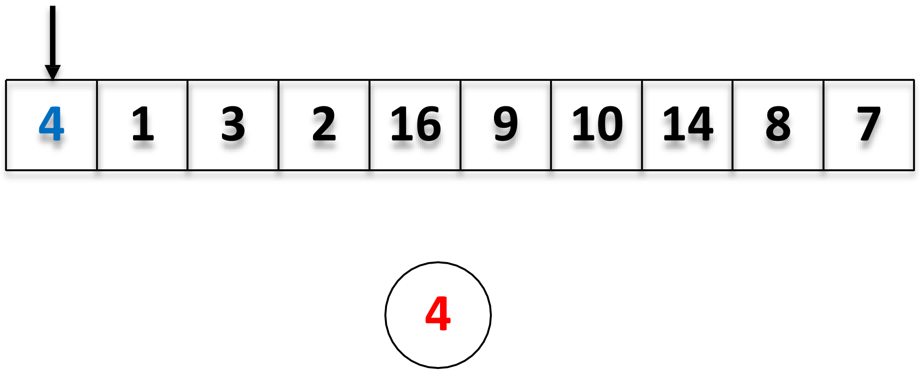
堆排序：算法实例

- 建立一个有n个节点的二叉堆

4	1	3	2	16	9	10	14	8	7

堆排序：算法实例

- 建立一个有n个节点的二叉堆



堆排序：算法实例

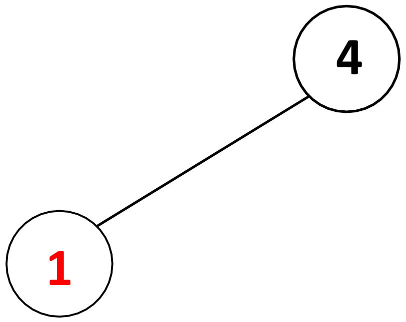
- 建立一个有n个节点的二叉堆

4	1	3	2	16	9	10	14	8	7

堆排序：算法实例

- 建立一个有n个节点的二叉堆

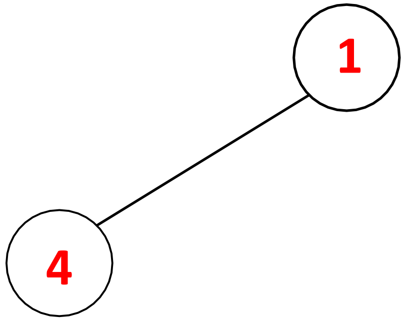
4	1	3	2	16	9	10	14	8	7



堆排序：算法实例

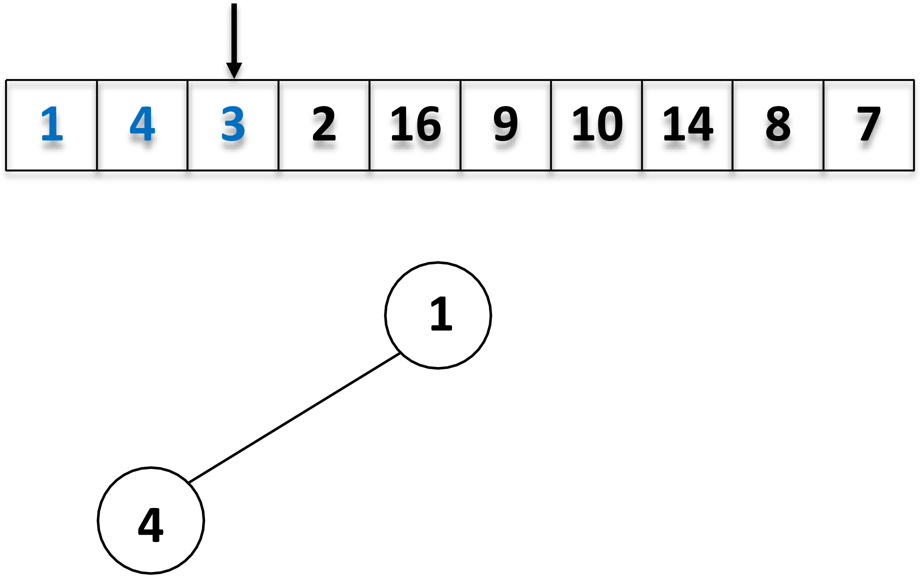
- 建立一个有n个节点的二叉堆

1	4	3	2	16	9	10	14	8	7



堆排序：算法实例

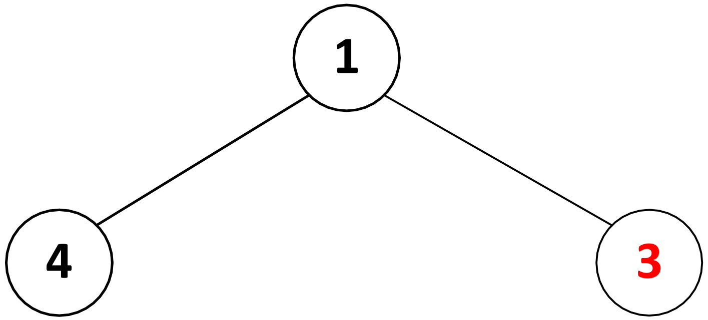
- 建立一个有n个节点的二叉堆



堆排序：算法实例

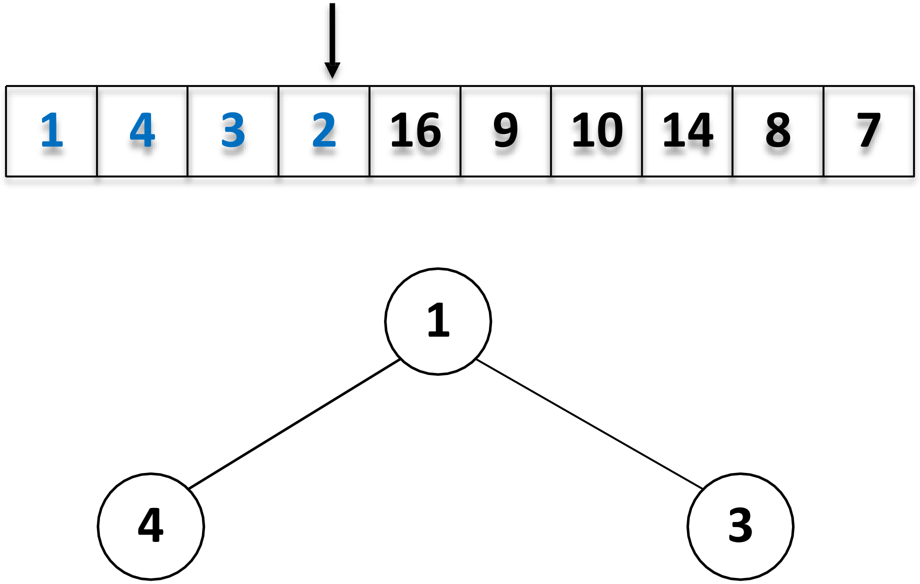
- 建立一个有n个节点的二叉堆

1	4	3	2	16	9	10	14	8	7



堆排序：算法实例

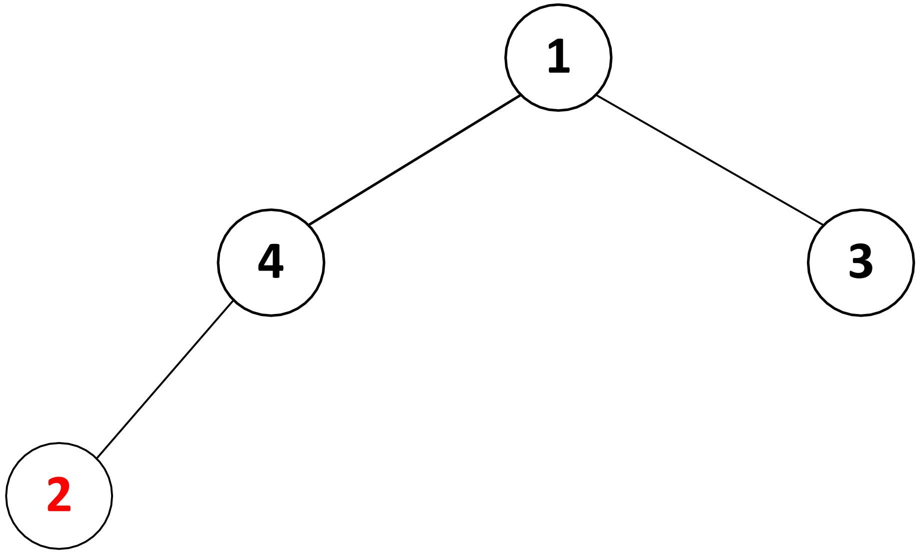
- 建立一个有n个节点的二叉堆



堆排序：算法实例

- 建立一个有n个节点的二叉堆

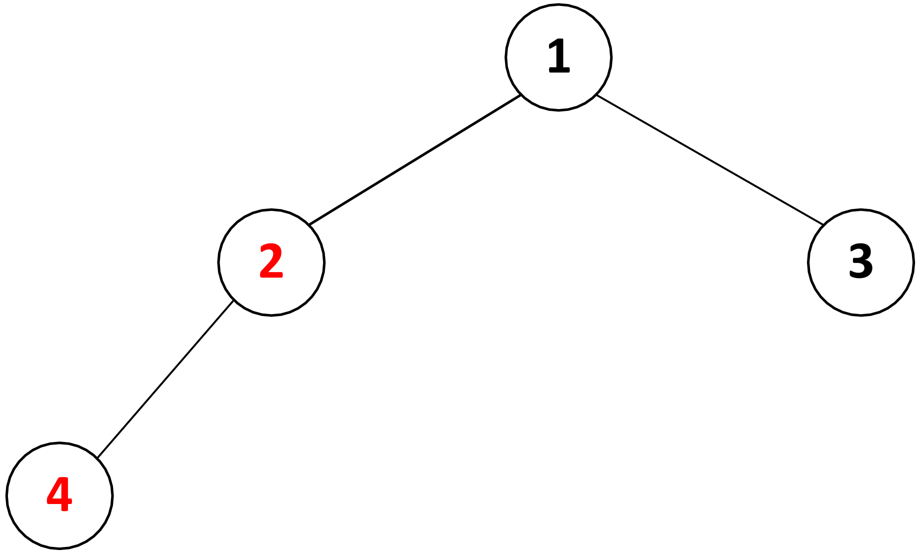
1	4	3	2	16	9	10	14	8	7



堆排序：算法实例

- 建立一个有n个节点的二叉堆

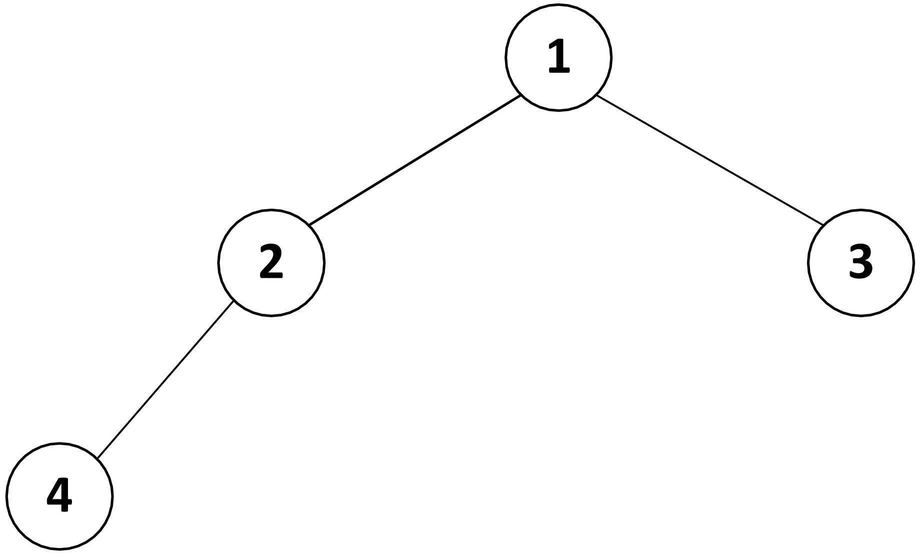
1	2	3	4	16	9	10	14	8	7



堆排序：算法实例

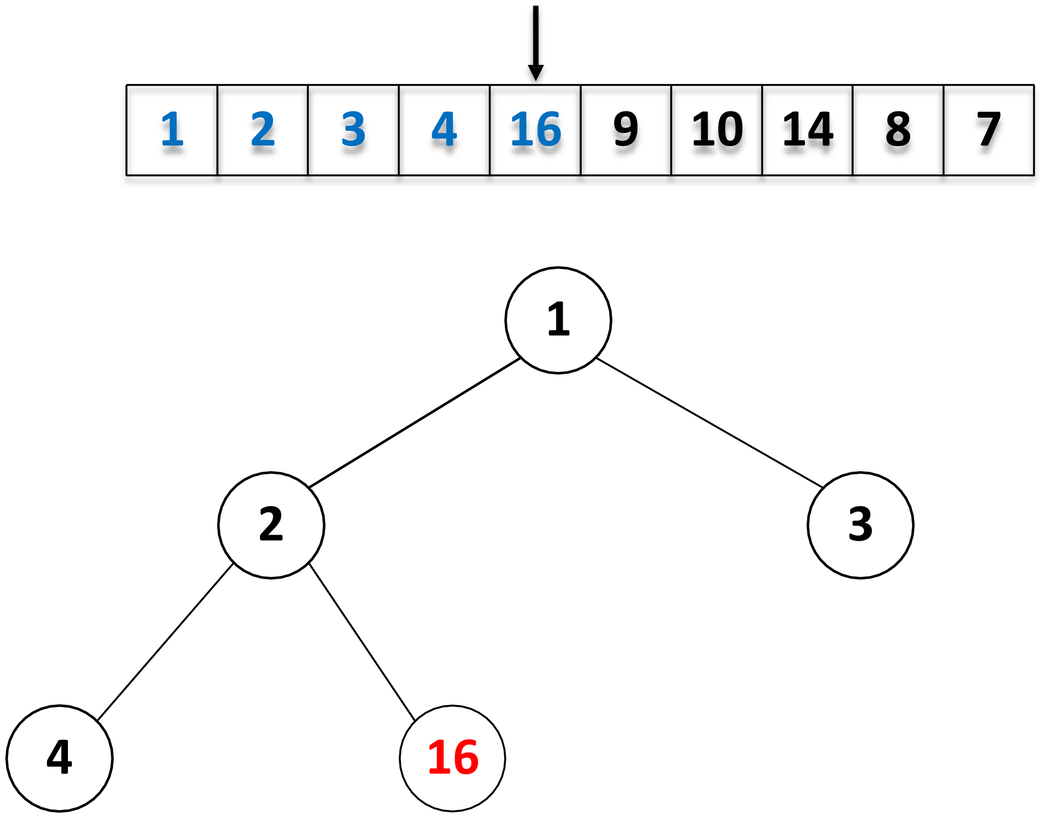
- 建立一个有n个节点的二叉堆

1	2	3	4	16	9	10	14	8	7



堆排序：算法实例

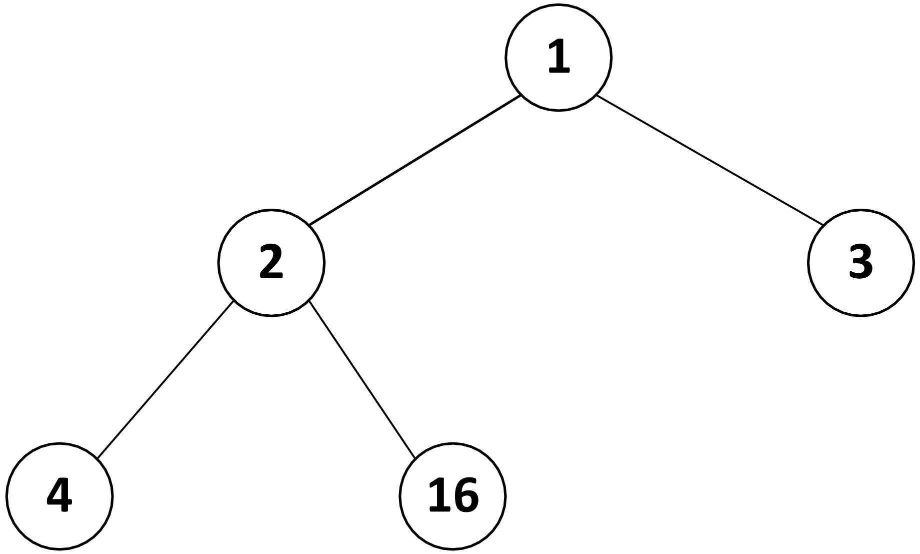
- 建立一个有n个节点的二叉堆



堆排序：算法实例

- 建立一个有n个节点的二叉堆

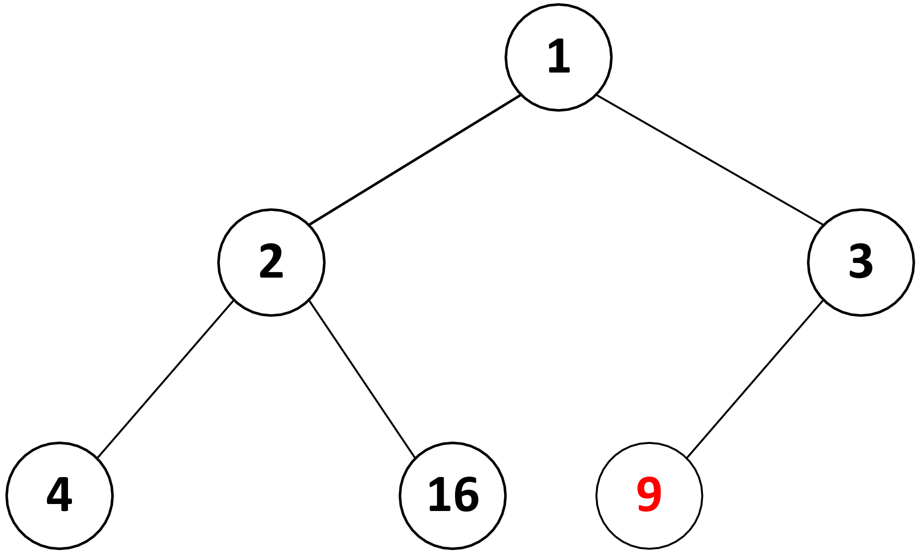
1	2	3	4	16	9	10	14	8	7



堆排序：算法实例

- 建立一个有n个节点的二叉堆

