



Data Structures

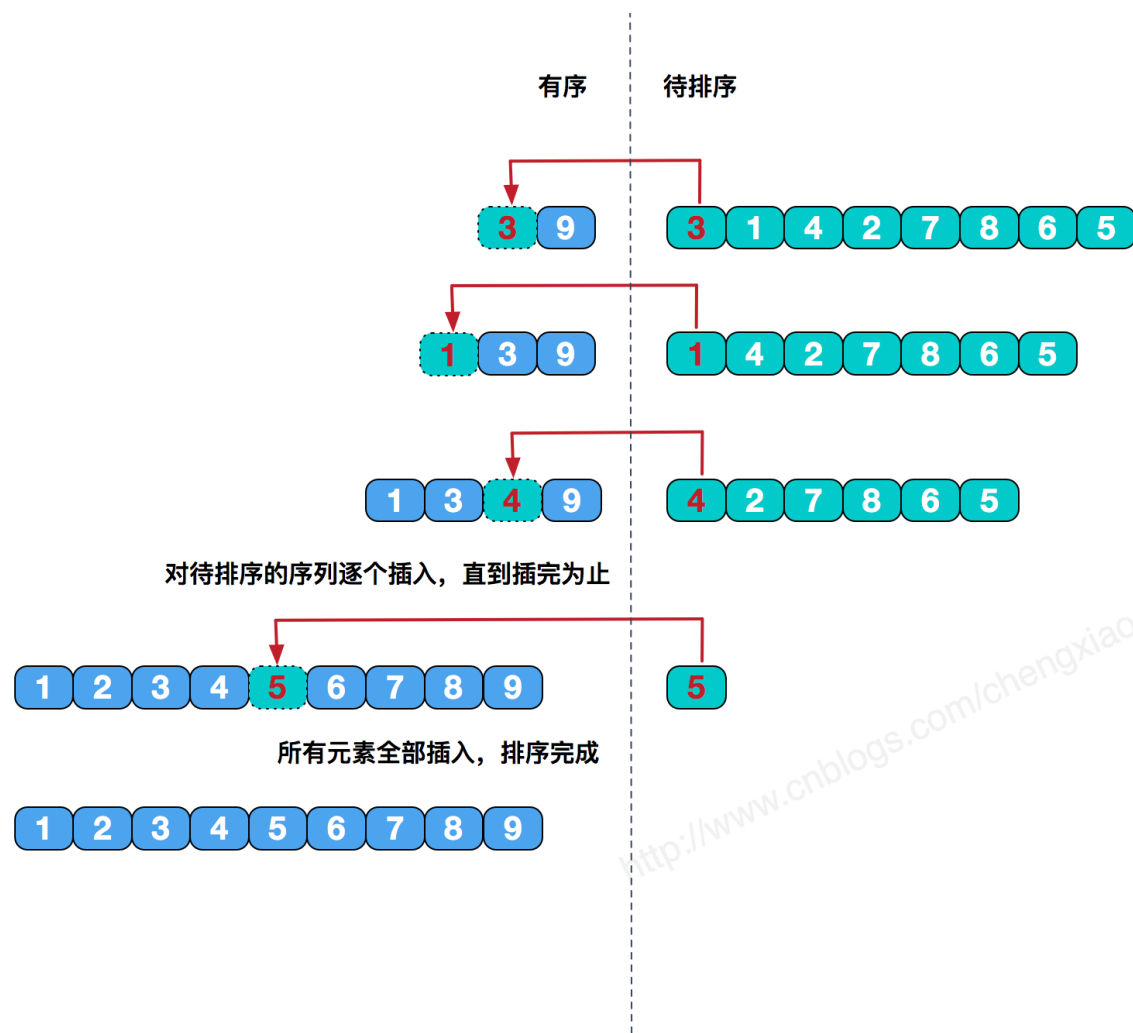
Ch8

排序 Sort

2024 年 12 月 17 日

- ➡ 8.1 概述
- ➡ 8.2 插入排序：直接插入排序、希尔排序
- ➡ 8.3 交换排序：起泡、快速排序
- ➡ **8.4 选择排序：简单选择、堆排序**
- ➡ 8.5 归并排序：二路归并排序
- ➡ 8.6 各种排序方法比较
- ➡ 8.7 扩展与提高