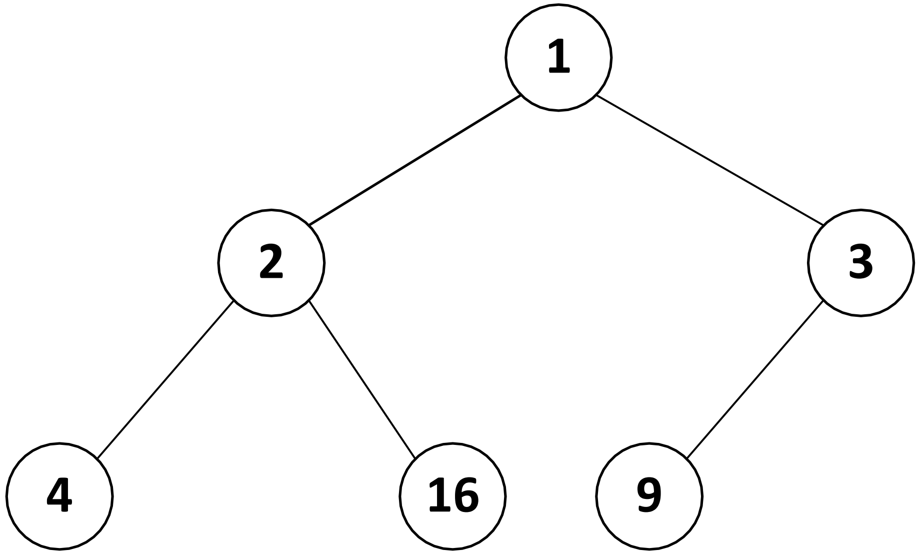
1	2	3	4	16	9	10	14	8	7



堆排序：算法实例

- 建立一个有n个节点的二叉堆

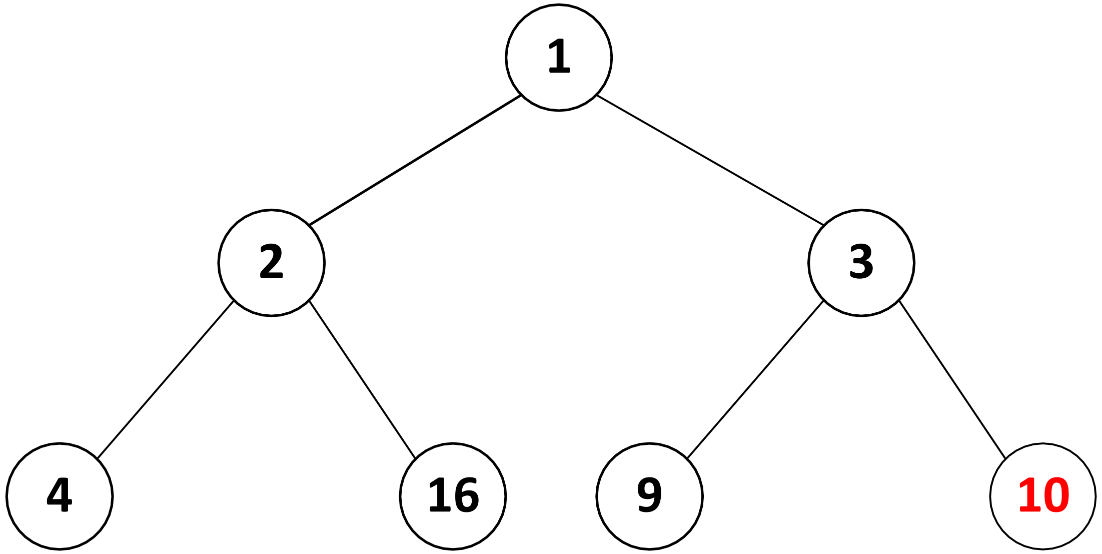
1	2	3	4	16	9	10	14	8	7



堆排序：算法实例

- 建立一个有n个节点的二叉堆

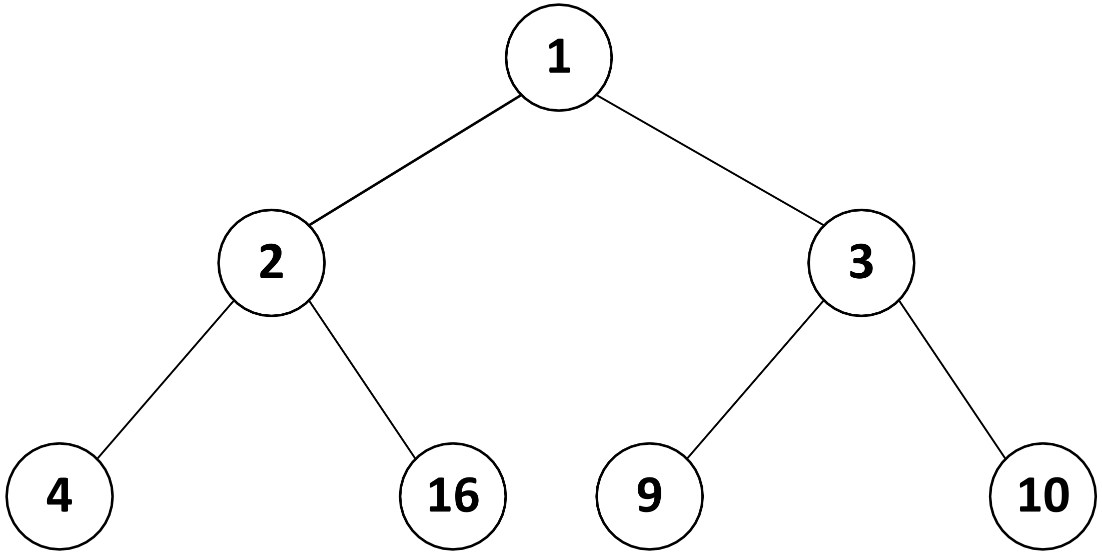
1	2	3	4	16	9	10	14	8	7



堆排序：算法实例

- 建立一个有n个节点的二叉堆

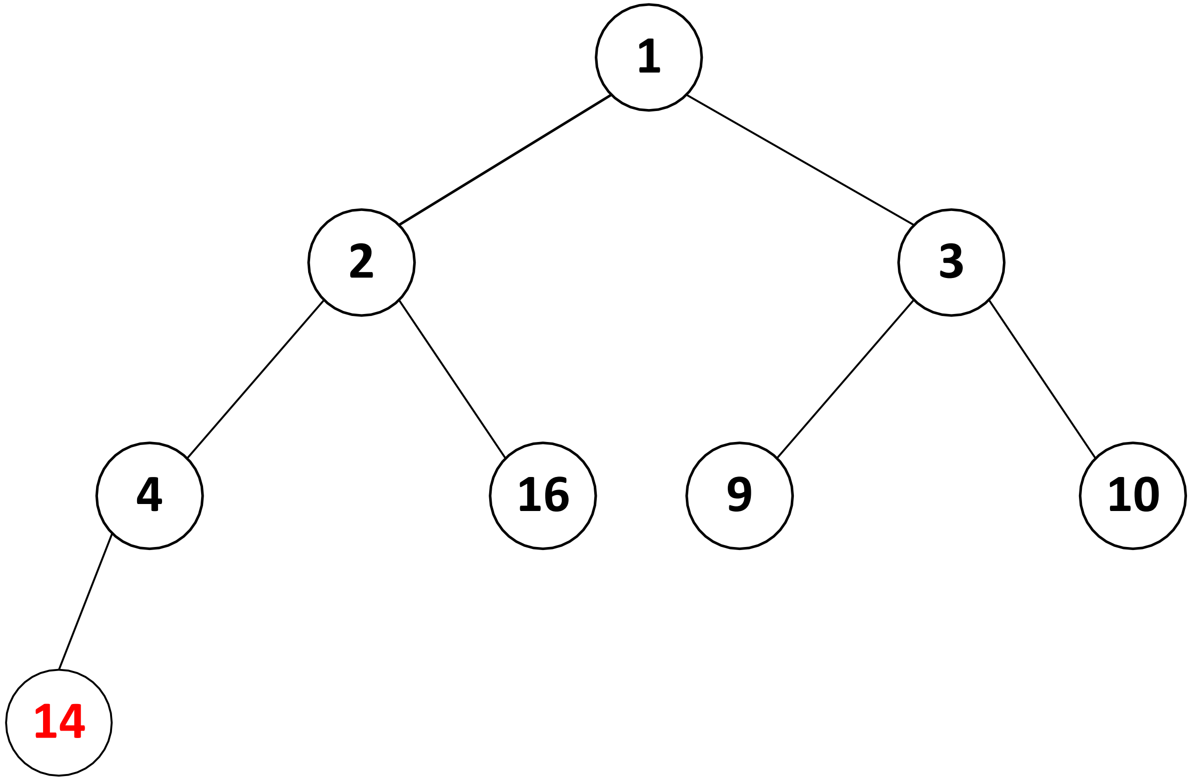
1	2	3	4	16	9	10	14	8	7



堆排序：算法实例

- 建立一个有n个节点的二叉堆

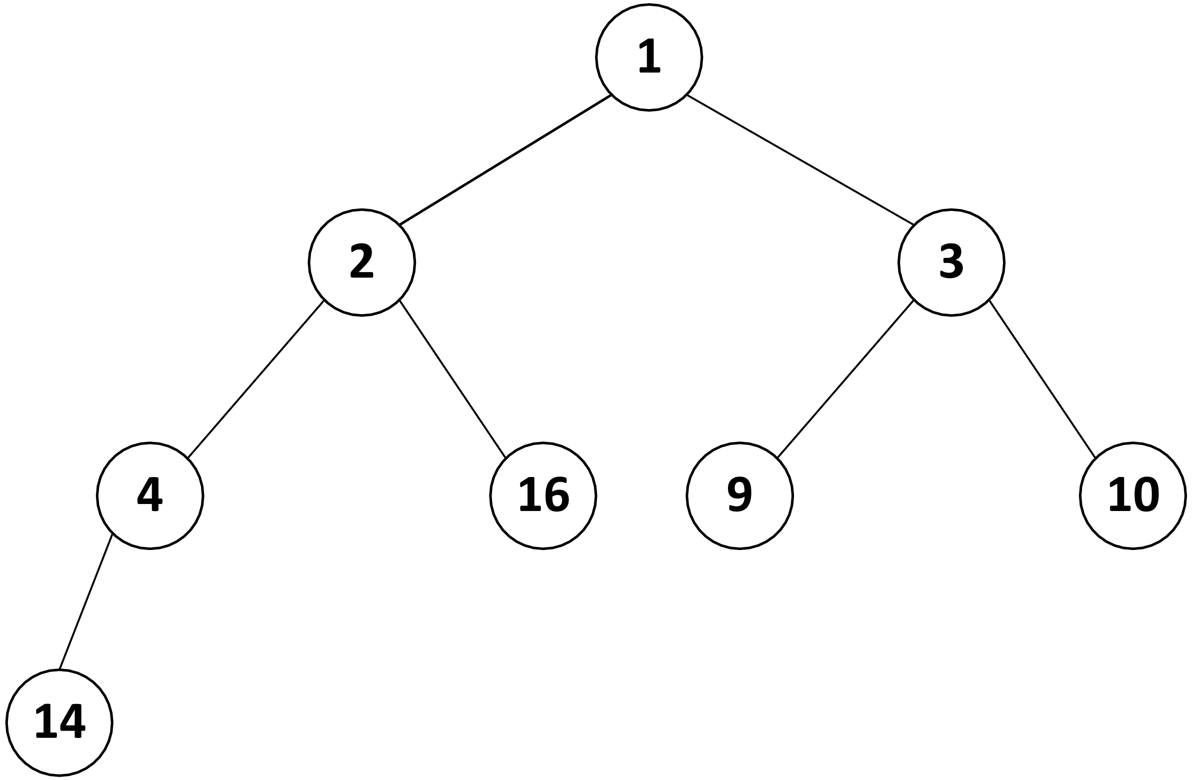
1	2	3	4	16	9	10	14	8	7



堆排序：算法实例

- 建立一个有n个节点的二叉堆

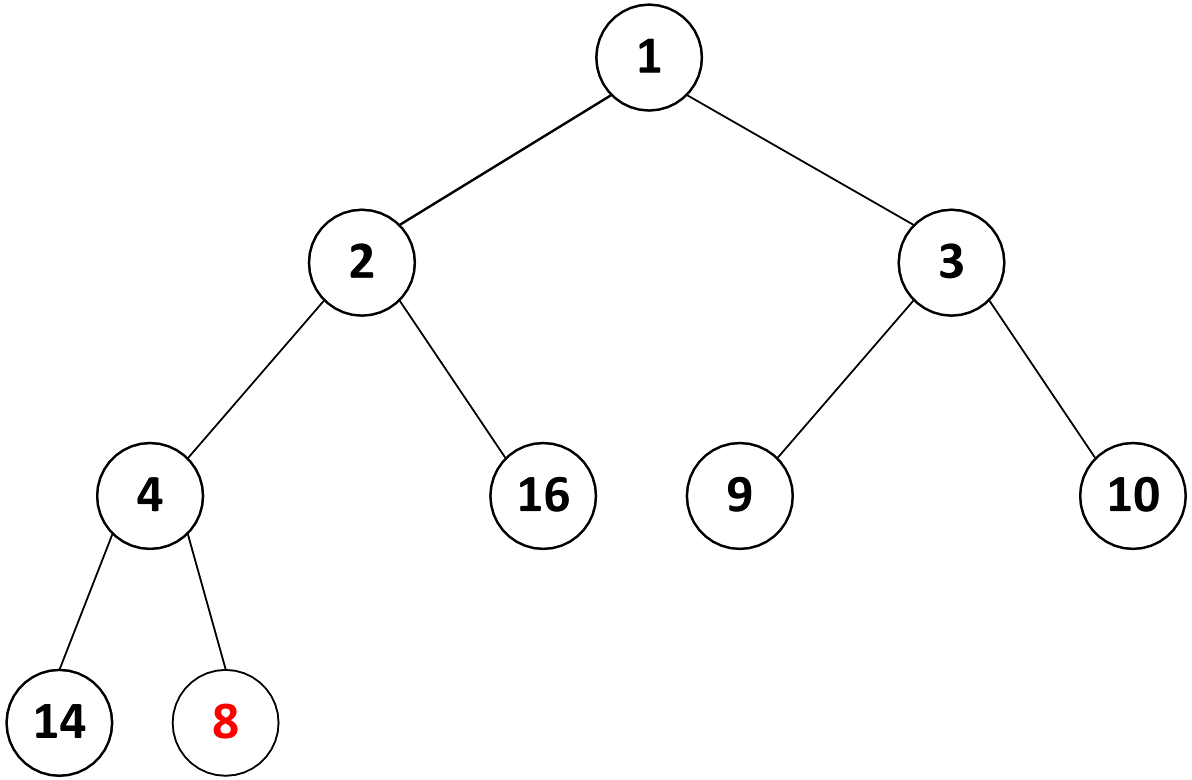
1	2	3	4	16	9	10	14	8	7



堆排序：算法实例

- 建立一个有n个节点的二叉堆

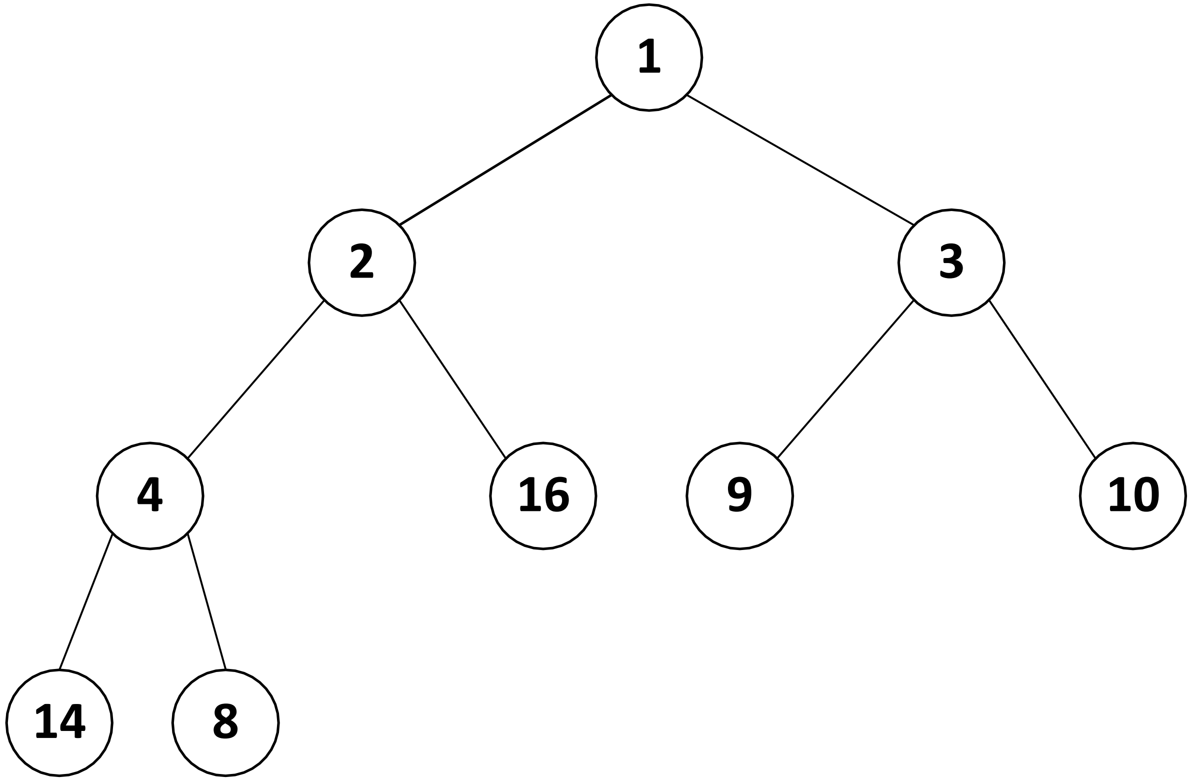
1	2	3	4	16	9	10	14	8	7



堆排序：算法实例

- 建立一个有n个节点的二叉堆

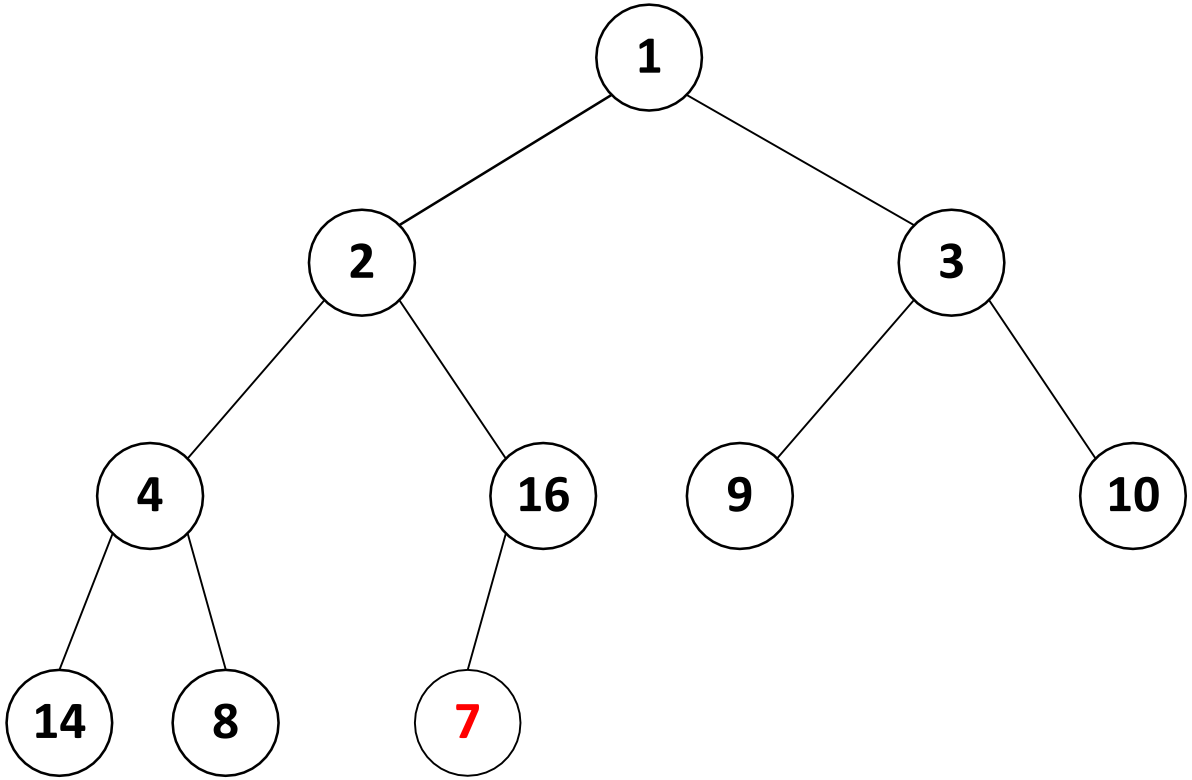
1	2	3	4	16	9	10	14	8	7



堆排序：算法实例

- 建立一个有n个节点的二叉堆

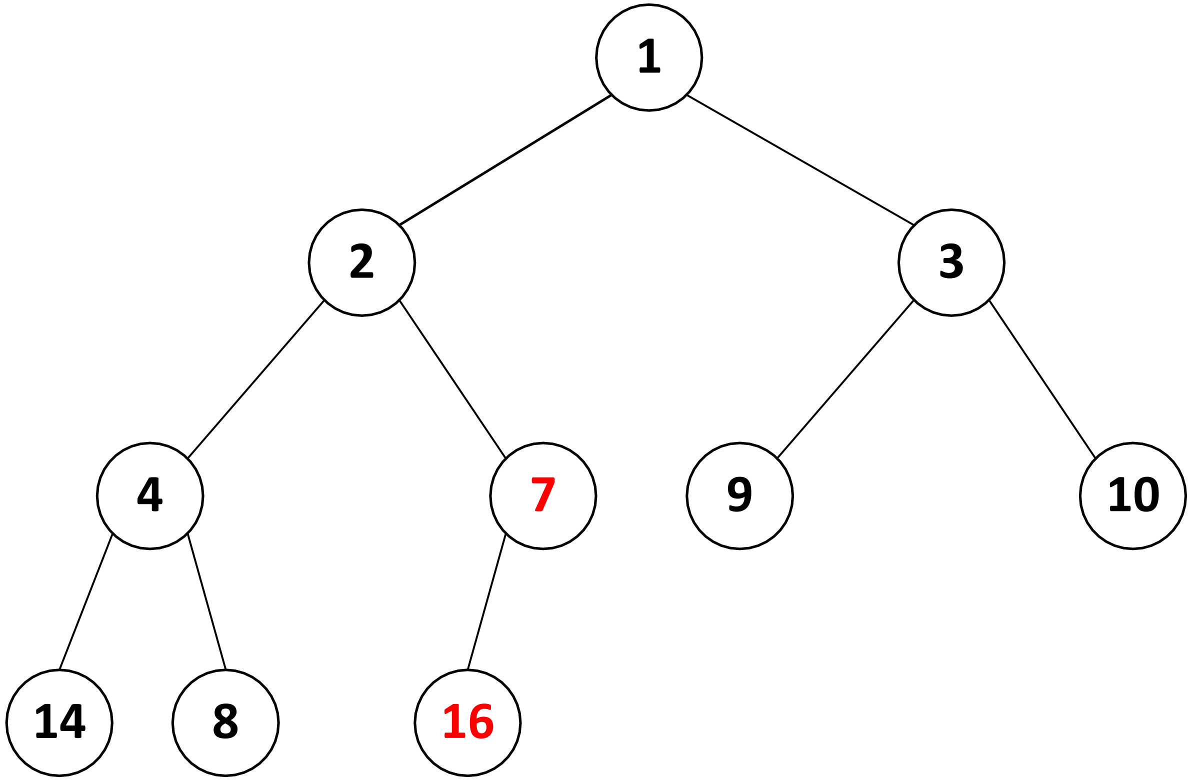
1	2	3	4	16	9	10	14	8	7



堆排序：算法实例

- 建立一个有n个节点的二叉堆

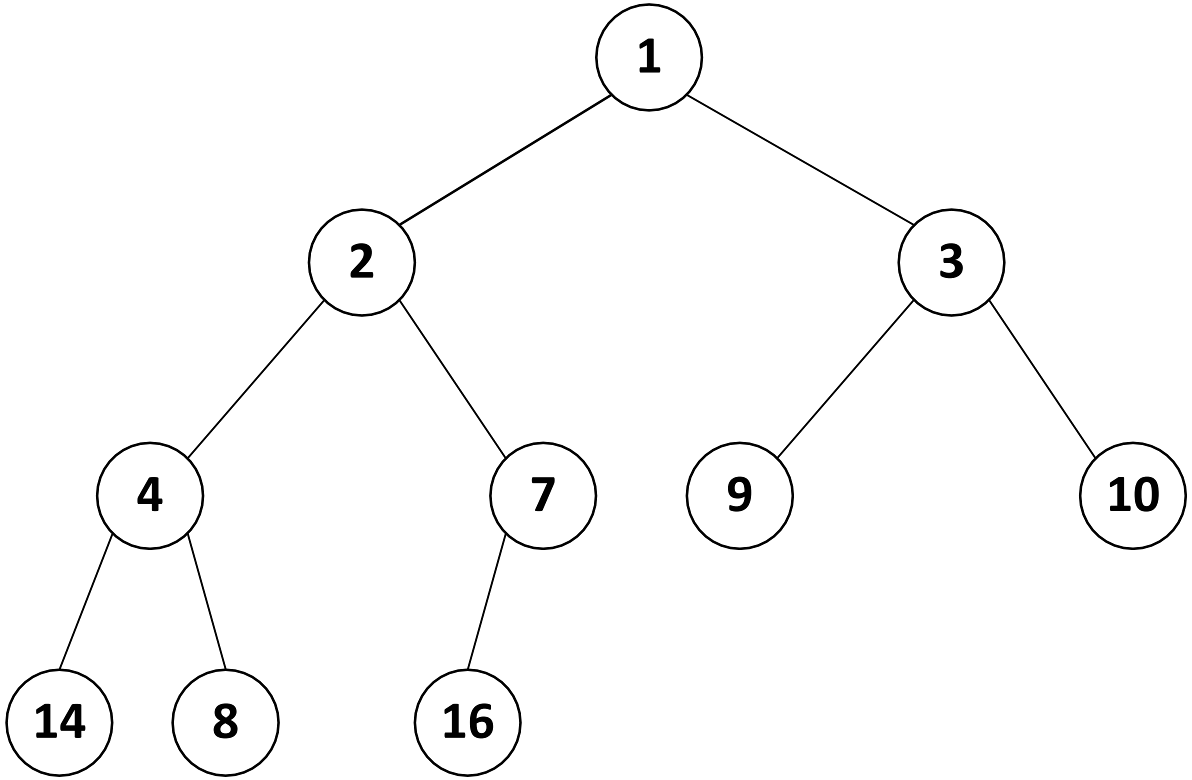
1	2	3	4	7	9	10	14	8	16



堆排序：算法实例

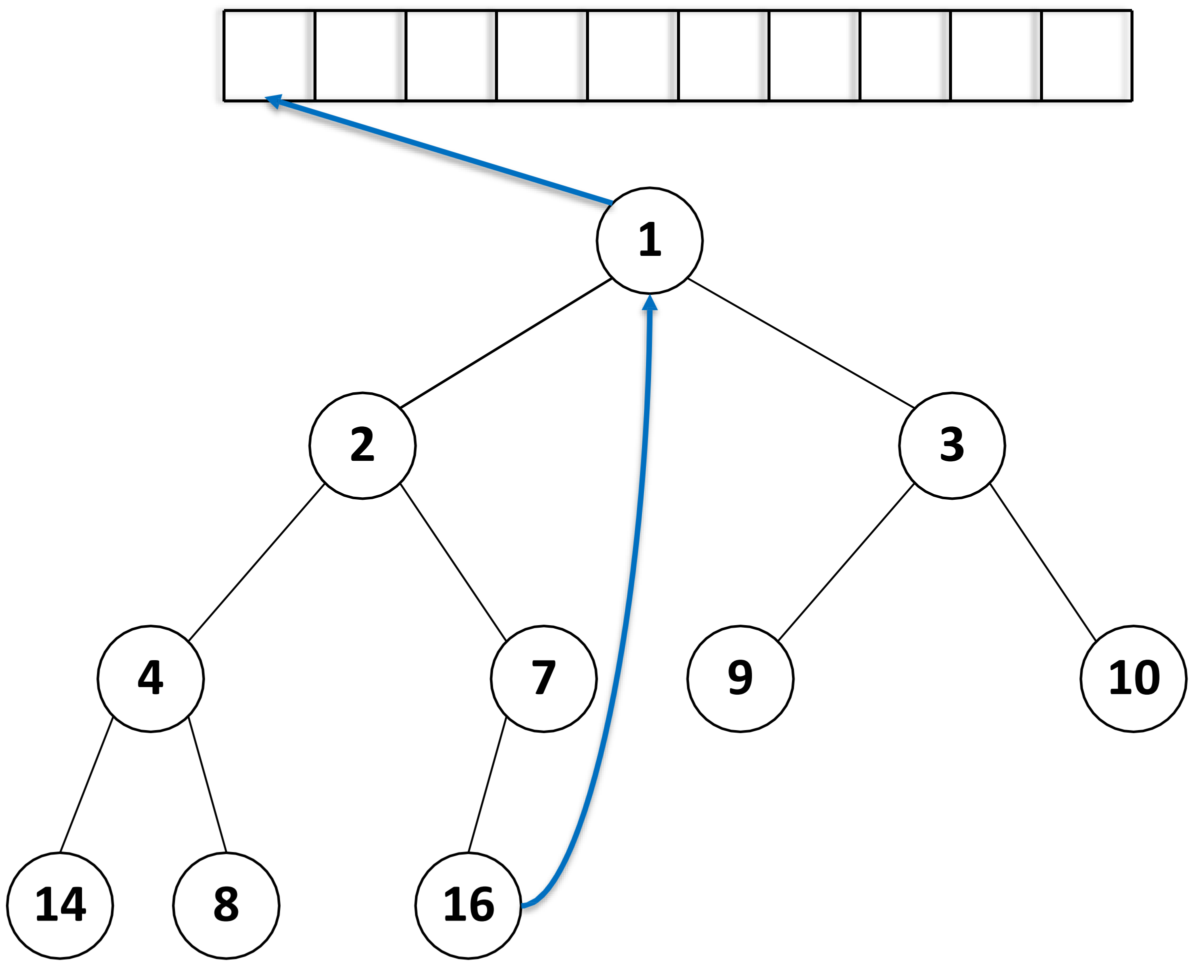
- 建立一个有n个节点的二叉堆

1	2	3	4	7	9	10	14	8	16



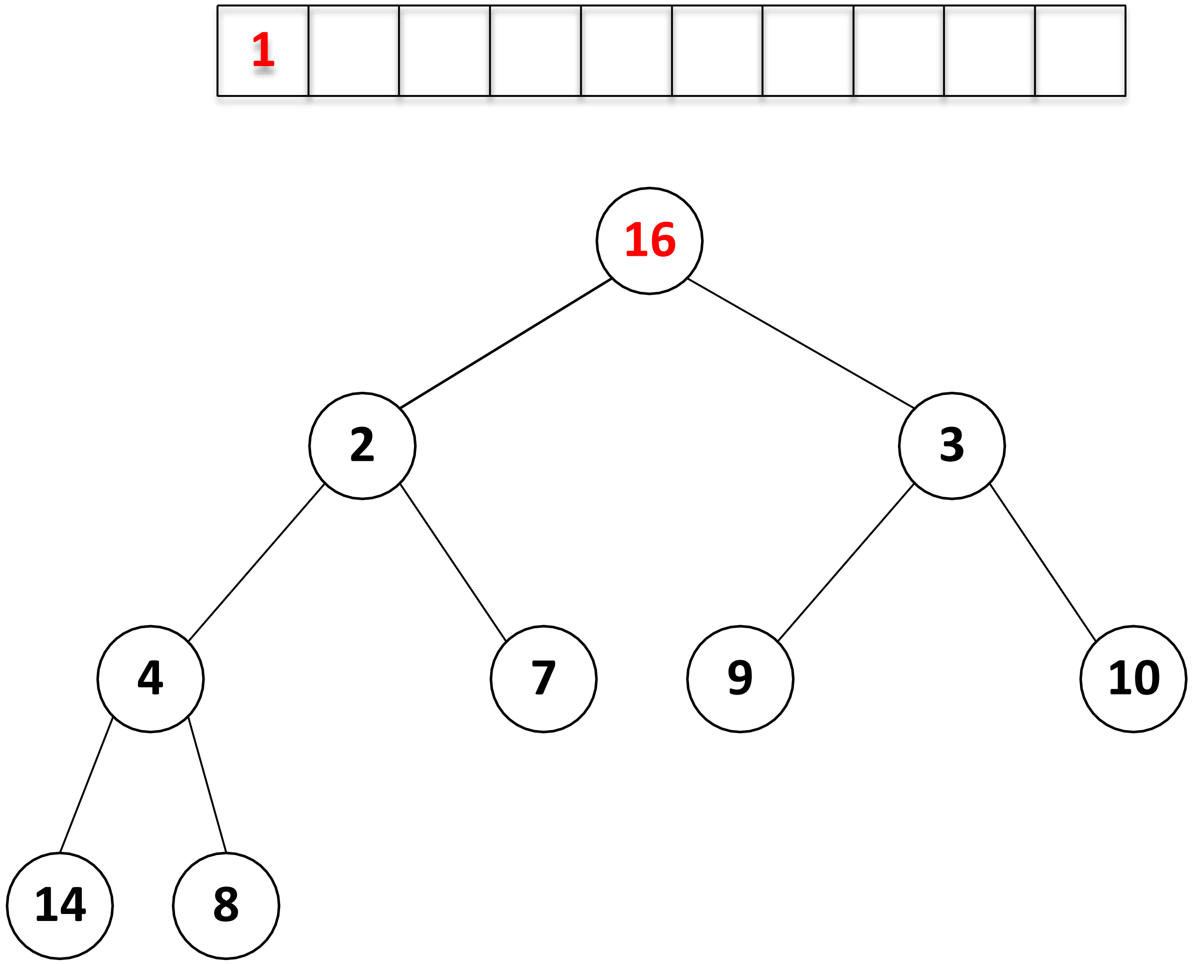
堆排序：算法实例

- 执行n次Extract-Min操作



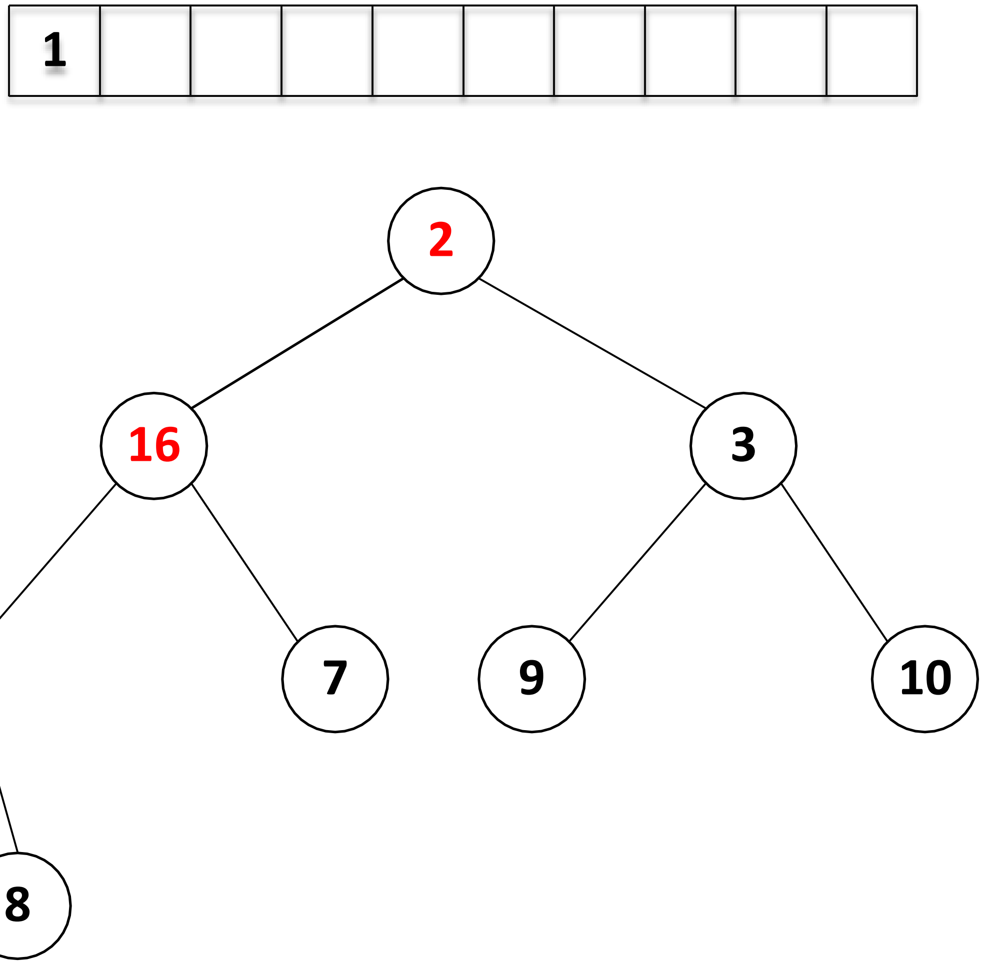
堆排序：算法实例