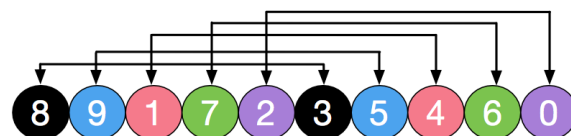
回顾



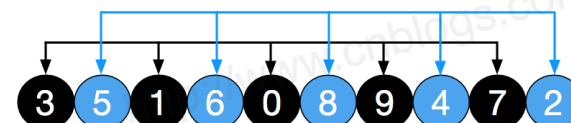
原始数组 以下数据元素颜色相同为一组



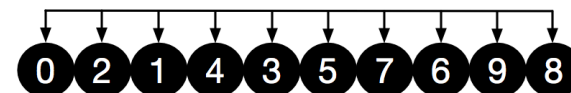
初始增量 $gap=length/2=5$ ，意味着整个数组被分为5组，[8,3] [9,5] [1,4] [7,6] [2,0]



对这5组分别进行直接插入排序，结果如下，可以看到，像3，5，6这些小元素都被调到前面了，然后缩小增量 $gap=5/2=2$ ，数组被分为2组 [3,1,0,9,7] [5,6,8,4,2]



对以上2组再分别进行直接插入排序，结果如下，可以看到，此时整个数组的有序程度更进一步啦。再缩小增量 $gap=2/2=1$ ，此时，整个数组为1组[0,2,1,4,3,5,7,6,9,8]，如下