- 执行n次Extract-Min操作



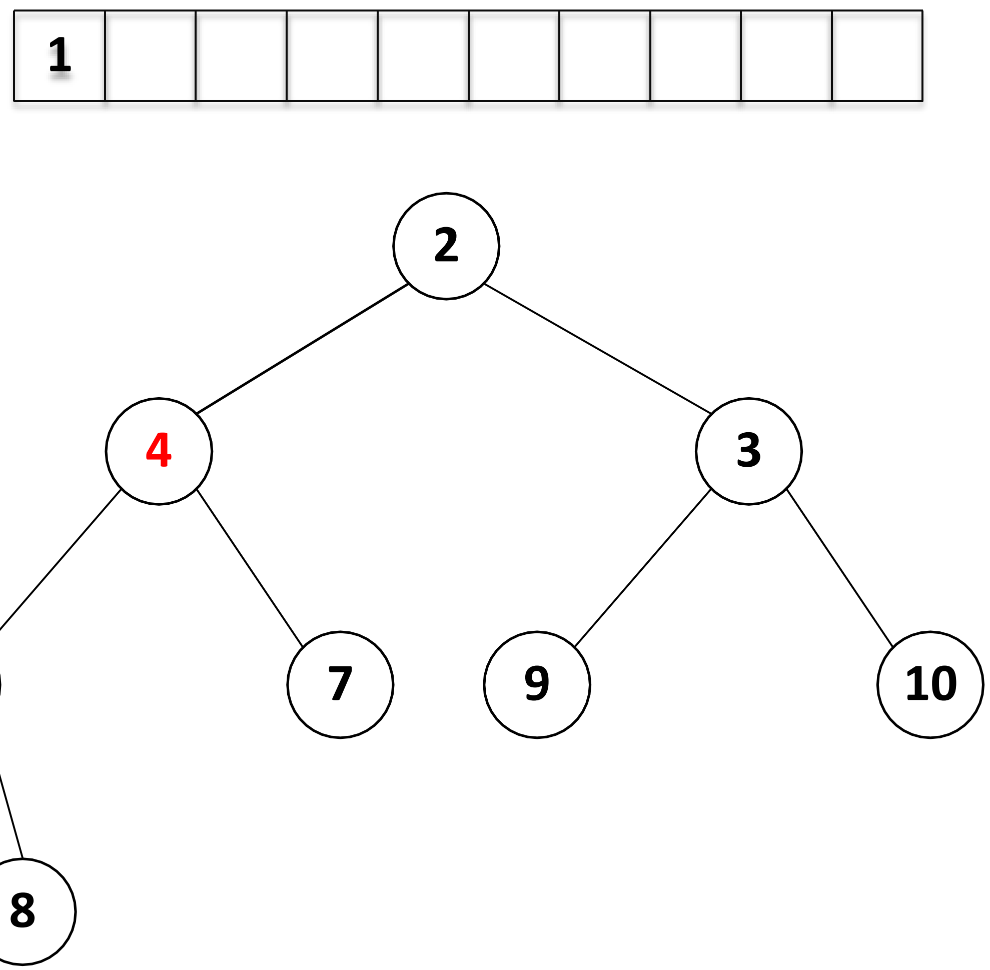
堆排序：算法实例

- 执行n次Extract-Min操作



堆排序：算法实例

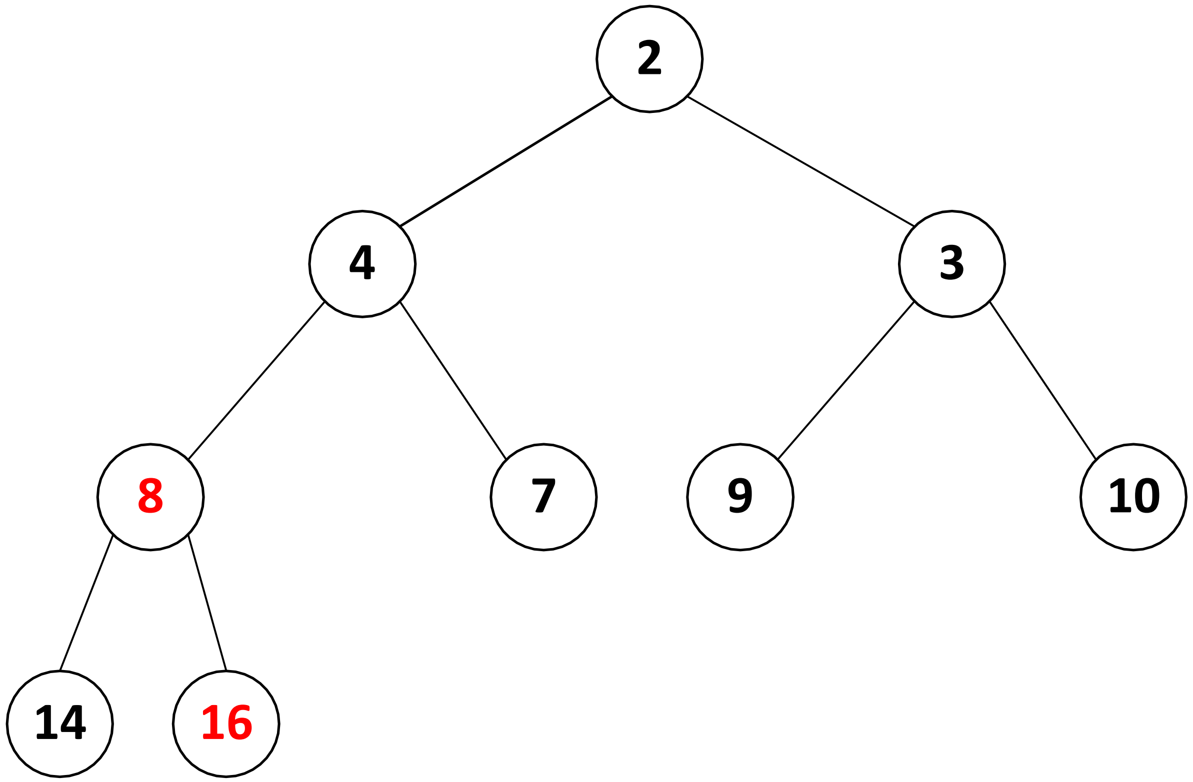
- 执行n次Extract-Min操作



堆排序：算法实例

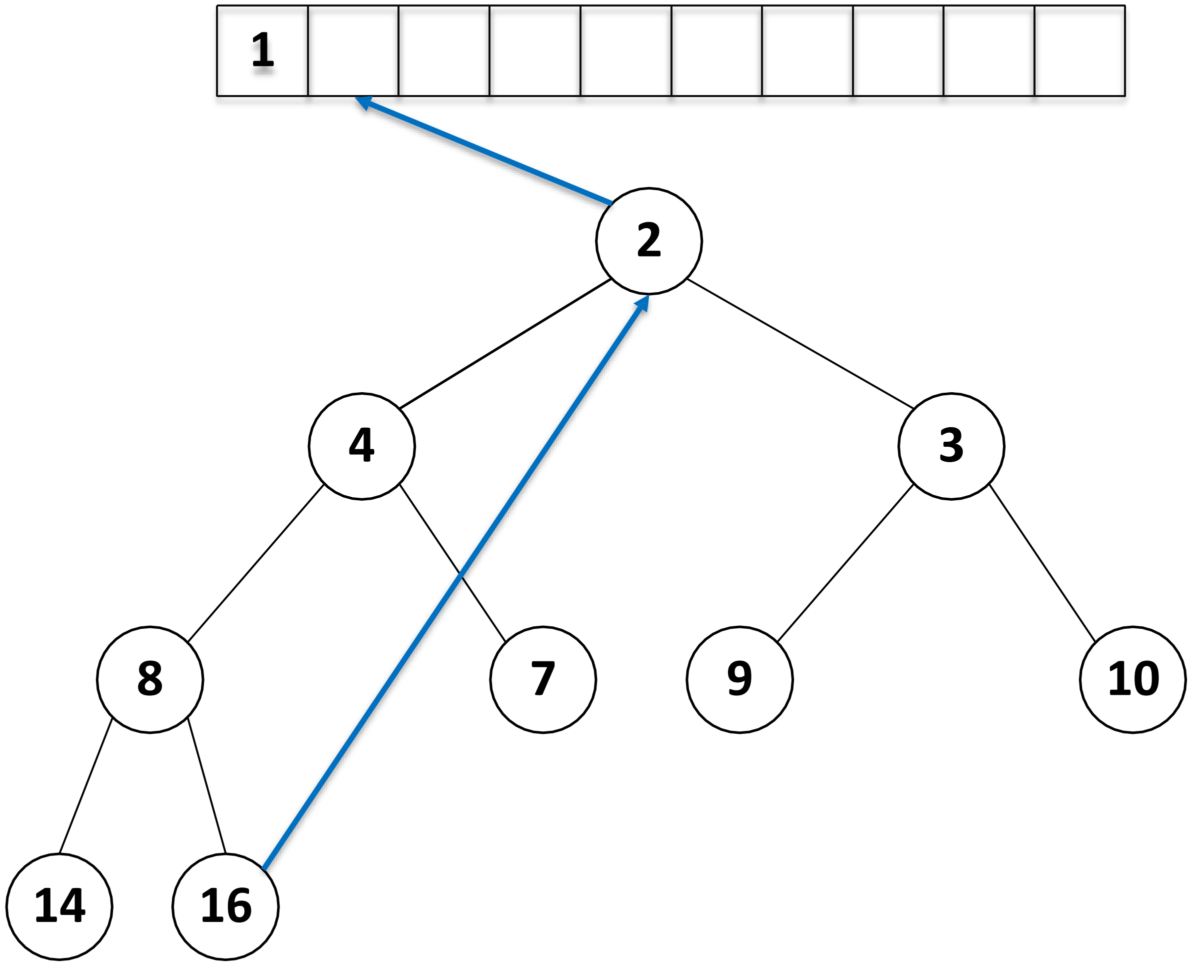
- 执行n次Extract-Min操作

1									



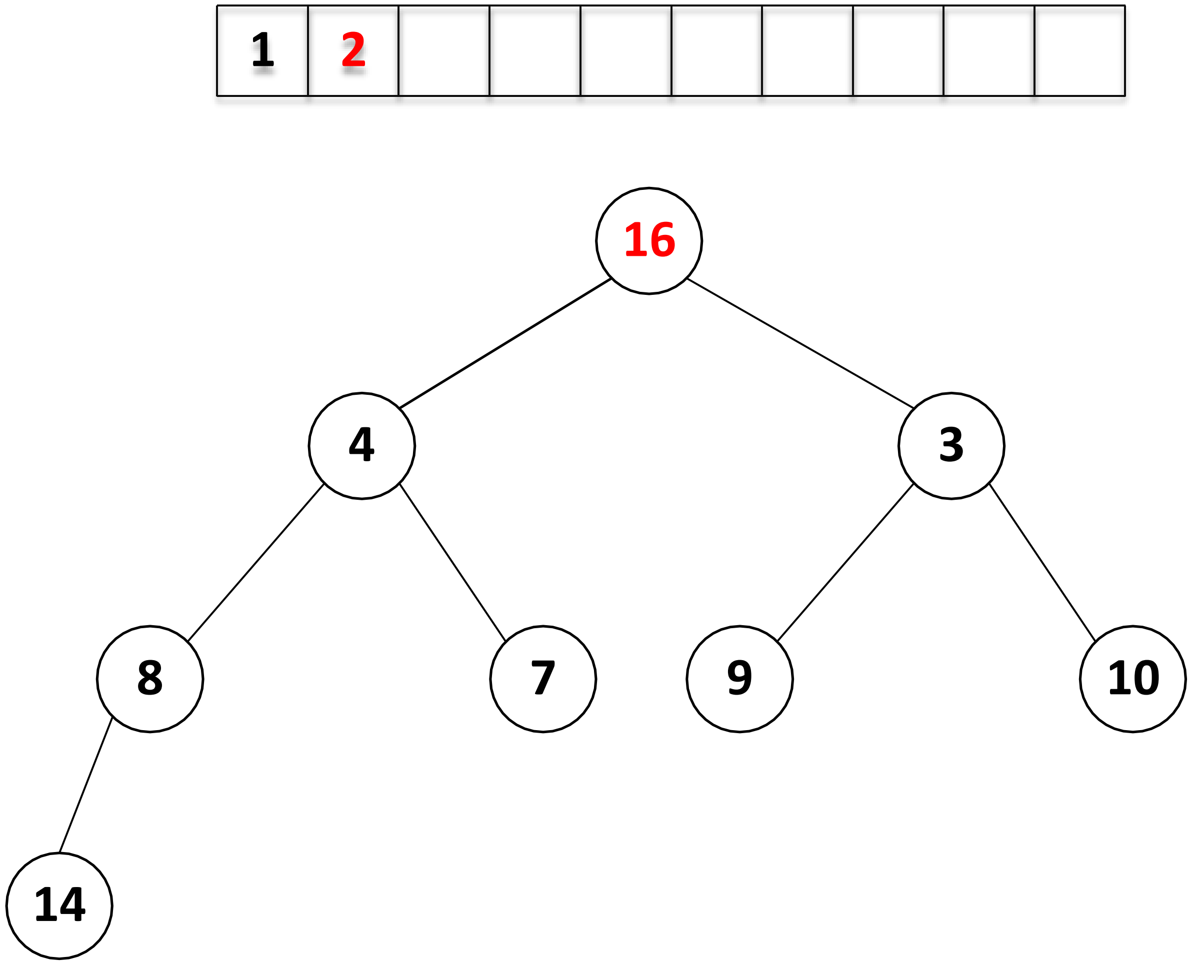
堆排序：算法实例

- 执行n次Extract-Min操作



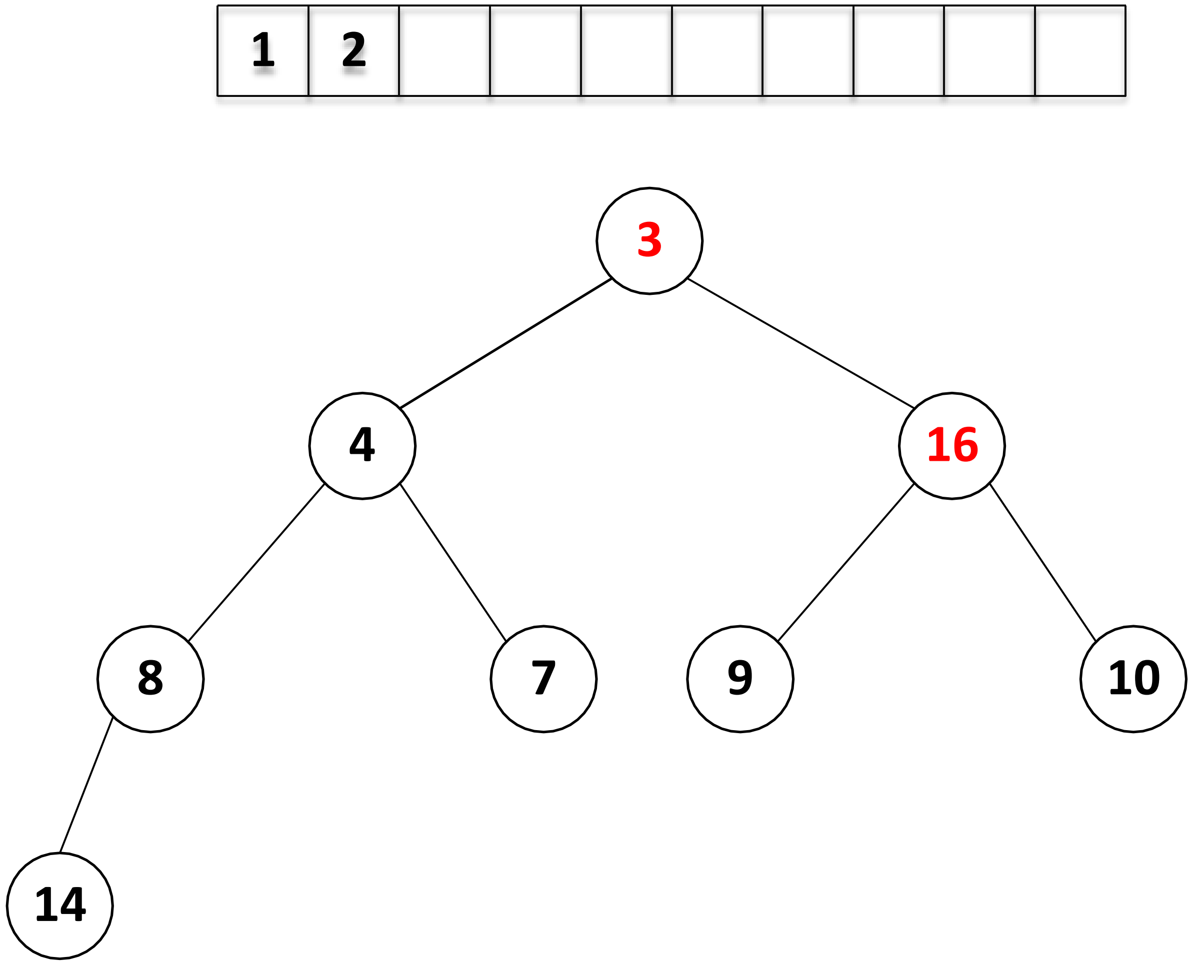
堆排序：算法实例

- 执行n次Extract-Min操作



堆排序：算法实例

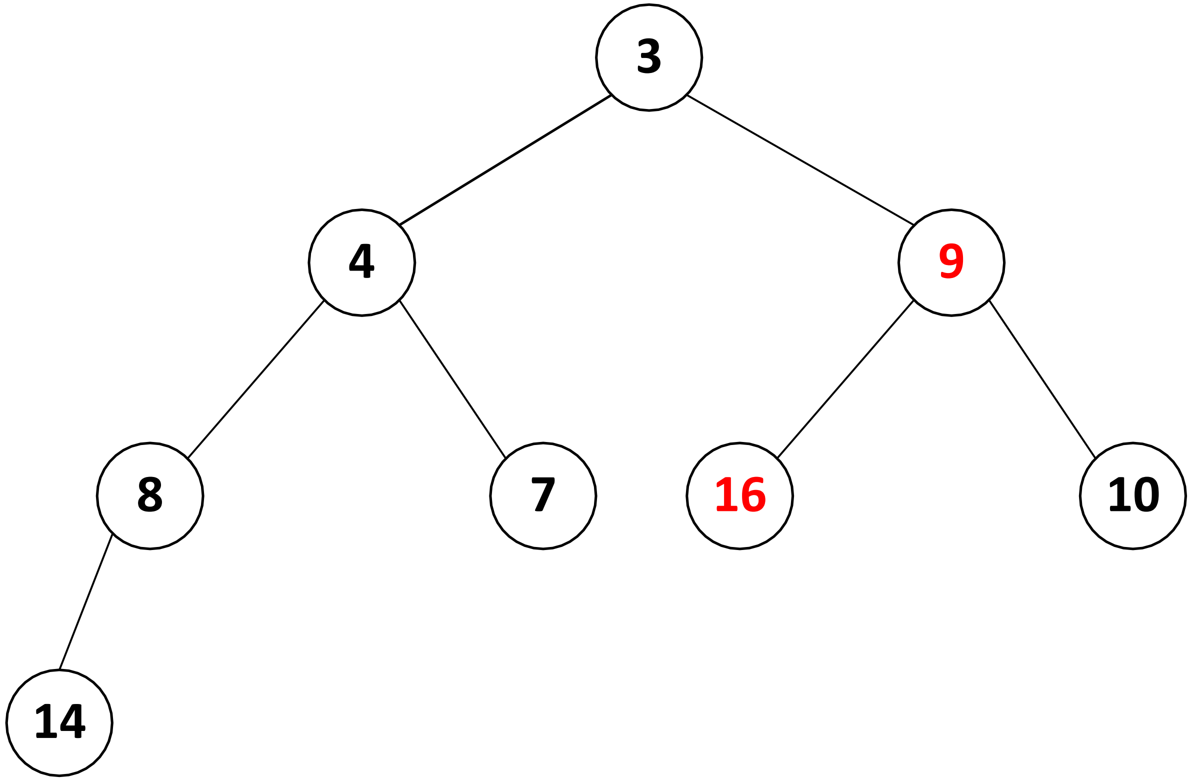
- 执行n次**Extract-Min**操作



堆排序：算法实例

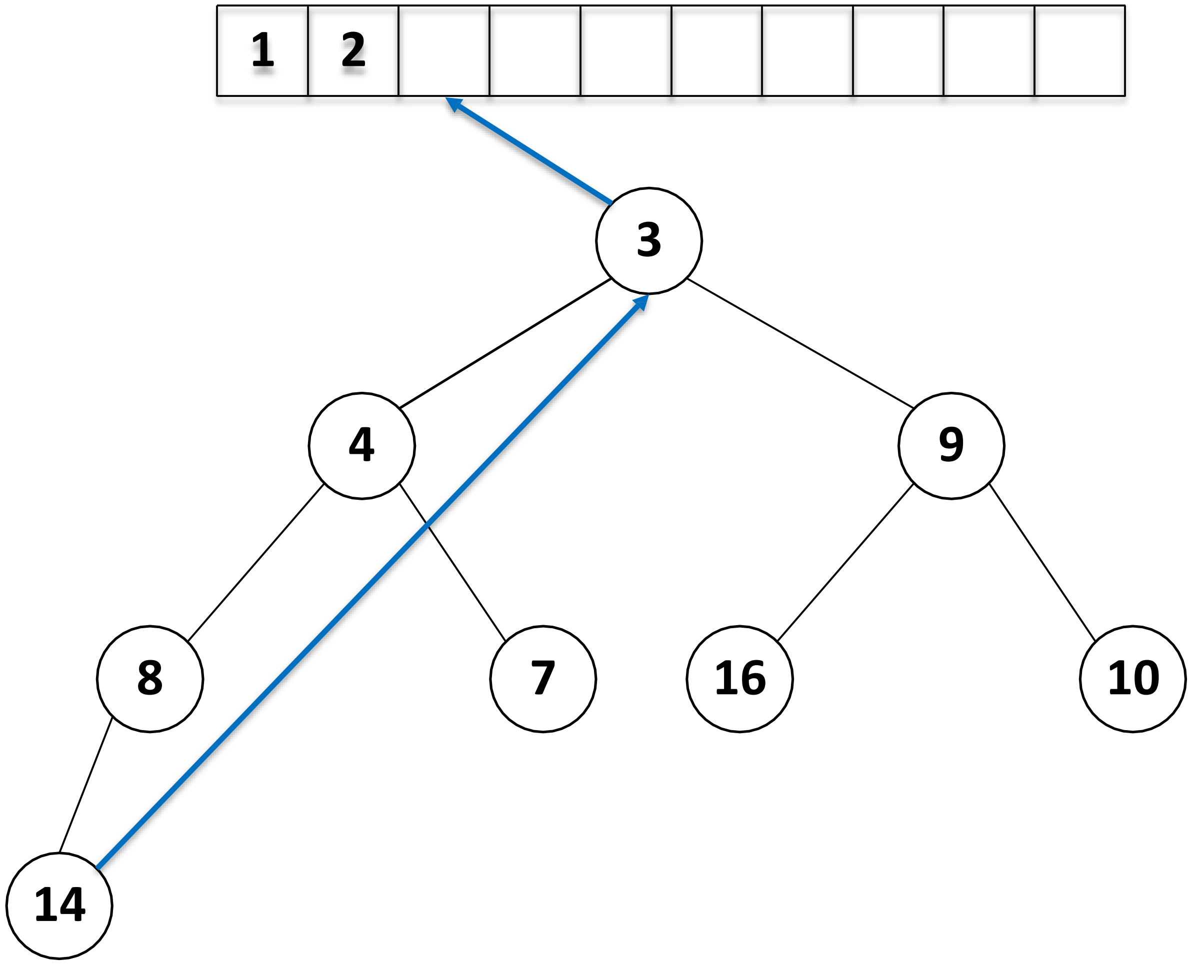
- 执行n次**Extract-Min**操作

1	2								



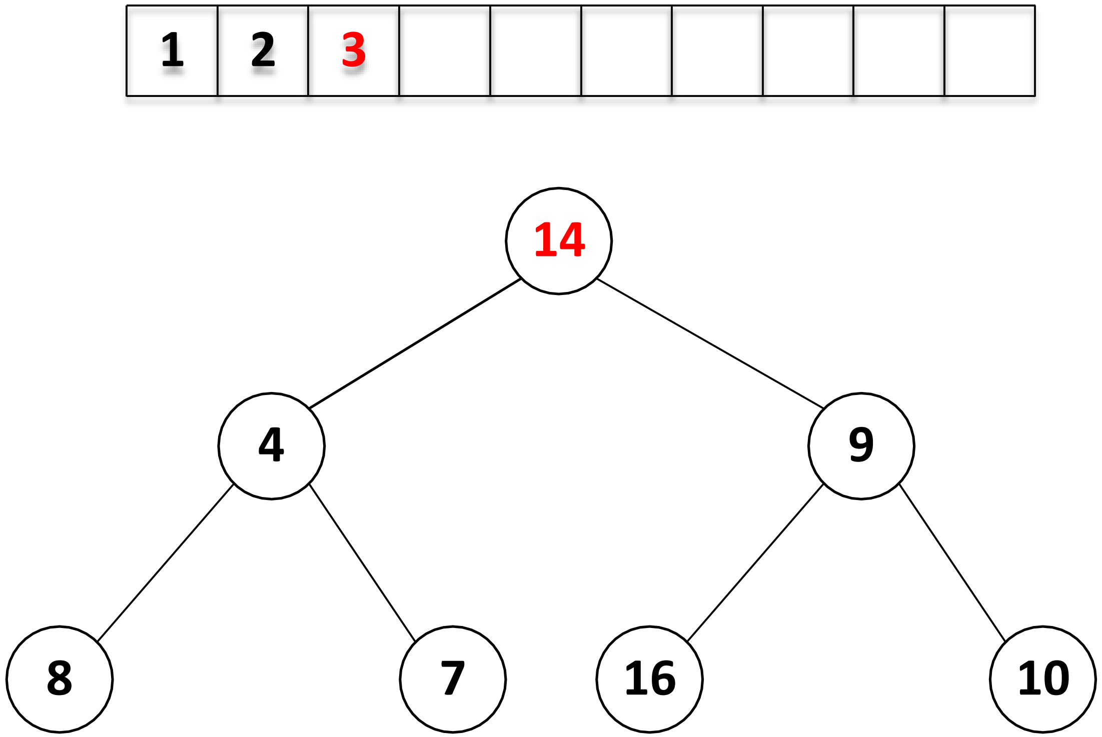
堆排序：算法实例

- 执行n次Extract-Min操作



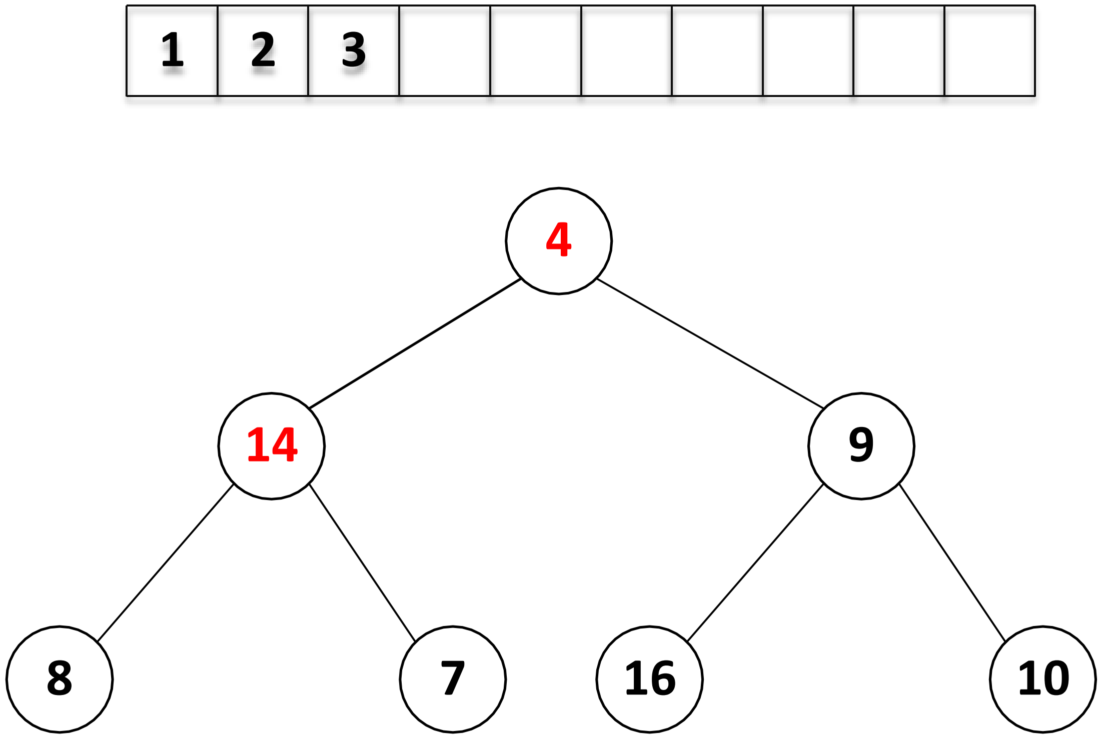
堆排序：算法实例

- 执行n次**Extract-Min**操作



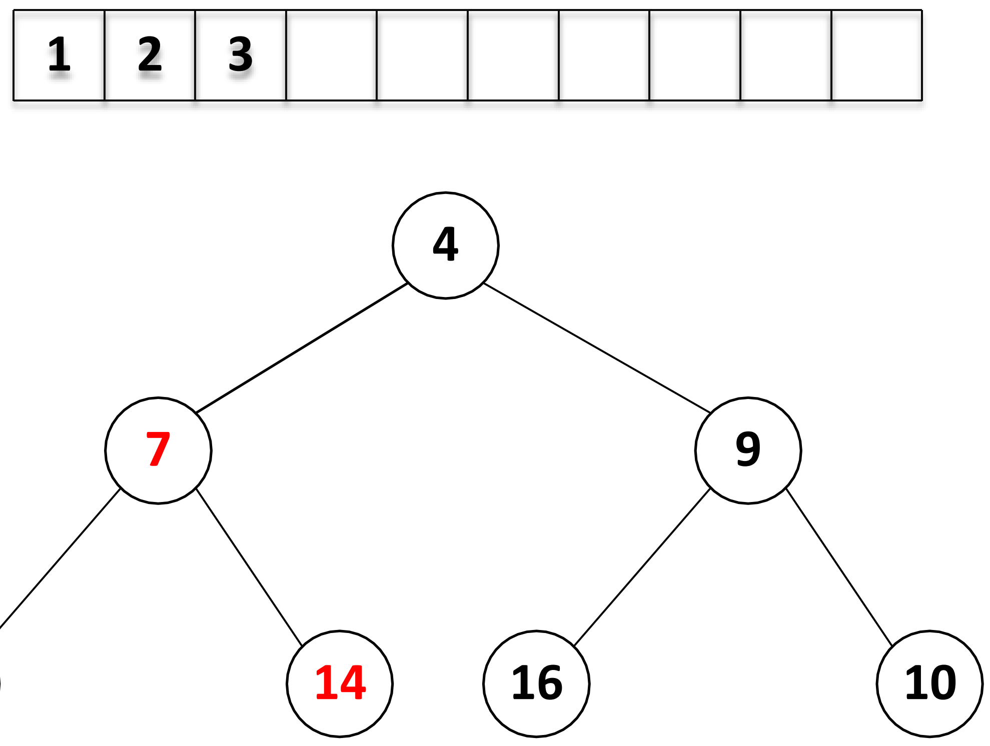
堆排序：算法实例

- 执行n次Extract-Min操作



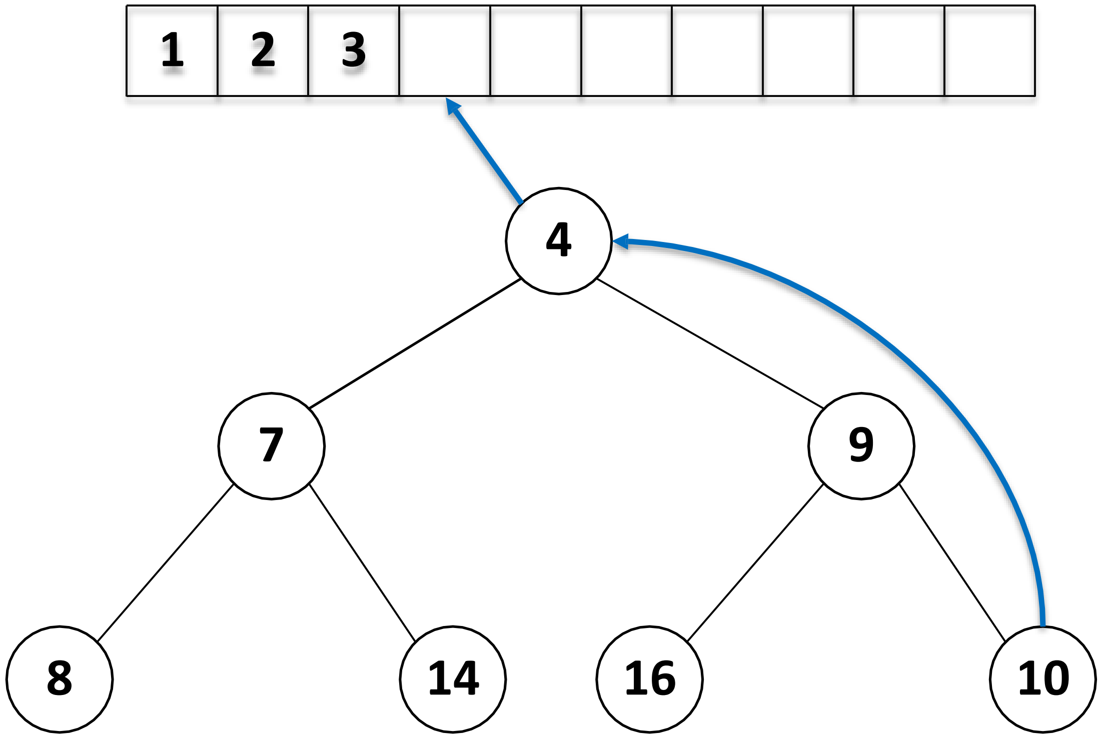
堆排序：算法实例

- 执行n次Extract-Min操作



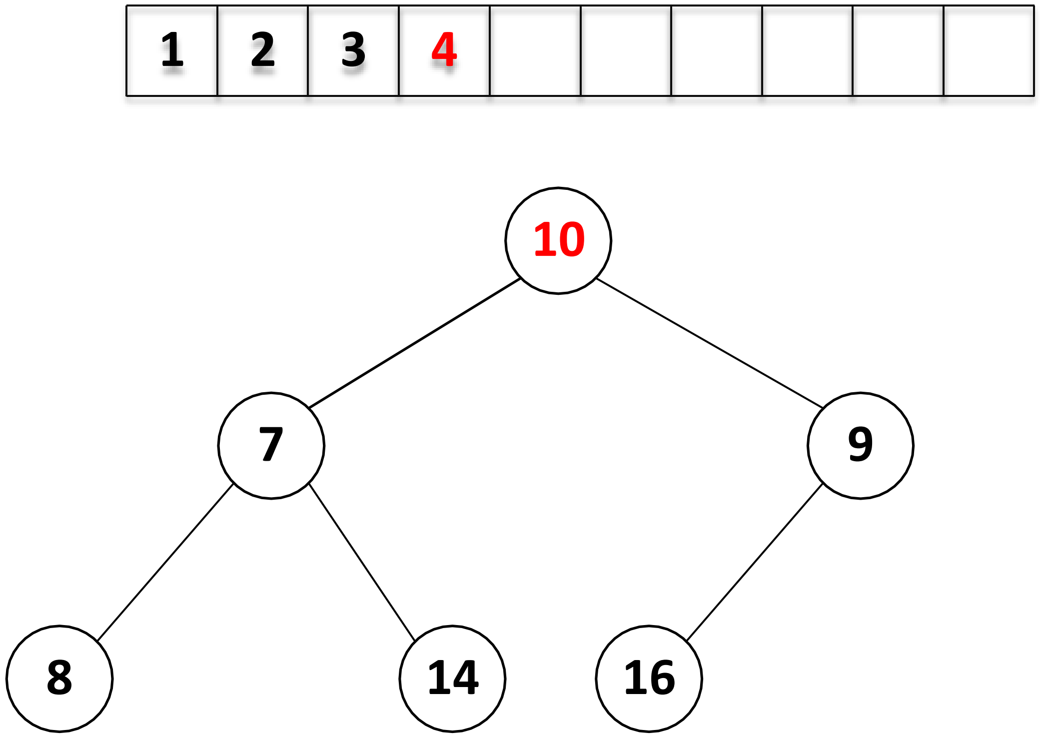
堆排序：算法实例

- 执行n次**Extract-Min**操作



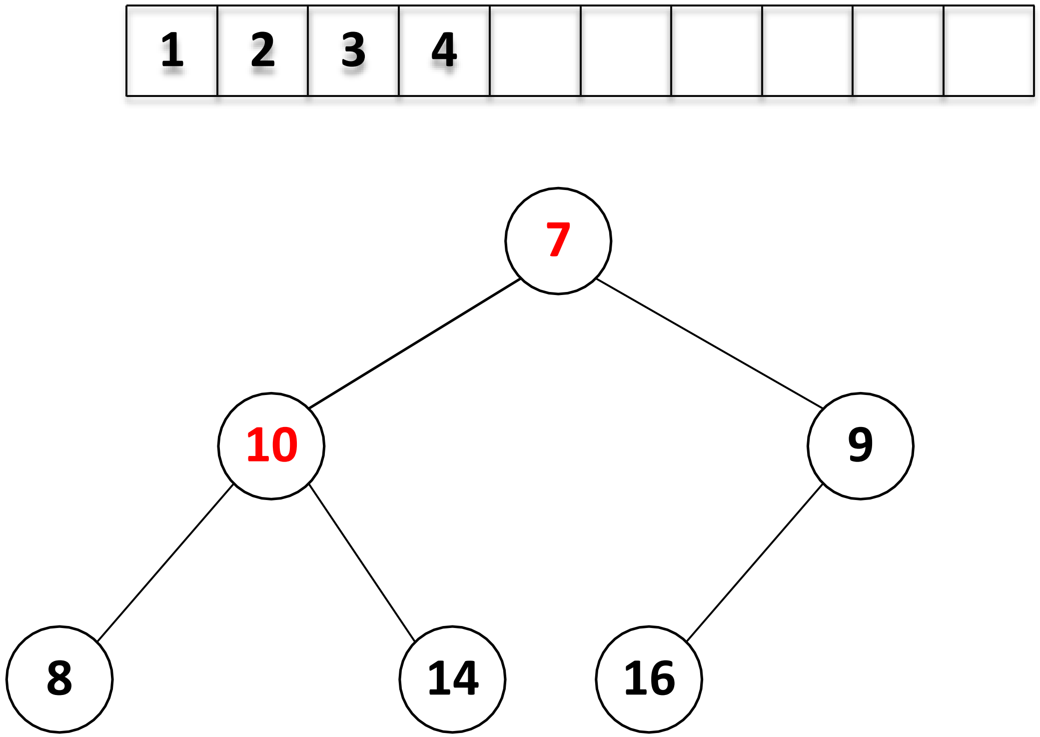
堆排序：算法实例

- 执行n次Extract-Min操作



堆排序：算法实例

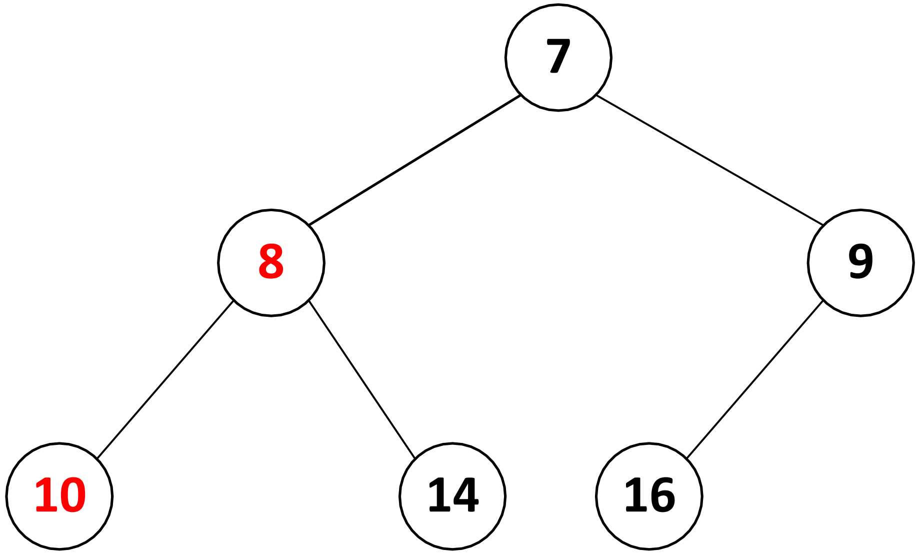
- 执行n次Extract-Min操作



堆排序：算法实例