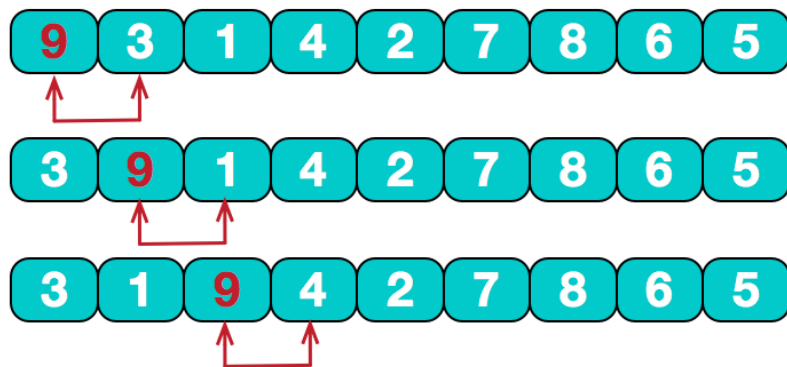


经过上面的“宏观调控”，整个数组的有序化程度成果喜人。此时，仅仅需要对以上数列简单微调，无需大量移动操作即可完成整个数组的排序。



回顾

相邻元素两两比较，反序则交换



第一轮完毕，将最大元素9浮到数组顶端



同理,第二轮将第二大元素8浮到数组顶端



排序完成

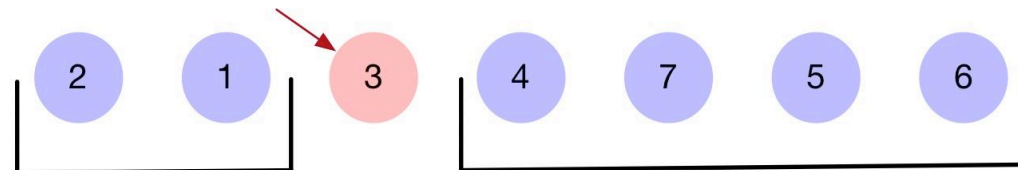