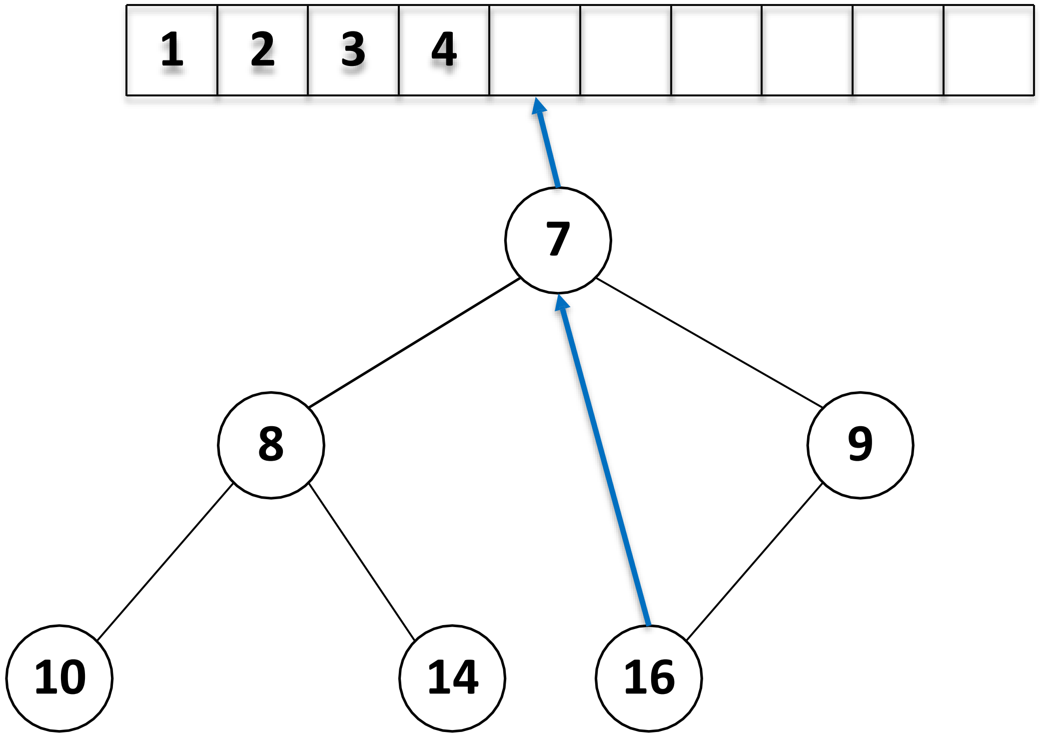
- 执行n次Extract-Min操作

1	2	3	4						



堆排序：算法实例

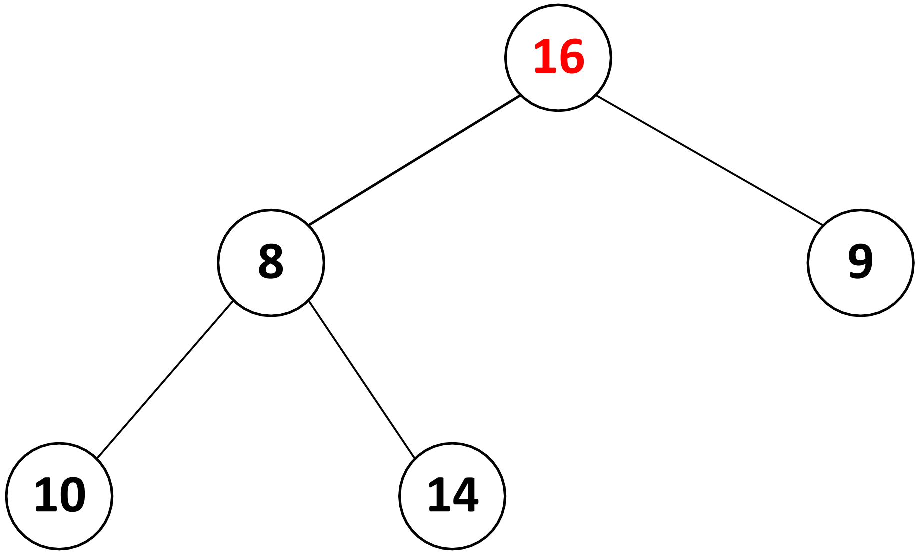
- 执行n次Extract-Min操作



堆排序：算法实例

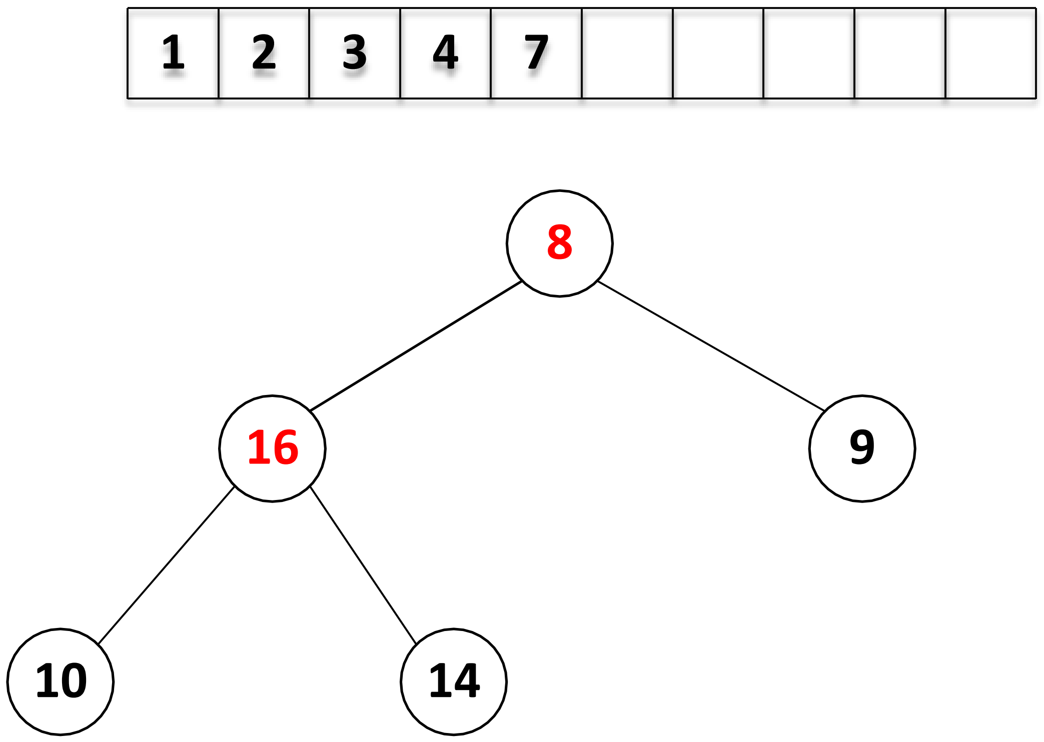
- 执行n次Extract-Min操作

1	2	3	4	7					



堆排序：算法实例

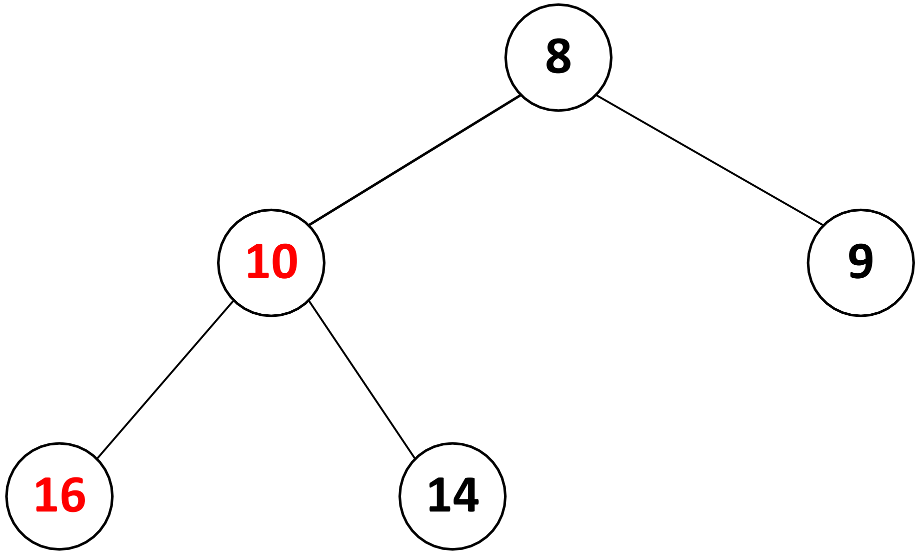
- 执行n次Extract-Min操作



堆排序：算法实例

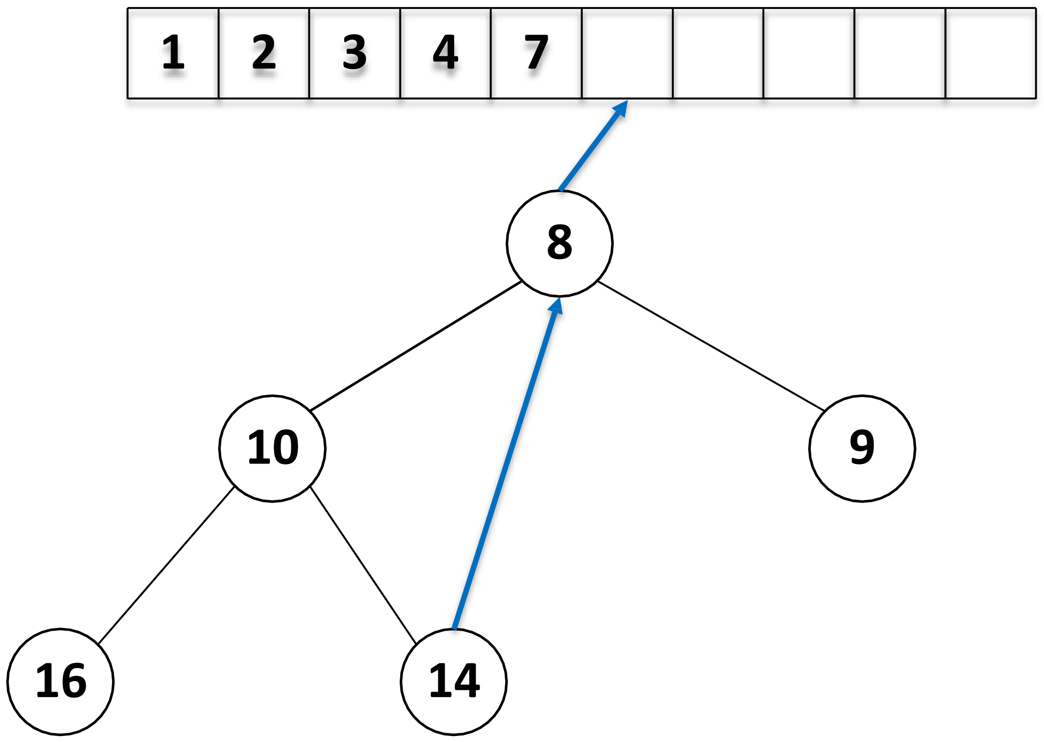
- 执行n次Extract-Min操作

1	2	3	4	7					



堆排序：算法实例

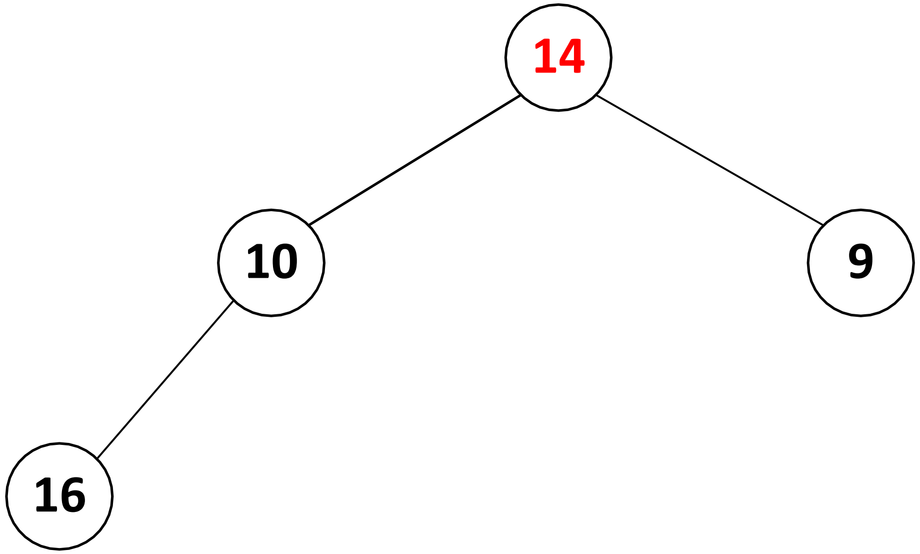
- 执行n次Extract-Min操作



堆排序：算法实例

- 执行n次Extract-Min操作

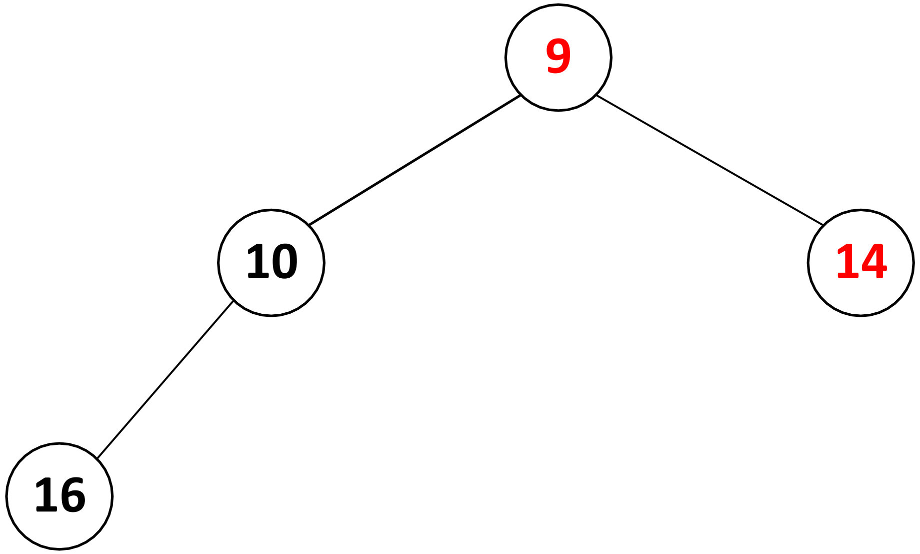
1	2	3	4	7	8				



堆排序：算法实例

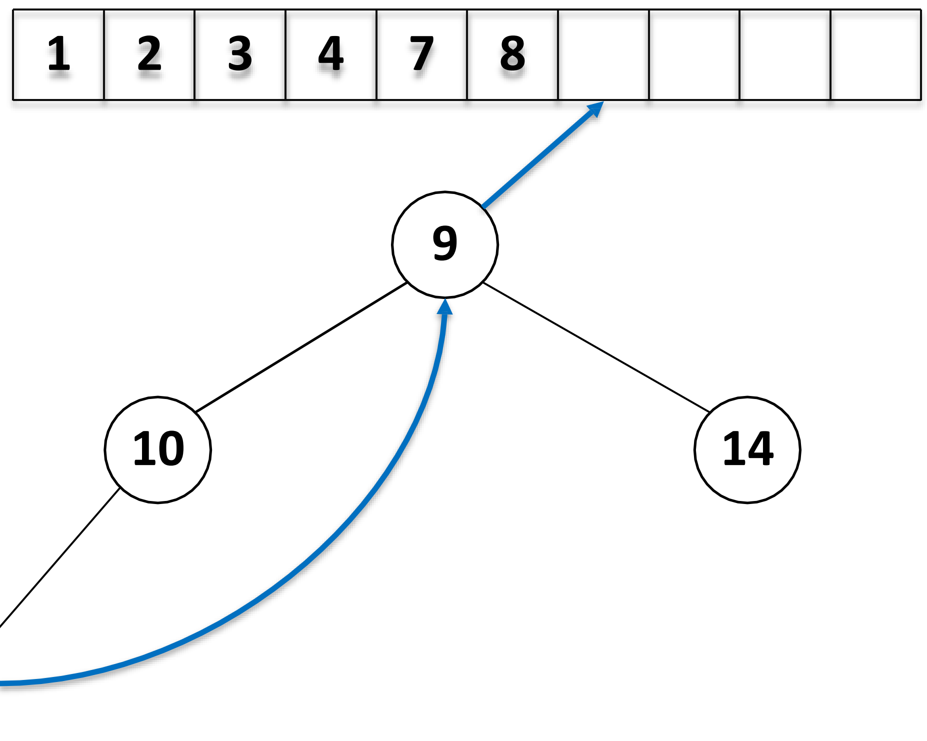
- 执行n次Extract-Min操作

1	2	3	4	7	8				



堆排序：算法实例

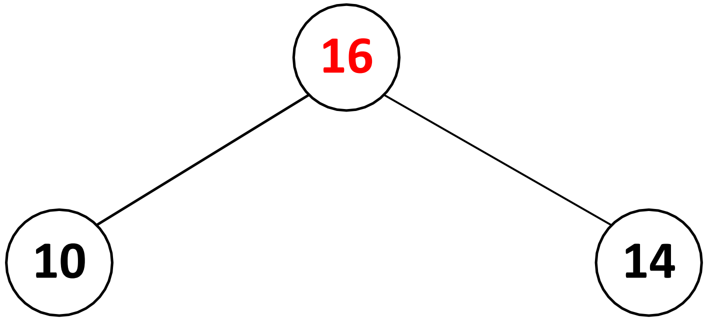
- 执行n次**Extract-Min**操作



堆排序：算法实例

- 执行n次Extract-Min操作

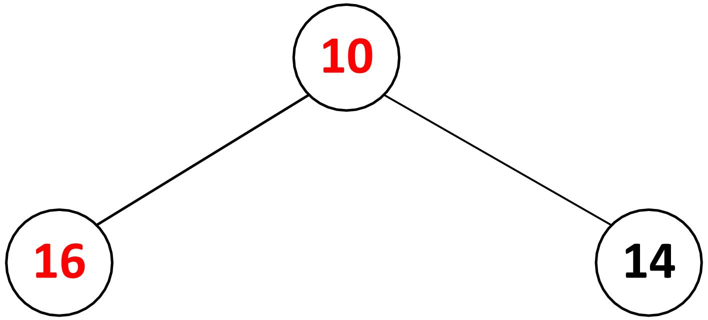
1	2	3	4	7	8	9			



堆排序：算法实例

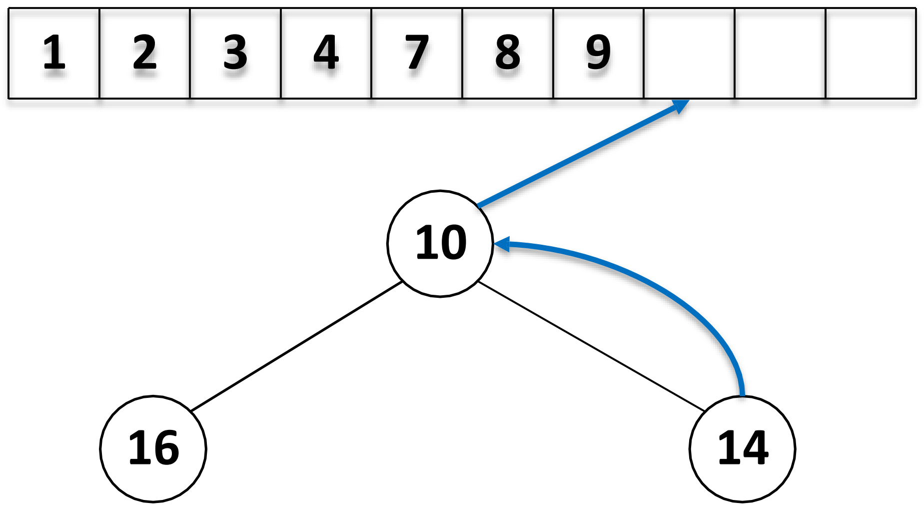
- 执行n次**Extract-Min**操作

1	2	3	4	7	8	9			



堆排序：算法实例

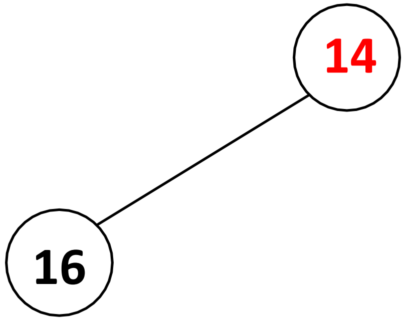
- 执行n次Extract-Min操作



堆排序：算法实例

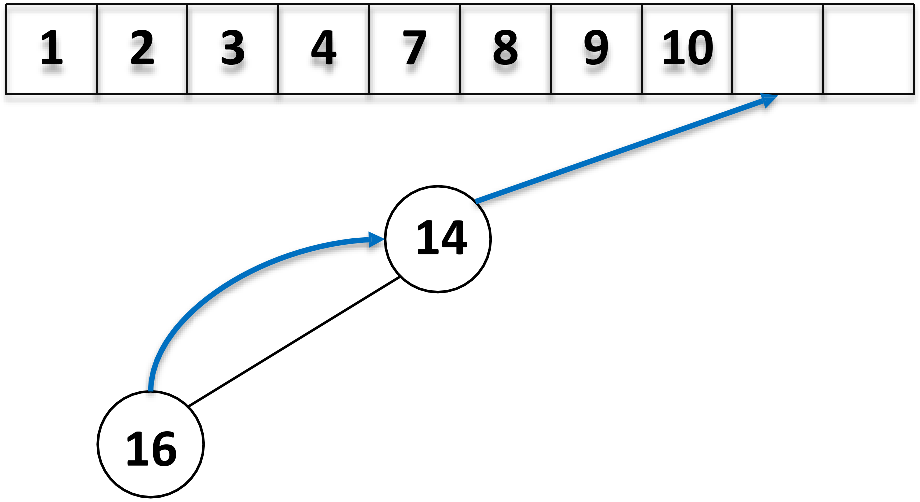
- 执行n次**Extract-Min**操作

1	2	3	4	7	8	9	10		



堆排序：算法实例

- 执行n次Extract-Min操作



堆排序：算法实例

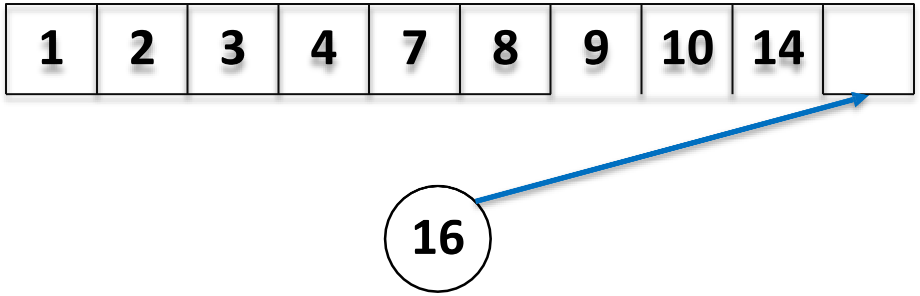
- 执行n次**Extract-Min**操作

1	2	3	4	7	8	9	10	14	

16

堆排序：算法实例

- 执行n次Extract-Min操作



堆排序：算法实例

- 执行n次**Extract-Min**操作

1	2	3	4	7	8	9	10	14	16

堆排序：算法实例

- 执行n次**Extract-Min**操作

1	2	3	4	7	8	9	10	14	16

堆排序小结

- 优先队列是一个抽象的数据结构，支持下述两种操作：**Insert** 和 **Extract-Min**。

堆排序小结

- 优先队列是一个抽象的数据结构，支持下述两种操作：**Insert** 和 **Extract-Min**。

- 如果使用堆来实现优先级队列，那么在$\boldsymbol{O}(\log n)$时间内可以支持这两种操作。

堆排序小结

- 优先队列是一个抽象的数据结构，支持下述两种操作：**Insert** 和 **Extract-Min**。

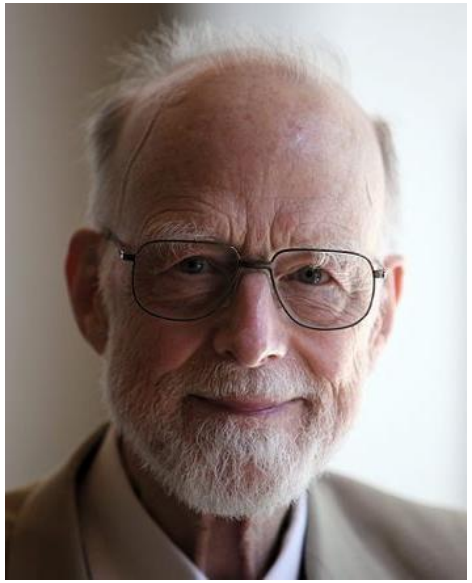
- 如果使用堆来实现优先级队列，那么在$O(\log n)$时间内可以支持这两种操作。

- 堆排序需要$O(n \log n)$时间，与归并排序和快速排序一样高效。

堆排序小结

John von Neumann

1945提出归并排序算法



Tony Hoare

1959年提出快速排序算法

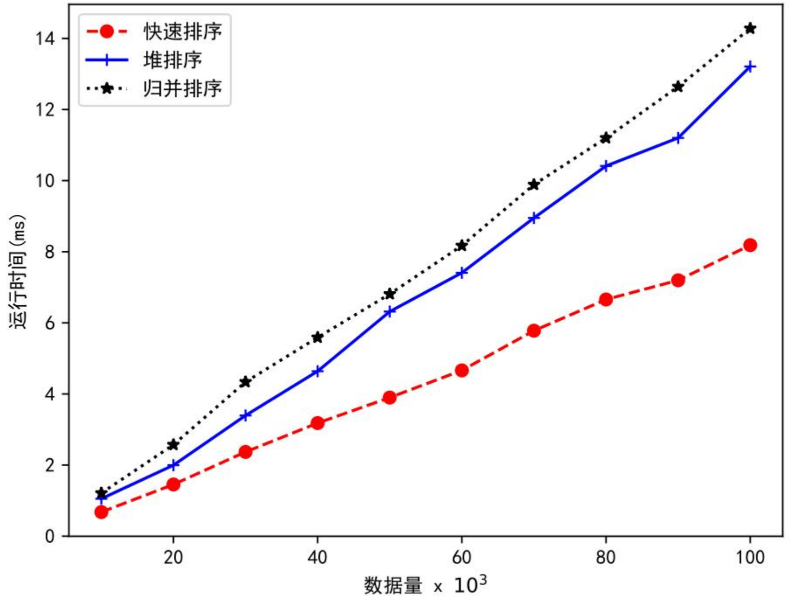
J. W. J. Williams

1964年提出堆排序算法

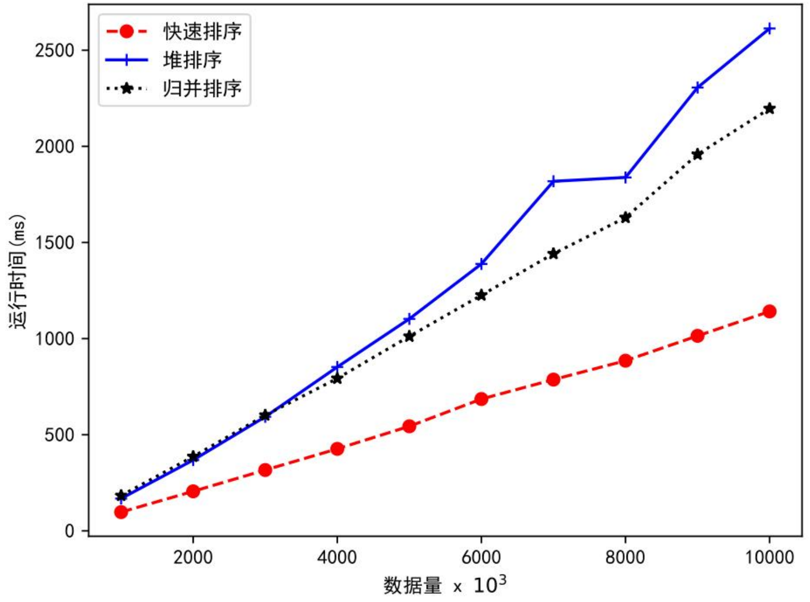
哪种算法在实践中是最好的？

实验比较

- 比较不同数规模下三种算法的运行时间，实验结果如下：



数据规模较小



数据规模较大

是否存在更快的排序算法？

提纲

- 优先队列
- （二叉）堆
- 堆排序
- 排序算法的下界
- 计数排序

猜测

- 目前遇到的排序算法都基于元素的比较
  - 例如：插入排序，归并排序，堆排序

猜测

- 目前遇到的排序算法都基于元素的比较
  - 例如：插入排序，归并排序，堆排序
- 插入排序的时间复杂度最高--$\boldsymbol{\theta(n^2)}$，其他两种排序算法时间复杂度是 $\boldsymbol{\theta(n\log n)}$