待排序数组 array



切分元素 Partition Element

切分



\leq Partition Element

\geq Partition Element

左半部分排序



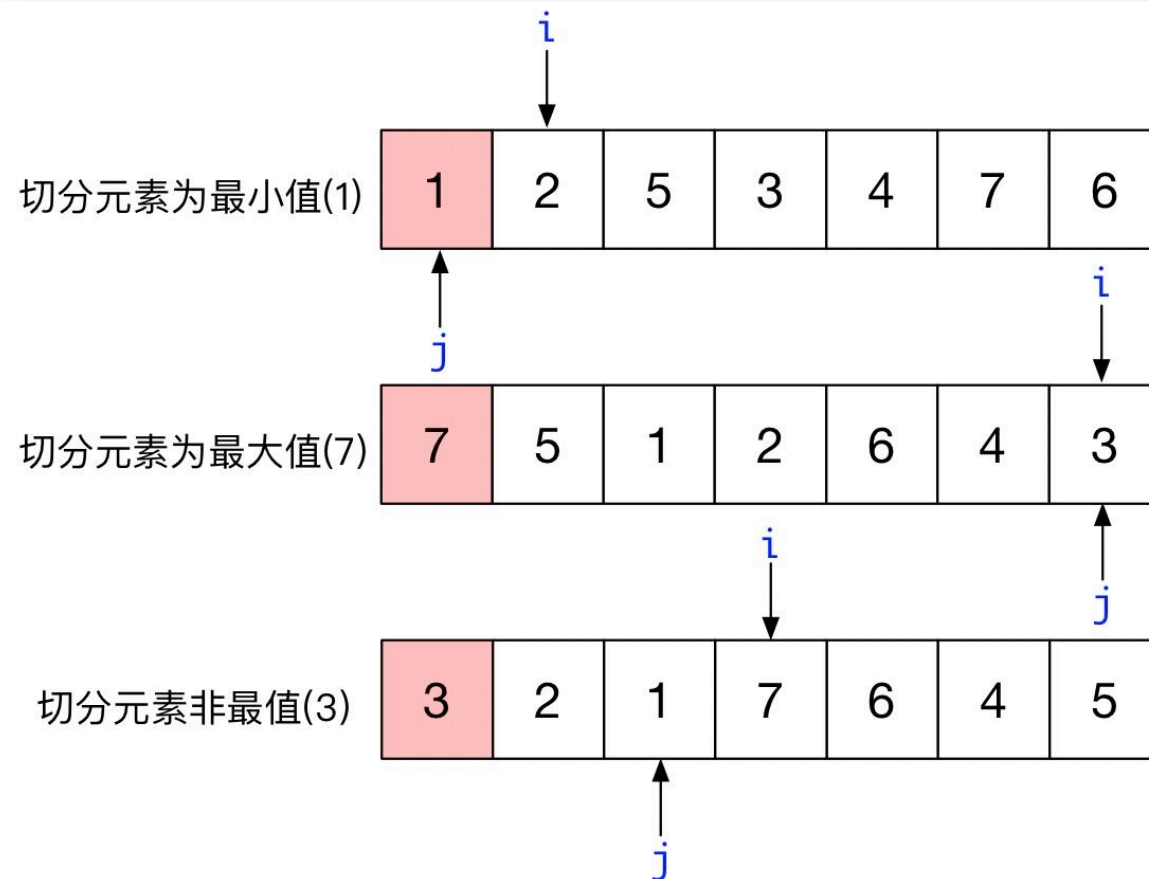
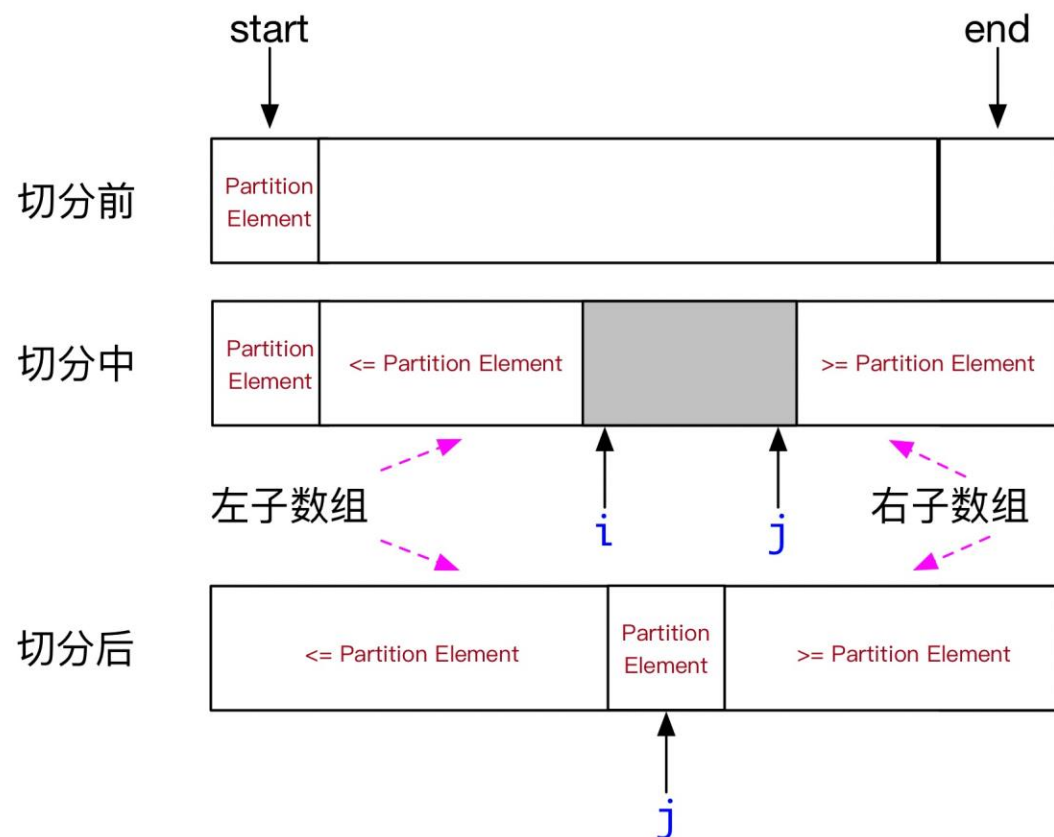
右半部分排序