猜测

- 目前遇到的排序算法都基于元素的比较
  - 例如：插入排序，归并排序，堆排序
- 插入排序的时间复杂度最高--$\boldsymbol{\theta(n^2)}$，其他两种排序算法时间复杂度是 $\boldsymbol{\theta(n\log n)}$

问题：是否存在更快的排序算法？

猜测

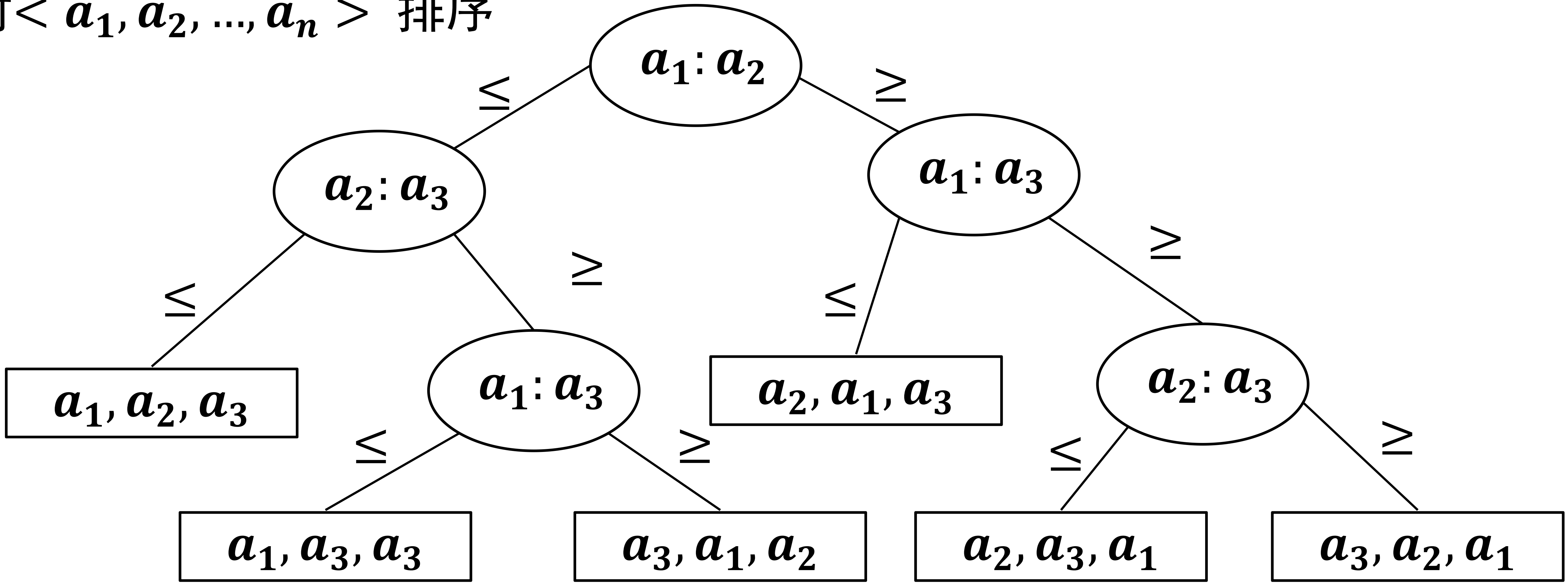
- 目前遇到的排序算法都基于元素的比较
  - 例如：插入排序，归并排序，堆排序
- 插入排序的时间复杂度最高--$\boldsymbol{\theta(n^2)}$，其他两种排序算法时间复杂度是 $\boldsymbol{\theta(n\log n)}$

问题：**是否存在更快的排序算法？**

猜测：对于任何基于比较的排序算法
其时间复杂度的下界均为 $\Omega(\mathbf{n\log n})$

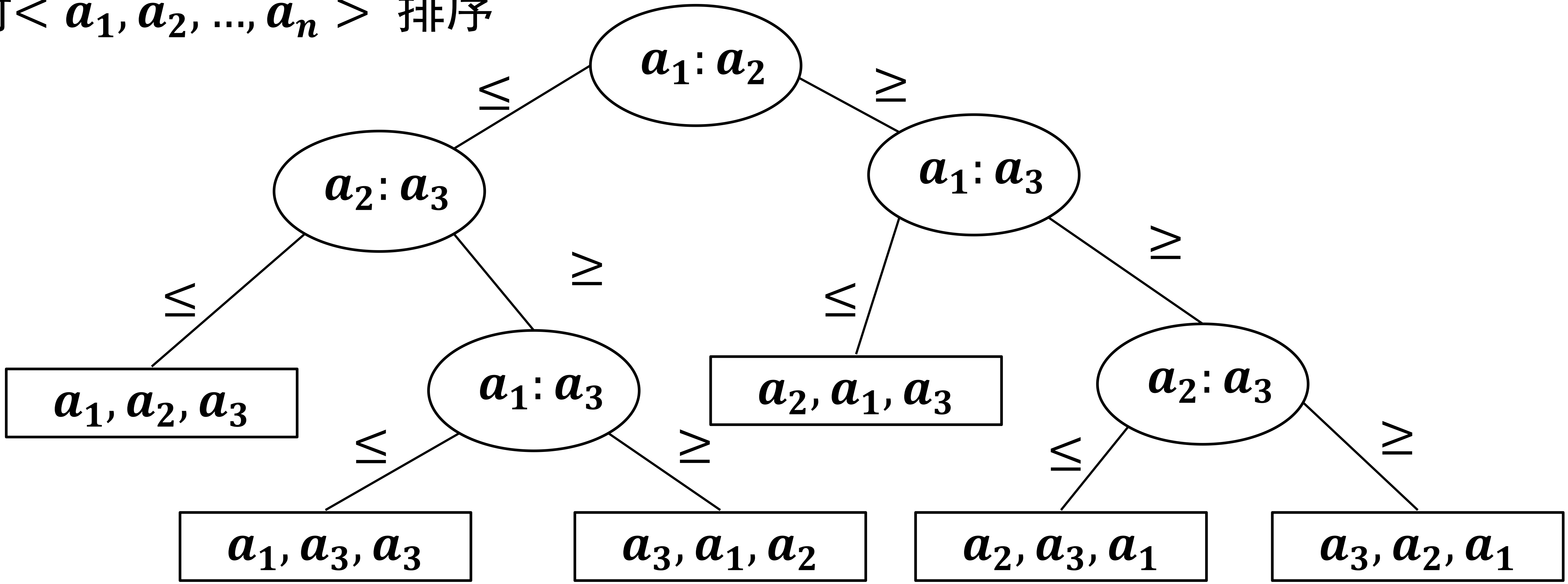
决策树模型

对$< a_1, a_2, \ldots, a_n >$ 排序



决策树模型

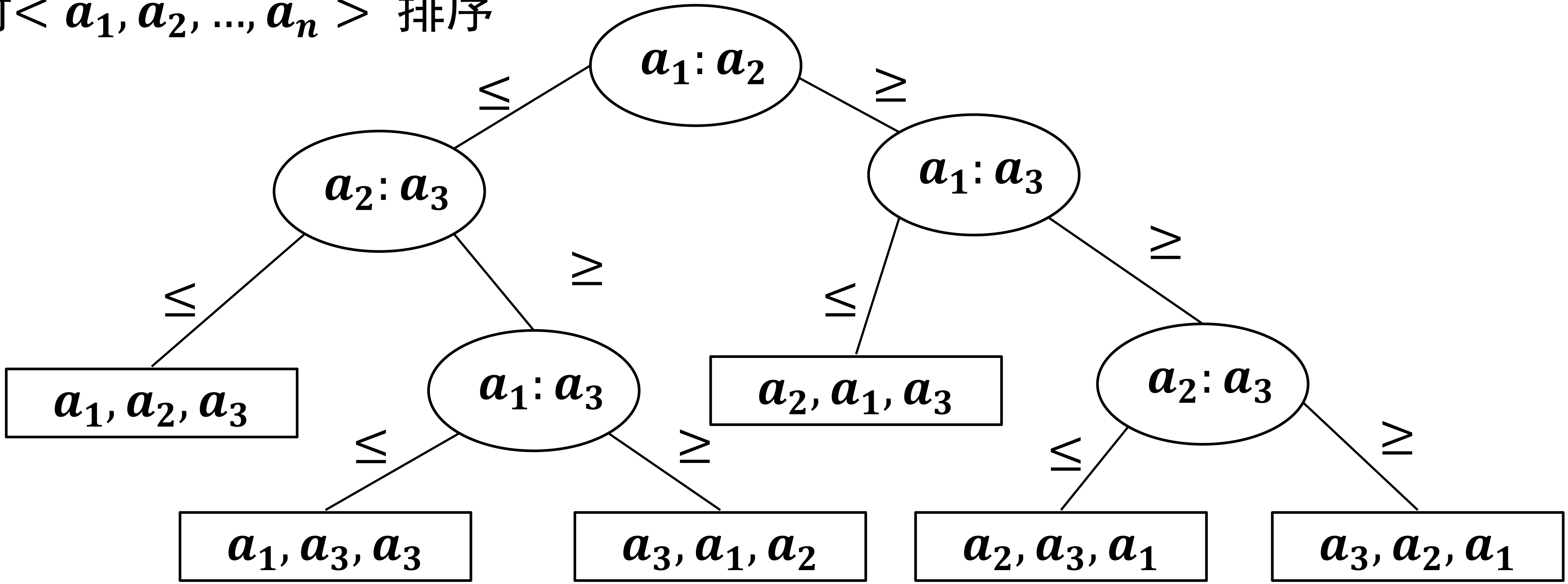
对$< a_1, a_2, \ldots, a_n >$ 排序



- 每个内部节点标记为 $a_i : a_j$（$i, j \in \{1,2, \ldots n\}$）

决策树模型

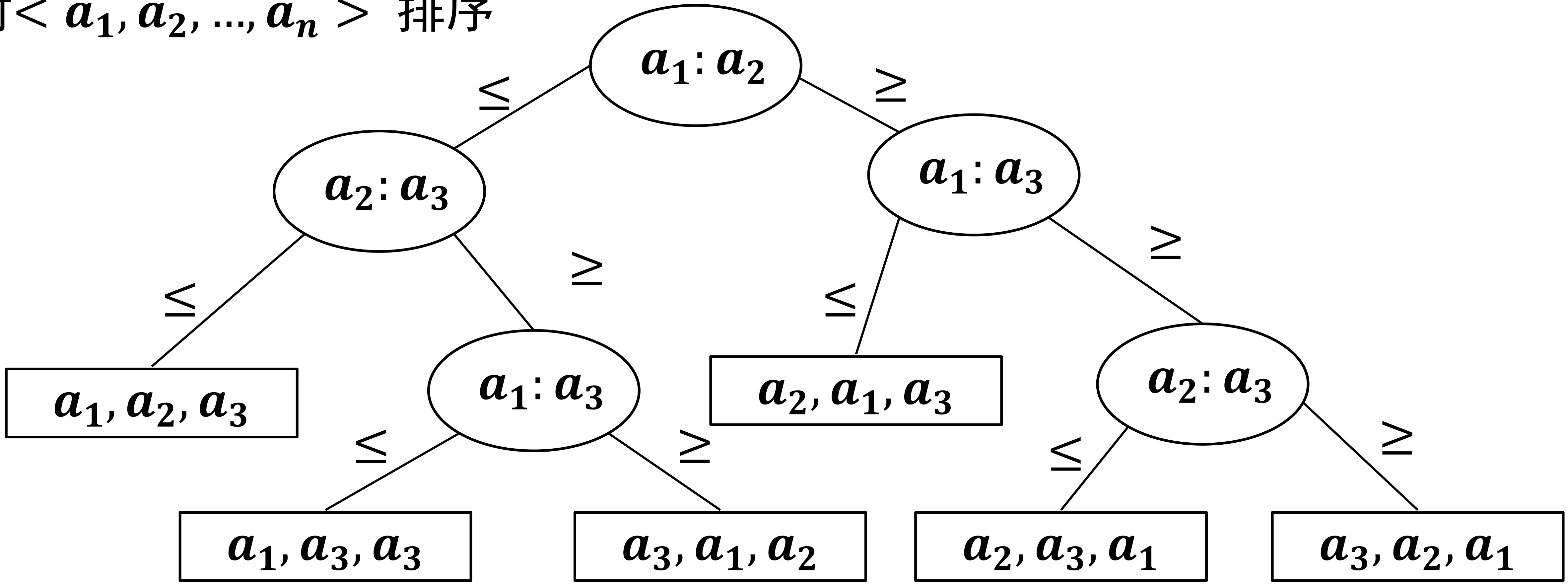
对$< a_1, a_2, \ldots, a_n >$ 排序



- 每个内部节点标记为 $a_i : a_j$（$i, j \in \{1, 2, \ldots n\}$）
  - 左子树表示当$a_i \le a_j$时，随后的比较情况

决策树模型

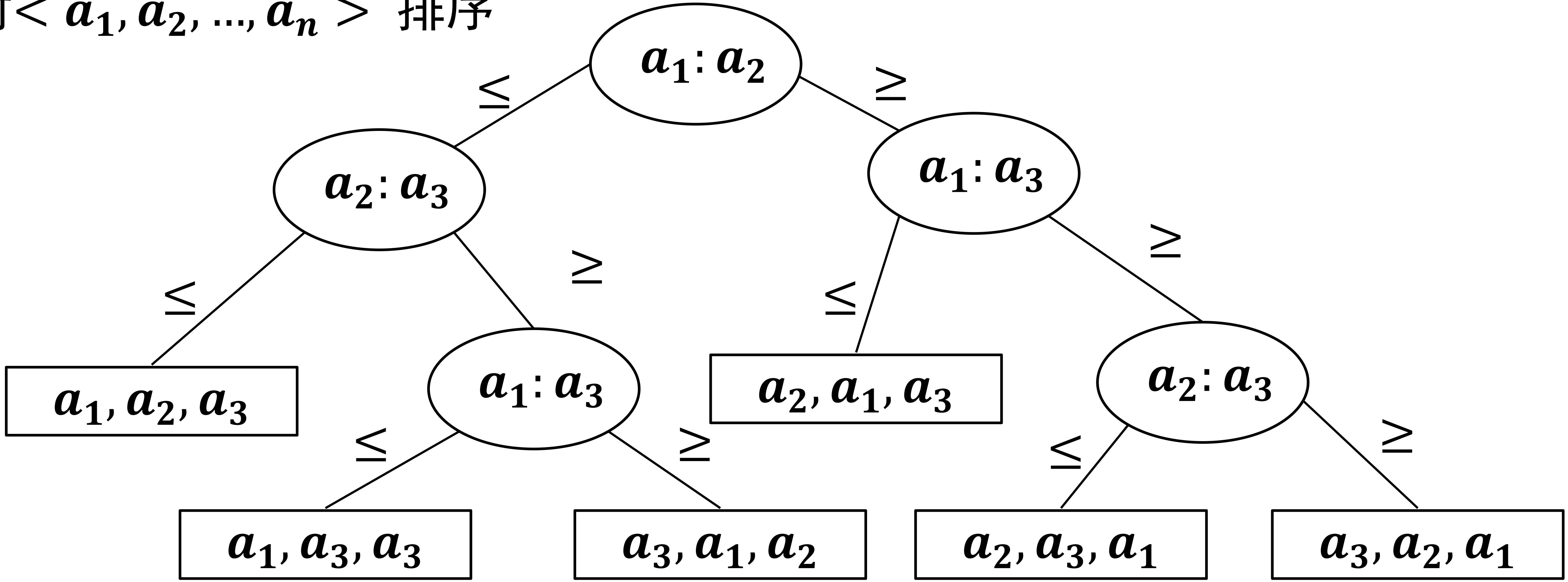
对$< a_1, a_2, \ldots, a_n >$ 排序



- 每个内部节点标记为 $a_i : a_j$（$i, j \in \{1,2,\ldots n\}$）
  - 左子树表示当$a_i \leq a_j$时，随后的比较情况
  - 右子树表示当$a_i > a_j$时，随后的比较情况

决策树模型

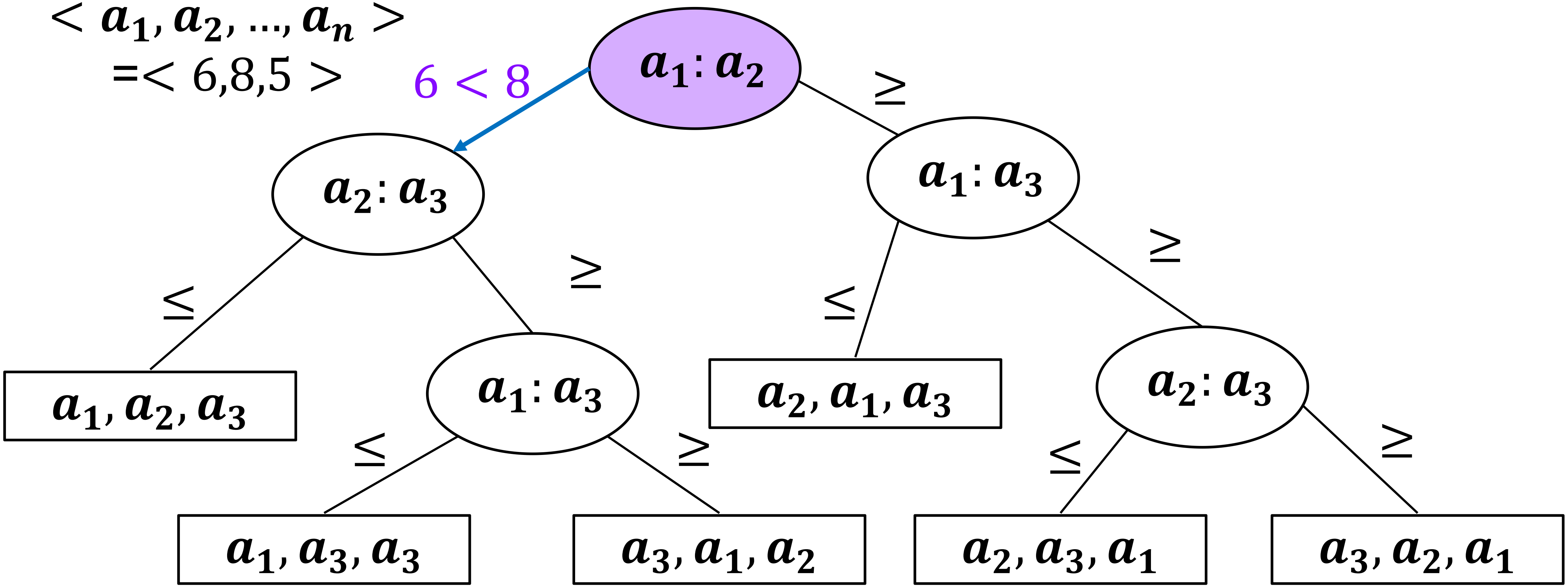
对$< a_1, a_2, \ldots, a_n >$ 排序



- 每个内部节点标记为 $a_i : a_j$（$i, j \in \{1, 2, \ldots n\}$）
  - 左子树表示当$a_i \leq a_j$时，随后的比较情况
  - 右子树表示当$a_i > a_j$时，随后的比较情况
- 每个叶子节点对应一种输入顺序

决策树模型

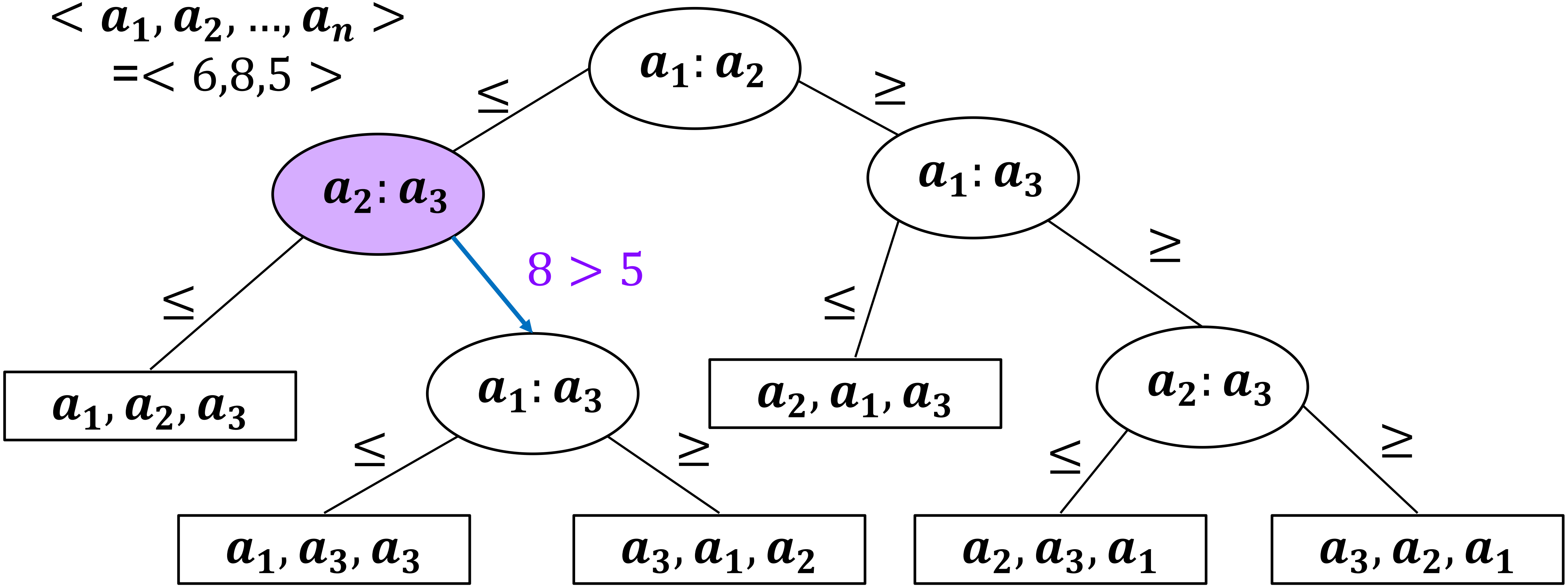
$< \boldsymbol{a_1, a_2, \ldots, a_n} >$
$=< 6,8,5 >$



- 每个内部节点标记为 a_i: a_j （$i, j \in \{1,2, \ldots n\}$）
  - 左子树表示当$a_i \le a_j$时，随后的比较情况
  - 右子树表示当$a_i > a_j$时，随后的比较情况
- 每个叶子节点对应一种输入顺序

决策树模型

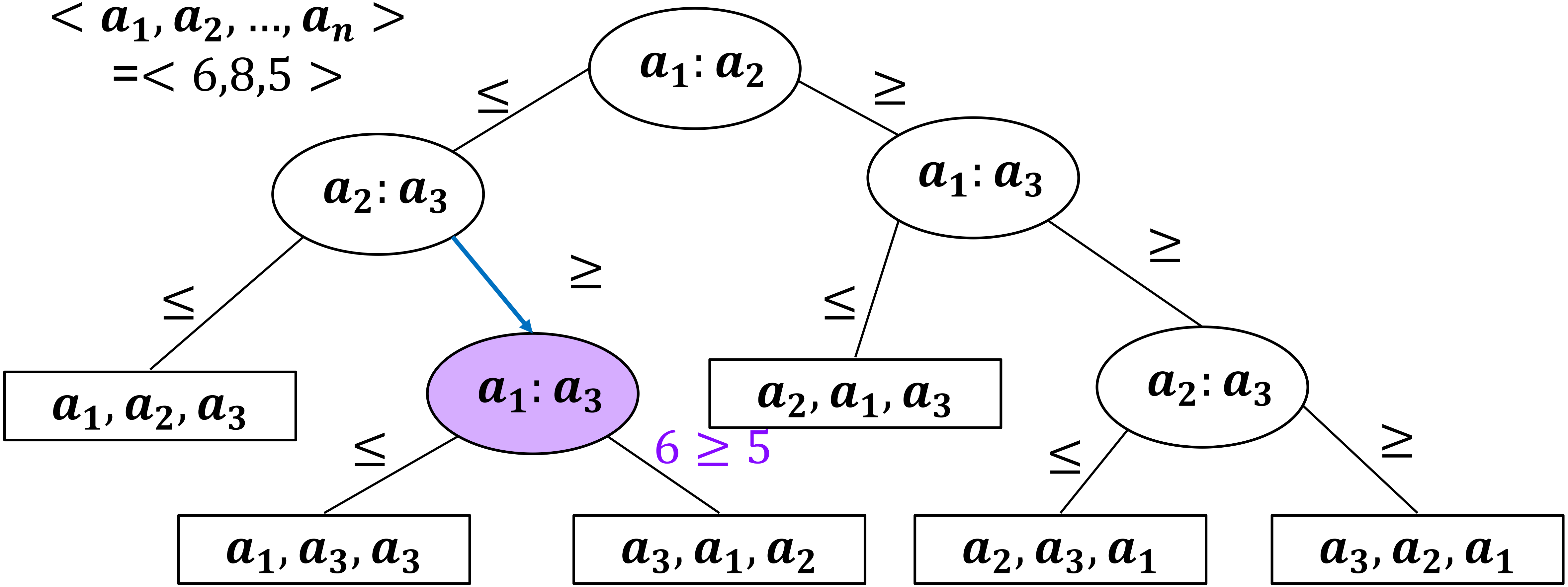
$< a_1, a_2, \ldots, a_n >$
$=< 6,8,5 >$



- 每个内部节点标记为 $a_i: a_j$（$i, j \in \{1,2, \ldots n\}$）
  - 左子树表示当$a_i \leq a_j$时，随后的比较情况
  - 右子树表示当$a_i > a_j$时，随后的比较情况
- 每个叶子节点对应一种输入顺序

决策树模型

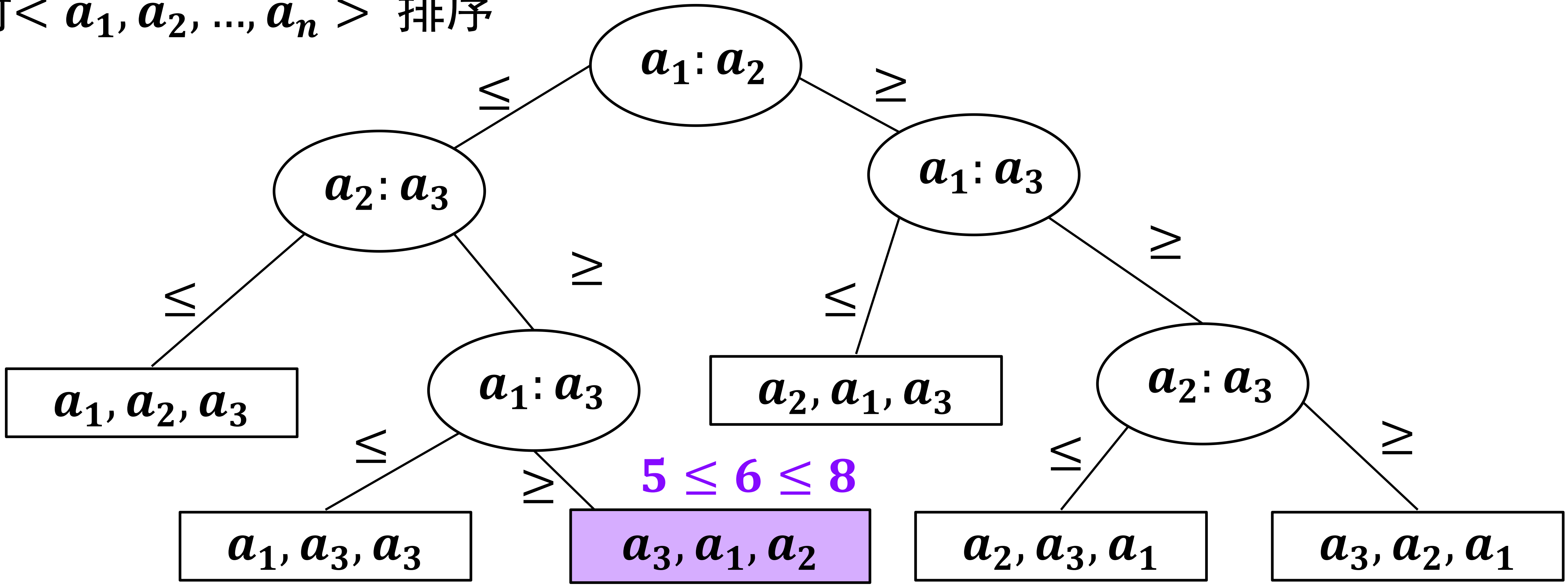
$< a_1, a_2, \ldots, a_n >$
$=< 6,8,5 >$



- 每个内部节点标记为 $a_i : a_j$（$i, j \in \{1,2, \ldots n\}$）
  - 左子树表示当$a_i \leq a_j$时，随后的比较情况
  - 右子树表示当$a_i > a_j$时，随后的比较情况
- 每个叶子节点对应一种输入顺序

决策树模型

对$< a_1, a_2, \ldots, a_n >$ 排序



- 每个内部节点标记为 $a_i : a_j$（$i, j \in \{1,2,\ldots n\}$）
  - 左子树表示当$a_i \leq a_j$时，随后的比较情况
  - 右子树表示当$a_i > a_j$时，随后的比较情况
- 每个叶子节点对应一种输入顺序

决策树模型

决策树模型能够模拟任意基于比较的排序算法的执行过程

- 每个大小为**n**的输入都能对应一个决策树

决策树模型

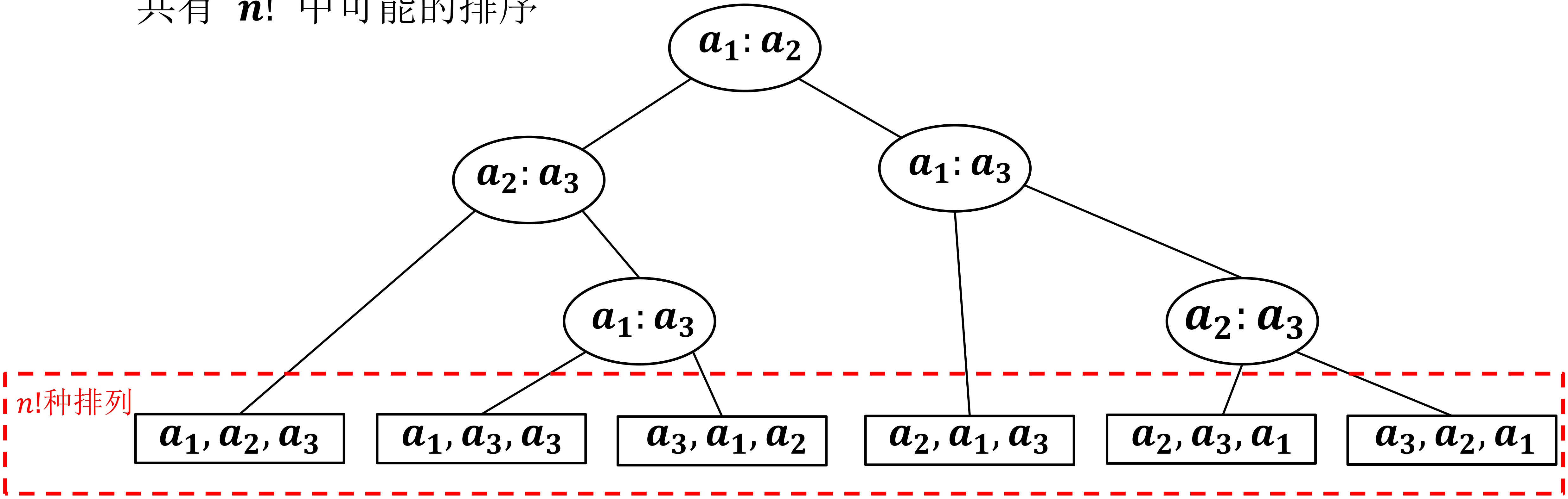
- 决策树模型能够模拟任意基于比较的排序算法的执行过程
- 每个大小为**n**的输入都能对应一个决策树
- 最坏情况的运行时间 = 决策树的高度

排序算法的下界

- 定理
  - 任意基于比较的排序算法都需要 $\boldsymbol{\Omega(n\log n)}$ 次比较

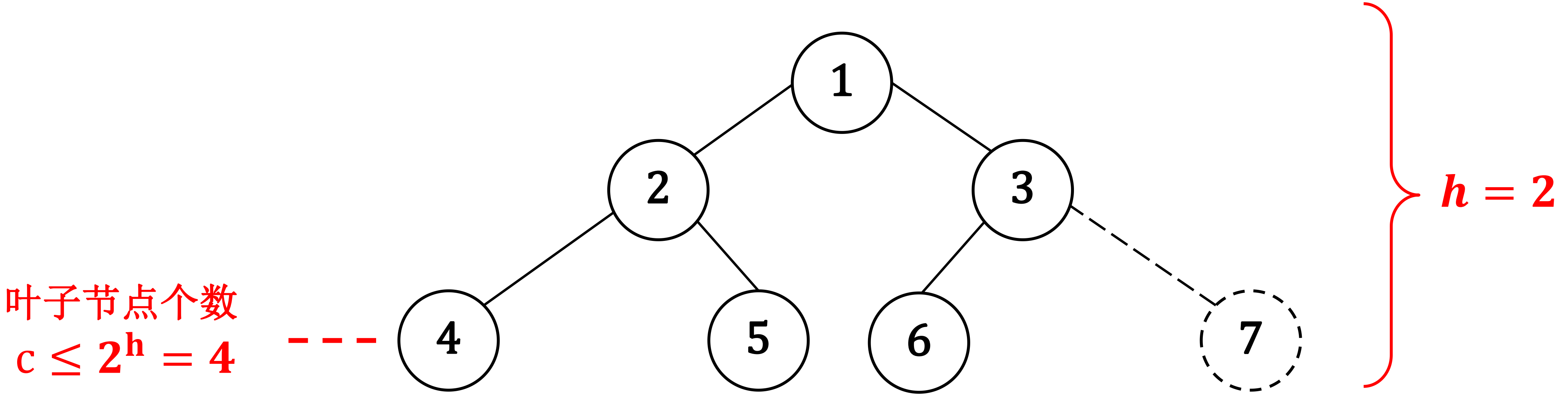
排序算法的下界

- 定理
  - 任意基于比较的排序算法都需要 $\boldsymbol{\Omega(n\log n)}$ 次比较
- 证明
  - 对于一个模拟 $\boldsymbol{n}$ 个元素排序的决策树，其至少含有 $\boldsymbol{n}!$ 个叶子节点，因为共有 $\boldsymbol{n}!$ 中可能的排序



排序算法的下界

- 定理
  - 任意基于比较的排序算法都需要 $\boldsymbol{\Omega(n\log n)}$ 次比较
- 证明
  - 对于一个模拟 $\boldsymbol{n}$ 个元素排序的决策树，其至少含有 $\boldsymbol{n!}$ 个叶子节点，因为共有 $\boldsymbol{n!}$ 中可能的排序
  - 一个高度为 $\boldsymbol{h}$ 的二叉树最多有 $\boldsymbol{2^h}$ 个叶子节点



排序算法的下界

- 定理
  - 任意基于比较的排序算法都需要 $\boldsymbol{\Omega(n\log n)}$ 次比较
- 证明
  - 对于一个模拟 $\boldsymbol{n}$ 个元素排序的决策树，其至少含有 $\boldsymbol{n!}$ 个叶子节点，因为共有 $\boldsymbol{n!}$ 中可能的排序
  - 一个高度为 $\boldsymbol{h}$ 的二叉树最多有 $\boldsymbol{2^h}$ 个叶子节点
  - 因此，$\boldsymbol{n! \leq 2^h}$

    $\Rightarrow \boldsymbol{h \geq \log n! = \Omega(n\log n)}$（前面课程中已经证明）

排序算法的下界

- 定理
  - 任意基于比较的排序算法都需要 $\boldsymbol{\Omega(n\log n)}$ 次比较
- 证明
  - 对于一个模拟 $\boldsymbol{n}$ 个元素排序的决策树，其至少含有 $\boldsymbol{n!}$ 个叶子节点，因为共有 $\boldsymbol{n!}$ 中可能的排序
  - 一个高度为 $\boldsymbol{h}$ 的二叉树最多有 $\boldsymbol{2^h}$ 个叶子节点
  - 因此，$\boldsymbol{n! \leq 2^h}$

    $\Rightarrow \boldsymbol{h \geq \log n! = \Omega(n\log n)}$（前面课程中已经证明）
- 推论
  - 堆排序以及归并排序是渐进最优的基于比较的排序算法

排序算法的下界

- 定理
  - 任意基于比较的排序算法都需要 $\Omega(n\log n)$ 次比较
- 证明
  - 对于一个模拟 n 个元素排序的决策树，其至少含有 $n!$ 个叶子节点，因为共有 $n!$ 中可能的排序
  - 一个高度为 h 的二叉树最多有 2^h 个叶子节点
  - 因此，$n! \leq 2^h$

    $\Rightarrow h \geq \log n! \ = \Omega(n\log n)$（前面课程中已经证明）
- 推论
  - 堆排序以及归并排序是渐进最优的基于比较的排序算法

问题：存在不基于比较的排序算法吗？
这些算法能够打破$\Omega(n\log n)$的下界吗?

提纲

- 优先队列
- （二叉）堆
- 堆排序
- 排序算法的下界
- 计数排序

主要思想

- 对于各个输入元素 $\boldsymbol{x}$，计数排序能够确定小于 $\boldsymbol{x}$ 的元素的数量

主要思想

- 对于各个输入元素 $\boldsymbol{x}$，计数排序能够确定小于 $\boldsymbol{x}$ 的元素的数量

- 计数排序根据这个信息能够直接将元素 $\boldsymbol{x}$ 放在输出数组的对应位置

主要思想

- 对于各个输入元素 $\boldsymbol{x}$，计数排序能够确定小于 $\boldsymbol{x}$ 的元素的数量

- 计数排序根据这个信息能够直接将元素 $\boldsymbol{x}$ 放在输出数组的对应位置
  - 例如，如果有**17**个元素小于 $\boldsymbol{x}$,那么 $\boldsymbol{x}$ 就在输出的第**18**位

运行实例：计数排序

计数排序

输入: 数组$A[1..n]$,其中$A[j] \in \{1, 2, ..., k\}$
输出: 排序后的数组 $B[1, ..., n]$

```
C ← []
for i ← 1 to k do
|   C[i] ← 0;
end
for j← 1ton do
|   C[A[j]] ← C[A[j]] + 1;   //C[i] = |{key = i}|
end
for i ← 2 to k do
|   C[i] ← C[i] + C[i − 1];   //C[i] = |{key ≤ i}|
end
for j← n to 1 do
|   B[C[A[j]]] ← A[j];
|   C[A[j]] ← C[A[j]] − 1;
end
return B;
```

运行实例：计数排序

计数排序

输入: 数组$A[1..n]$, 其中$A[j] \in \{1, 2, ..., k\}$
输出: 排序后的数组 $B[1, ..., n]$

$C \leftarrow []$
for $i \leftarrow 1$ *to* k **do**
　$C[i] \leftarrow 0;$
end
for $j \leftarrow 1 to n$ **do**
　$C[A[j]] \leftarrow C[A[j]] + 1;$　$//C[i] = |\{key = i\}|$
end
for $i \leftarrow 2$ *to* k **do**
　$C[i] \leftarrow C[i] + C[i-1];$　$//C[i] = |\{key \leq i\}|$
end
for $j \leftarrow n$ *to* 1 **do**
　$B[C[A[j]]] \leftarrow A[j];$
　$C[A[j]] \leftarrow C[A[j]] - 1;$
end
return $B;$

运行实例：计数排序

A

1	2	3	4	5
4	**2**	**1**	**4**	**2**

C

B

运行实例：计数排序

计数排序

输入: 数组$A[1..n]$,其中$A[j] \in \{1, 2, ..., k\}$
输出: 排序后的数组 $B[1, ..., n]$

$C \leftarrow []$
for $i \leftarrow 1$ *to* k **do**
  | $C[i] \leftarrow 0;$
end
for $j \leftarrow 1 to n$ **do**
  | $C[A[j]] \leftarrow C[A[j]] + 1;$ $//C[i] = |\{key = i\}|$
end
for $i \leftarrow 2$ *to* k **do**
  | $C[i] \leftarrow C[i] + C[i-1];$ $//C[i] = |\{key \leq i\}|$
end
for $j \leftarrow n$ *to* 1 **do**
  | $B[C[A[j]]] \leftarrow A[j];$
  | $C[A[j]] \leftarrow C[A[j]] - 1;$
end
return $B;$

初始化中间数组C

运行实例：计数排序

	1	2	3	4	5
A	**4**	**2**	**1**	**4**	**2**

	1	2	3	4
C	**0**	**0**	**0**	**0**

B					

for $i \leftarrow 1$ *to* k **do**
| $C[i] \leftarrow 0;$
end

运行实例：计数排序

计数排序

输入: 数组$A[1..n]$,其中$A[j] \in \{1,2,...,k\}$
输出: 排序后的数组 $B[1,...,n]$

$C \leftarrow []$
for $i \leftarrow 1$ *to* k **do**
　$C[i] \leftarrow 0;$
end
for $j \leftarrow 1 ton$ **do**
　$C[A[j]] \leftarrow C[A[j]] + 1;$ 　$//C[i] = |\{key = i\}|$
end
for $i \leftarrow 2$ *to* k **do**
　$C[i] \leftarrow C[i] + C[i-1];$ 　$//C[i] = |\{key \leq i\}|$
end
for $j \leftarrow n$ *to* 1 **do**
　$B[C[A[j]]] \leftarrow A[j];$
　$C[A[j]] \leftarrow C[A[j]] - 1;$
end
return $B;$

$C[i]$对应输入数组A中i出现的次数

运行实例：计数排序

	1	2	3	4	5
A	**4**	**2**	**1**	**4**	**2**
B					

	1	2	3	4
C	**0**	**0**	**0**	**1**

for $j \leftarrow 1ton$ **do**

| $C[A[j]] \leftarrow C[A[j]] + 1;$ $//C[i] = |\{key = i\}|$

end

运行实例：计数排序

	1	2	3	4	5
A	4	2	1	4	2

	1	2	3	4
C	0	1	0	1

B					

for $j \leftarrow 1 to n$ **do**

| $C[A[j]] \leftarrow C[A[j]] + 1;$ $//C[i] = |\{key = i\}|$

end

运行实例：计数排序

	1	2	3	4	5
A	**4**	**2**	**1**	**4**	**2**
B					

	1	2	3	4
C	**1**	**1**	**0**	**1**

for $j \leftarrow 1 to n$ **do**

  $C[A[j]] \leftarrow C[A[j]] + 1;$ $//C[i] = |\{key = i\}|$

end

运行实例：计数排序

	1	2	3	4	5
A	**4**	**2**	**1**	**4**	**2**

	1	2	3	4
C	**1**	**1**	**0**	**2**

B					

for $j \leftarrow 1 ton$ **do**
$\quad C[A[j]] \leftarrow C[A[j]] + 1;$ $//C[i] = |\{key = i\}|$
end

运行实例：计数排序

A

1	2	3	4	5
4	2	1	4	2

C

1	2	3	4
1	2	0	2

B

for $j \leftarrow 1 ton$ **do**

 | $C[A[j]] \leftarrow C[A[j]] + 1;$ $//C[i] = |\{key = i\}|$

end

运行实例：计数排序

计数排序

输入: 数组$A[1..n]$，其中$A[j] \in \{1, 2, ..., k\}$

输出: 排序后的数组 $B[1, ..., n]$

$C \leftarrow []$

for $i \leftarrow 1$ *to* k **do**

 $C[i] \leftarrow 0;$

end

for $j \leftarrow 1 to n$ **do**

 $C[A[j]] \leftarrow C[A[j]] + 1;$ $//C[i] = |\{key = i\}|$

end

for $i \leftarrow 2$ *to* k **do**

 $C[i] \leftarrow C[i] + C[i-1];$ $//C[i] = |\{key \leq i\}|$

end

for $j \leftarrow n$ *to* 1 **do**

 $B[C[A[j]]] \leftarrow A[j];$

 $C[A[j]] \leftarrow C[A[j]] - 1;$

end

return B;

$C[i]$对应输入数组A中$\leq i$元素出现的次数

运行实例：计数排序

	1	2	3	4	5
A	**4**	**2**	**1**	**4**	**2**
B					

	1	2	3	4
C	**1**	**2**	**0**	**2**

+

C	**1**	**3**	**0**	**2**

for $i \leftarrow 2$ *to* k **do**

$\quad C[i] \leftarrow C[i] + C[i-1];\quad //C[i] = |\{key \leq i\}|$

end

运行实例：计数排序

	1	2	3	4	5
A	**4**	**2**	**1**	**4**	**2**

B					

	1	2	3	4
C	**1**	**3**	**0**	**2**

+

C	**1**	**3**	**3**	**2**

for $i \leftarrow 2$ *to* k **do**

$\quad C[i] \leftarrow C[i] + C[i-1];$ $//C[i] = |\{key \leq i\}|$

end

运行实例：计数排序

	1	2	3	4	5
A	**4**	**2**	**1**	**4**	**2**
B					

	1	2	3	4
C	**1**	**3**	**3**	**2**
C	**1**	**3**	**3**	**5**

for $i \leftarrow 2$ *to* k **do**

$\quad C[i] \leftarrow C[i] + C[i-1];$ $\color{blue}{//C[i] = |\{key \leq i\}|}$

end

运行实例：计数排序

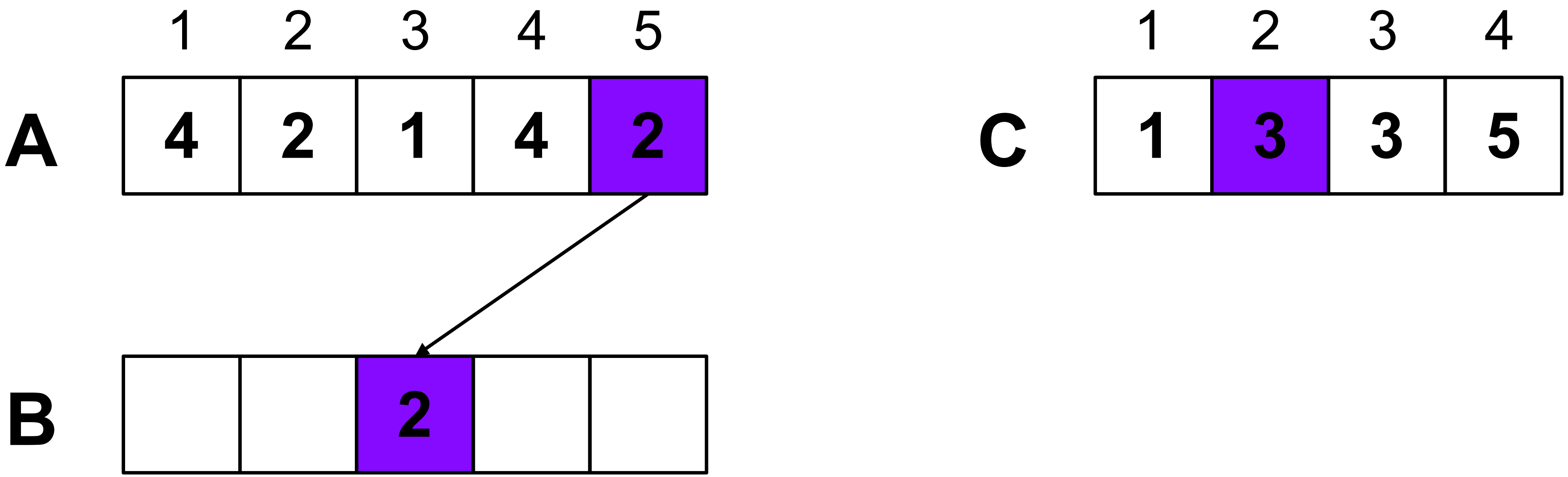
计数排序

输入: 数组 $A[1..n]$，其中 $A[j] \in \{1, 2, ..., k\}$
输出: 排序后的数组 $B[1, ..., n]$

$C \leftarrow []$
for $i \leftarrow 1$ *to* k **do**
| $C[i] \leftarrow 0;$
end
for $j \leftarrow 1 ton$ **do**
| $C[A[j]] \leftarrow C[A[j]] + 1;$ $//C[i] = |\{key = i\}|$
end
for $i \leftarrow 2$ *to* k **do**
| $C[i] \leftarrow C[i] + C[i-1];$ $//C[i] = |\{key \leq i\}|$
end
for $j \leftarrow n$ *to* 1 **do**
| $B[C[A[j]]] \leftarrow A[j];$
| $C[A[j]] \leftarrow C[A[j]] - 1;$
end
return $B;$

将A中的元素依据C记录的位置从后向前放入输出数组

运行实例：计数排序



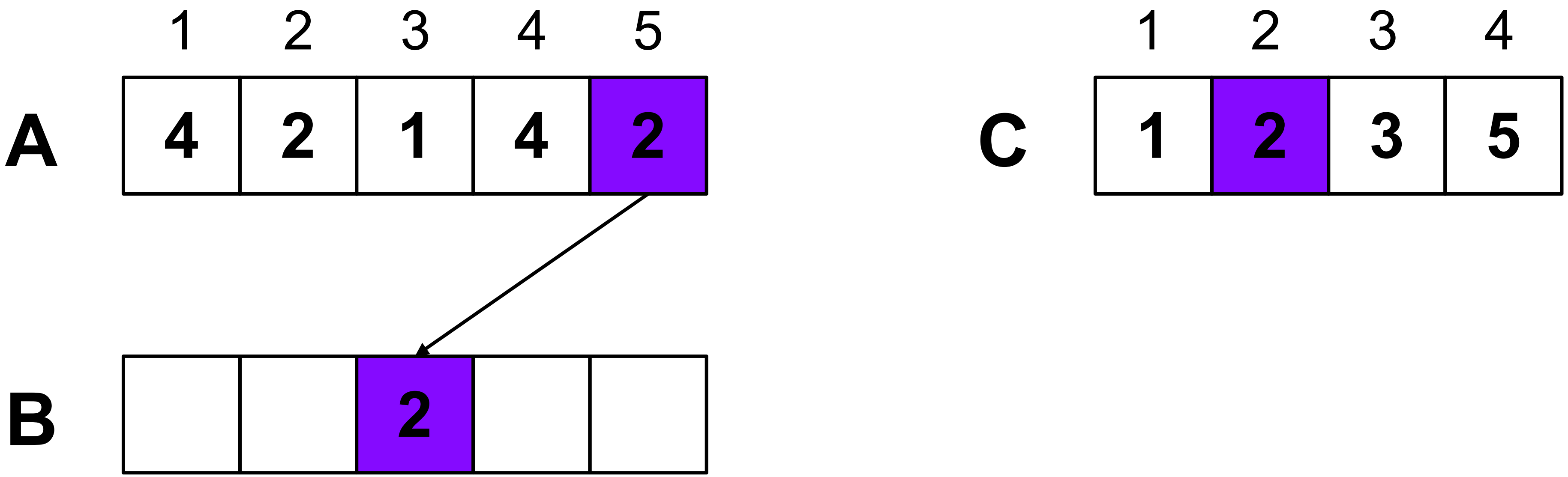
for $j \leftarrow n$ *to* 1 **do**

$\quad \boxed{B[C[A[j]]] \leftarrow A[j];}$

$\quad C[A[j]] \leftarrow C[A[j]] - 1;$

end

运行实例：计数排序



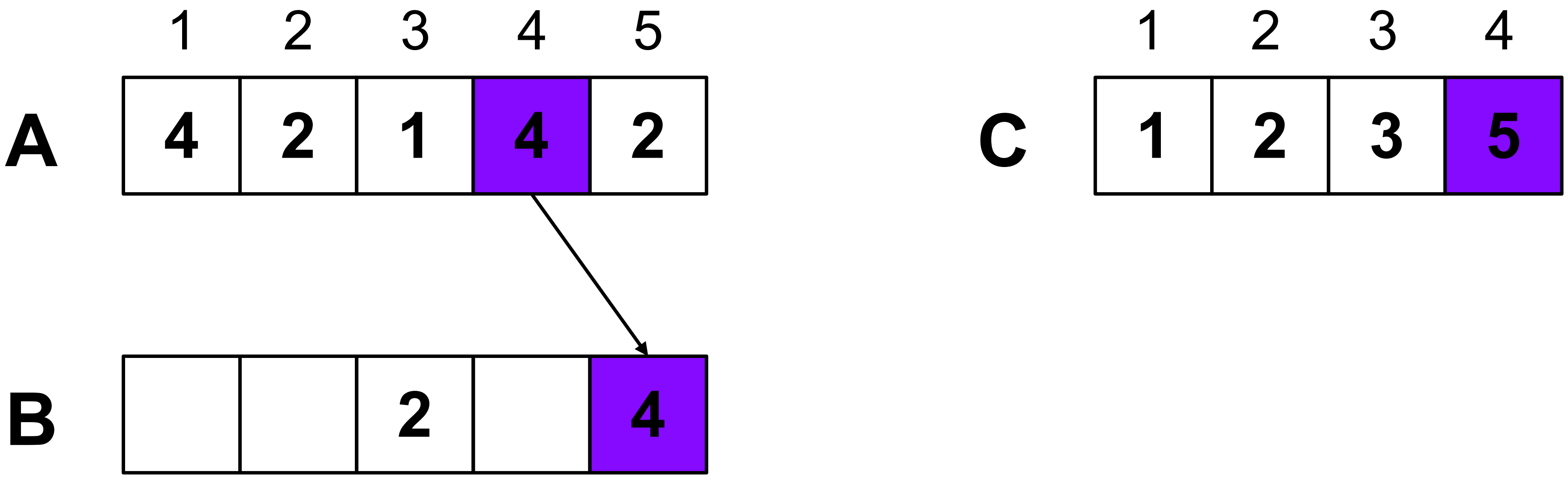
for $j \leftarrow n$ *to* 1 **do**

$\quad B[C[A[j]]] \leftarrow A[j];$

$\quad \boxed{C[A[j]] \leftarrow C[A[j]] - 1;}$

end

运行实例：计数排序



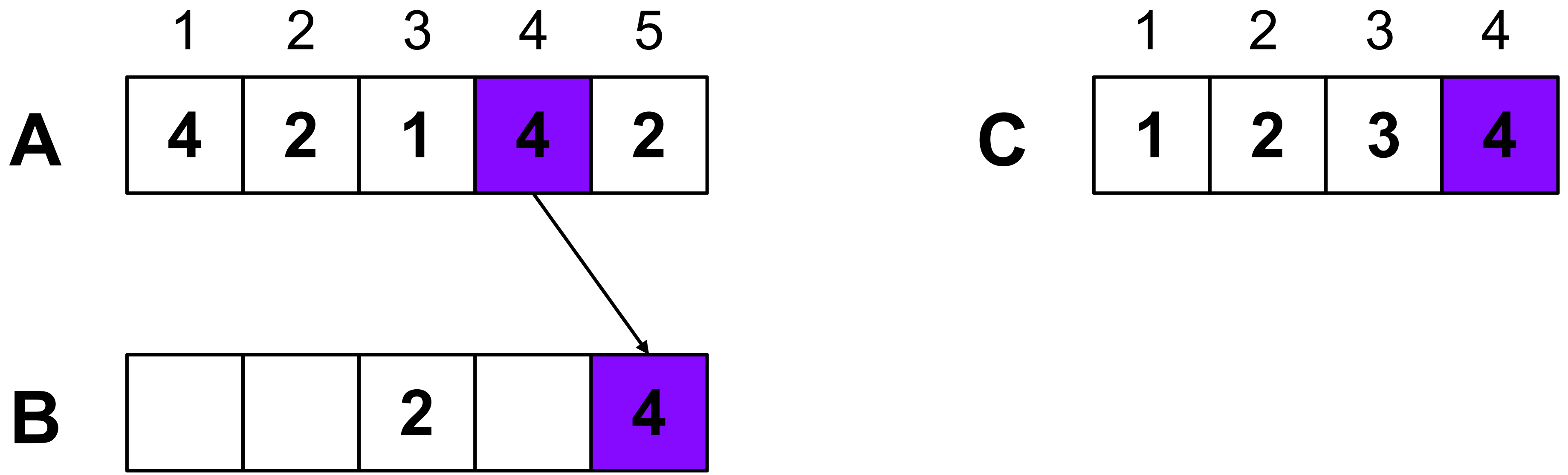
for $j \leftarrow n$ *to* 1 **do**

$\boxed{B[C[A[j]]] \leftarrow A[j];}$

$C[A[j]] \leftarrow C[A[j]] - 1;$

end

运行实例：计数排序



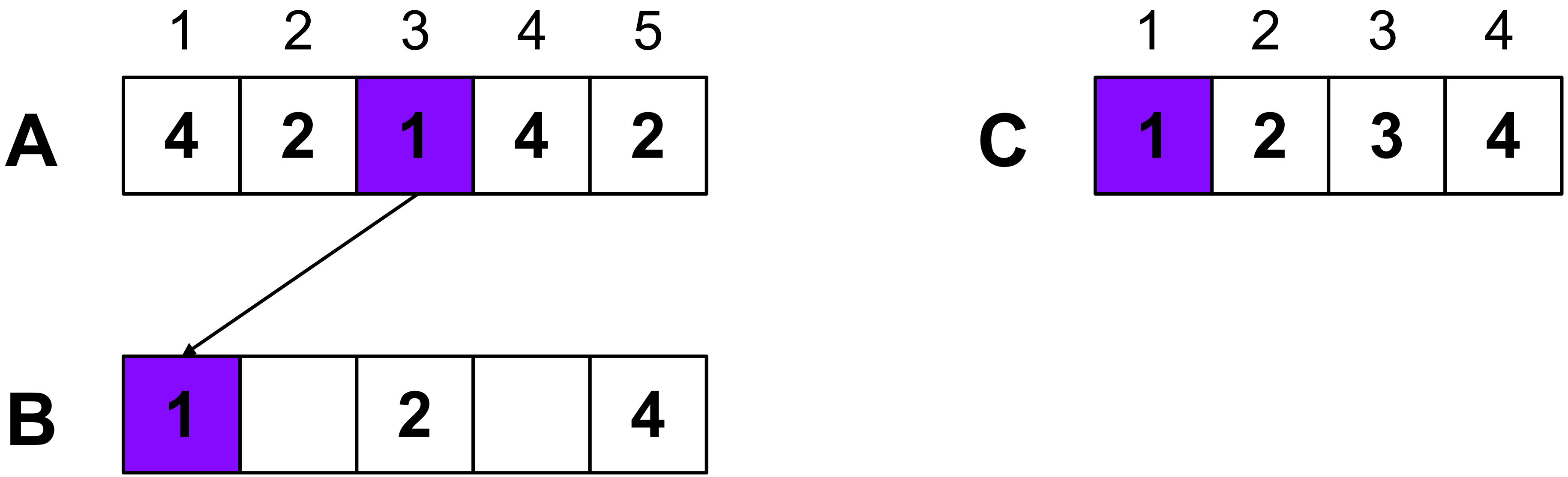
for $j \leftarrow n$ *to* 1 **do**

$\quad B[C[A[j]]] \leftarrow A[j];$

$\quad \boxed{C[A[j]] \leftarrow C[A[j]] - 1;}$

end

运行实例：计数排序



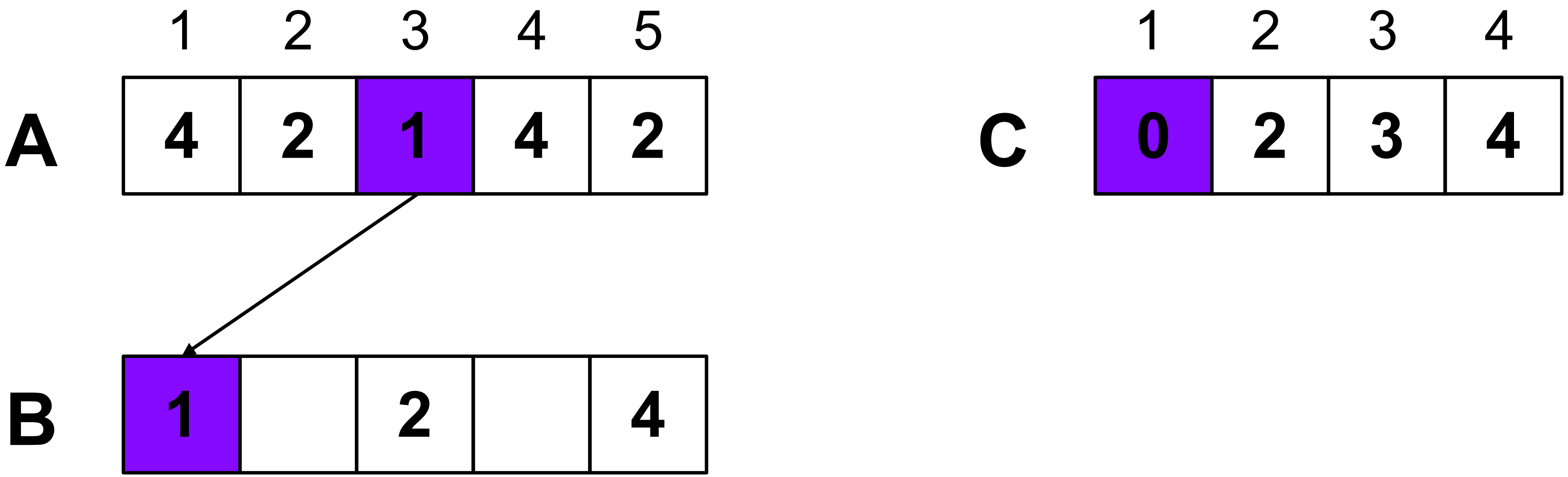
for $j \leftarrow n$ *to* 1 **do**

$\quad \boxed{B[C[A[j]]] \leftarrow A[j];}$

$\quad C[A[j]] \leftarrow C[A[j]] - 1;$

end

运行实例：计数排序



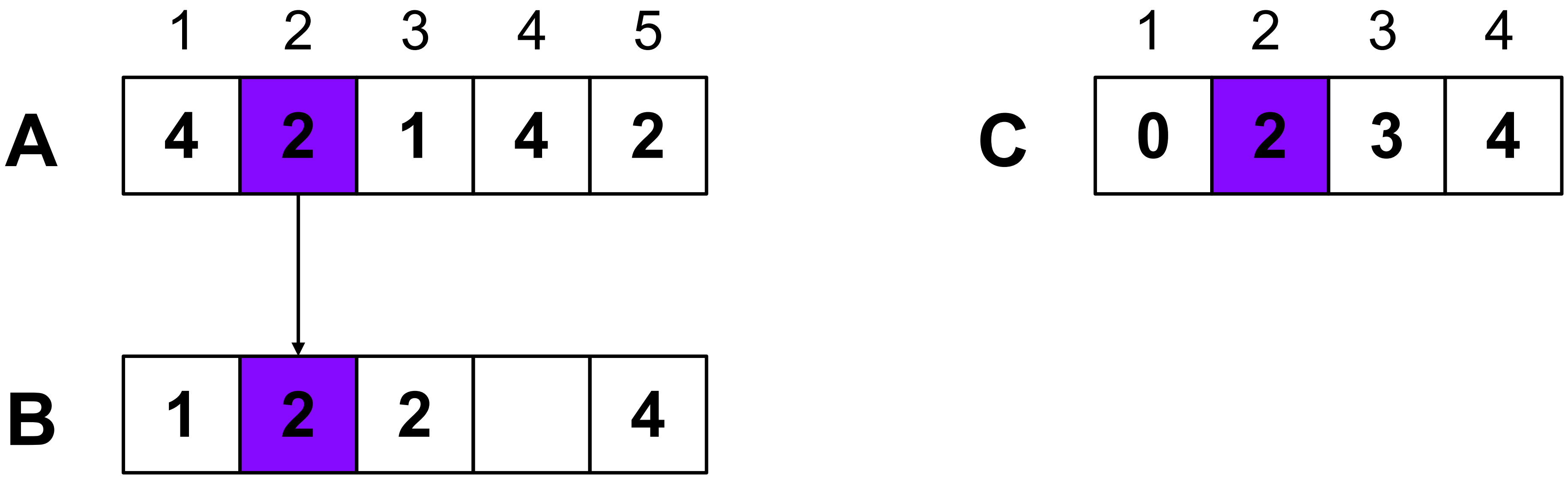
for $j \leftarrow n$ *to* 1 **do**

$\quad B[C[A[j]]] \leftarrow A[j];$

$\quad \boxed{C[A[j]] \leftarrow C[A[j]] - 1;}$

end

运行实例：计数排序



for $j \leftarrow n$ *to* 1 **do**

$\boxed{B[C[A[j]]] \leftarrow A[j];}$

$C[A[j]] \leftarrow C[A[j]] - 1;$

end

运行实例：计数排序

	1	2	3	4	5
A	**4**	**2**	**1**	**4**	**2**
B	**1**	**2**	**2**		**4**

	1	2	3	4
C	**0**	**1**	**3**	**4**

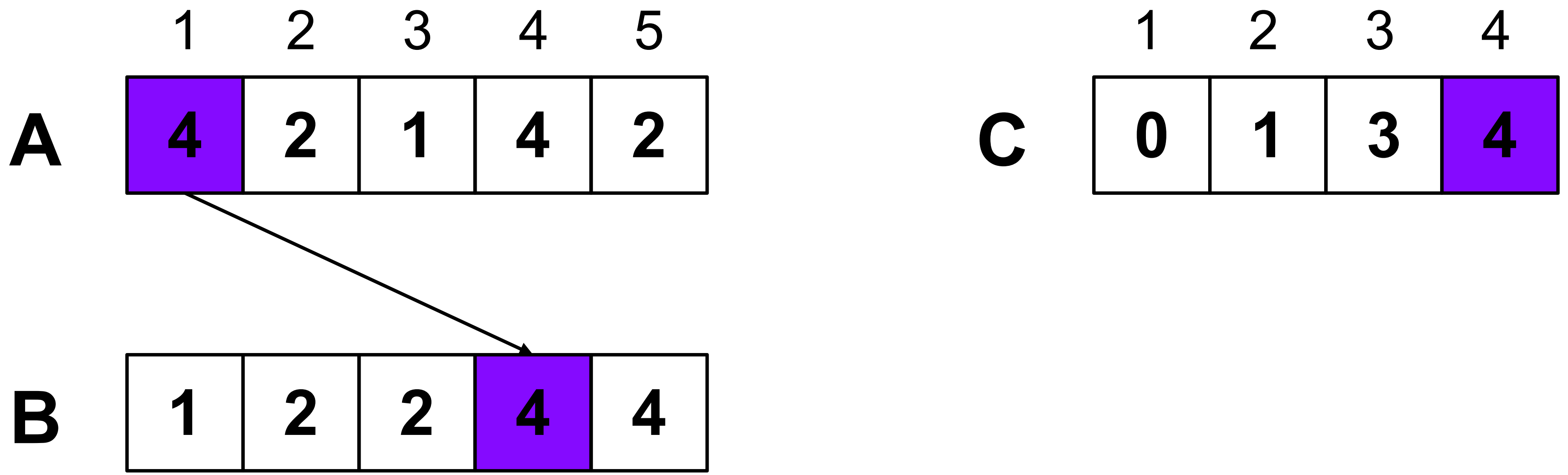
for $j \leftarrow n$ *to* 1 **do**

　$B[C[A[j]]] \leftarrow A[j]$;

　$\boxed{C[A[j]] \leftarrow C[A[j]] - 1;}$

end

运行实例：计数排序



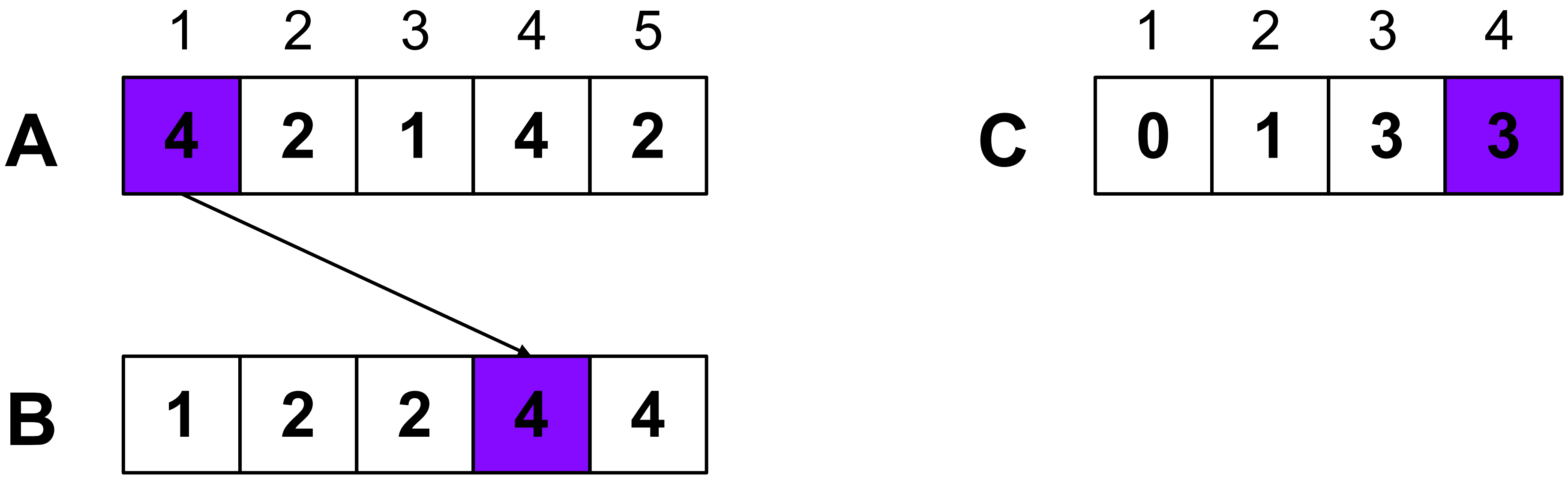
for $j \leftarrow n$ *to* 1 **do**

$\quad \boxed{B[C[A[j]]] \leftarrow A[j];}$

$\quad C[A[j]] \leftarrow C[A[j]] - 1;$

end

运行实例：计数排序



for $j \leftarrow n$ *to* 1 **do**

  $B[C[A[j]]] \leftarrow A[j];$

  $\boxed{C[A[j]] \leftarrow C[A[j]] - 1;}$

end

分析

计数排序

输入: 数组$A[1..n]$,其中$A[j] \in \{1, 2, ..., k\}$
输出: 排序后的数组 $B[1, ..., n]$
$C \leftarrow []$
for $i \leftarrow 1$ *to* k **do**
  $\boxed{C[i] \leftarrow 0;}$　　$O(k)$
end

分析

计数排序

输入: 数组$A[1..n]$，其中$A[j] \in \{1, 2, ..., k\}$
输出: 排序后的数组 $B[1, ..., n]$

$C \leftarrow []$
for $i \leftarrow 1$ *to* k **do**
| $\boxed{C[i] \leftarrow 0;}$ $O(k)$
end
for $j \leftarrow 1 to n$ **do**
| $\boxed{C[A[j]] \leftarrow C[A[j]] + 1;}$ $O(n)$
end

分析

计数排序

输入: 数组$A[1..n]$,其中$A[j] \in \{1,2,\ldots,k\}$
输出: 排序后的数组 $B[1,\ldots,n]$

$C \leftarrow []$
for $i \leftarrow 1$ *to* k **do**
| $\boxed{C[i] \leftarrow 0;}$ $O(k)$
end
for $j \leftarrow 1$ *to* n **do**
| $\boxed{C[A[j]] \leftarrow C[A[j]] + 1;}$ $O(n)$
end
for $i \leftarrow 2$ *to* k **do**
| $\boxed{C[i] \leftarrow C[i] + C[i-1];}$ $O(k)$
end

分析

计数排序

输入: 数组 $A[1..n]$，其中 $A[j] \in \{1, 2, ..., k\}$
输出: 排序后的数组 $B[1, ..., n]$

$C \leftarrow []$
for $i \leftarrow 1$ *to* k **do**
  $C[i] \leftarrow 0;$ $O(k)$
end
for $j \leftarrow 1$ *to* n **do**
  $C[A[j]] \leftarrow C[A[j]] + 1;$ $O(n)$
end
for $i \leftarrow 2$ *to* k **do**
  $C[i] \leftarrow C[i] + C[i-1];$ $O(k)$
end
for $j \leftarrow n$ *to* 1 **do**
  $B[C[A[j]]] \leftarrow A[j];$
  $C[A[j]] \leftarrow C[A[j]] - 1;$ $O(n)$
end
return $B;$

分析

计数排序

输入: 数组 $A[1..n]$ **, 其中** $A[j] \in \{1, 2, ..., k\}$
输出: 排序后的数组 $B[1, ..., n]$

```
C ← []
for i ← 1 to k do
|  C[i] ← 0;                              O(k)
end
for j← 1ton do
|  C[A[j]] ← C[A[j]] + 1;                 O(n)
end
for i ← 2 to k do
|  C[i] ← C[i] + C[i − 1];                O(k)
end
for j← n to 1 do
|  B[C[A[j]]] ← A[j];
|  C[A[j]] ← C[A[j]] − 1;                 O(n)
end
return B;
```

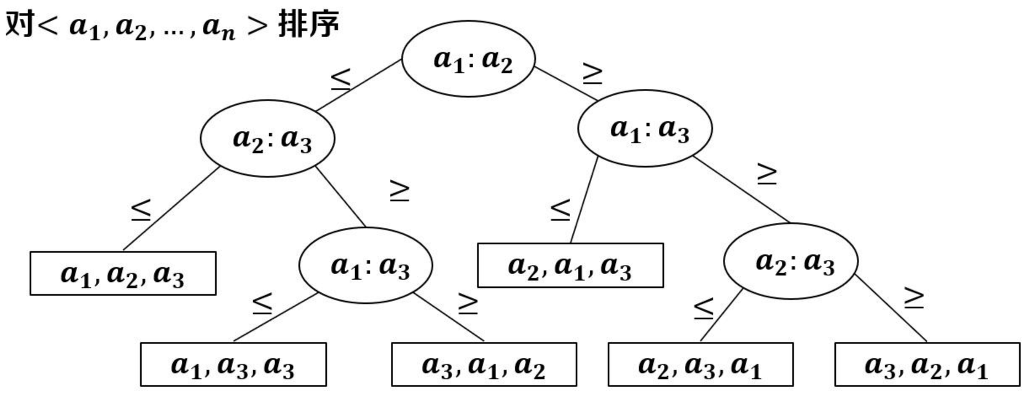
总的时间复杂度：$O(n+k)$

运行时间

若 $\boldsymbol{k}=\boldsymbol{O}(\boldsymbol{n})$，则计数排序需要 $\boldsymbol{O}(\boldsymbol{n})$ 的时间

- 决策树模型证明排序时间的下界是 $\boldsymbol{\Omega}(\boldsymbol{n}\log\boldsymbol{n})$?
- 决策树模型证明的是基于比较的排序算法下界是 $\boldsymbol{\Omega}(\boldsymbol{n}\log\boldsymbol{n})$!

基于比较的排序算法



计数排序不基于比较