有序



回顾



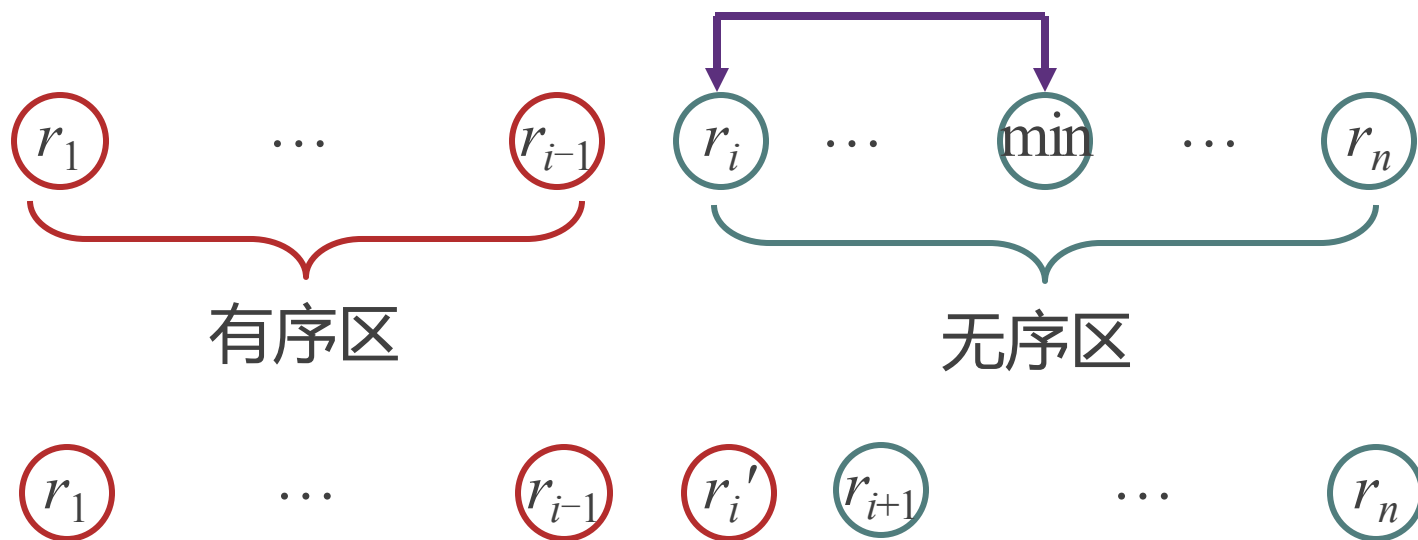
8.4 选择排序

8-4-1 简单选择排序



1. 简单选择排序

📌 简单选择排序的基本思想：第 i 趟 ($1 \leq i \leq n-1$) 排序在待排序序列 $r[i] \sim r[n]$ 中选取最小记录，并和第 i 个记录交换。



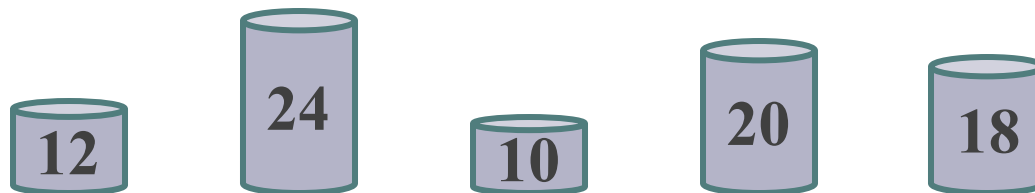
8.4 选择排序

8-4-1 简单选择排序



1. 简单选择排序

待排序序列



第一趟排序结果



第二趟排序结果



第三趟排序结果



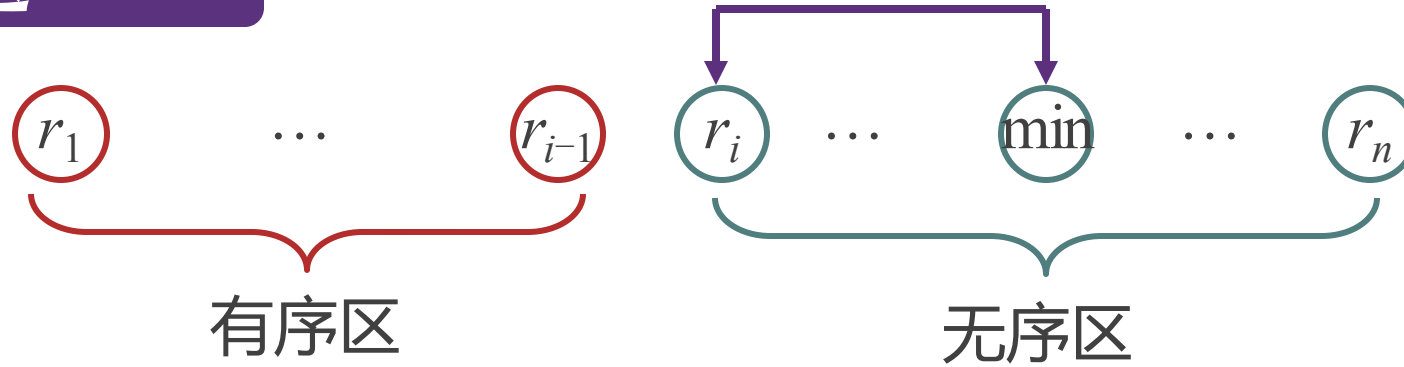
第四趟排序结果



8.4 选择排序

8-4-1 简单选择排序

2. 关键问题



算法描述：

```
for (i = 0; i < length-1; i++)  
{  
    第 i 趟简单选择排序;  
}
```

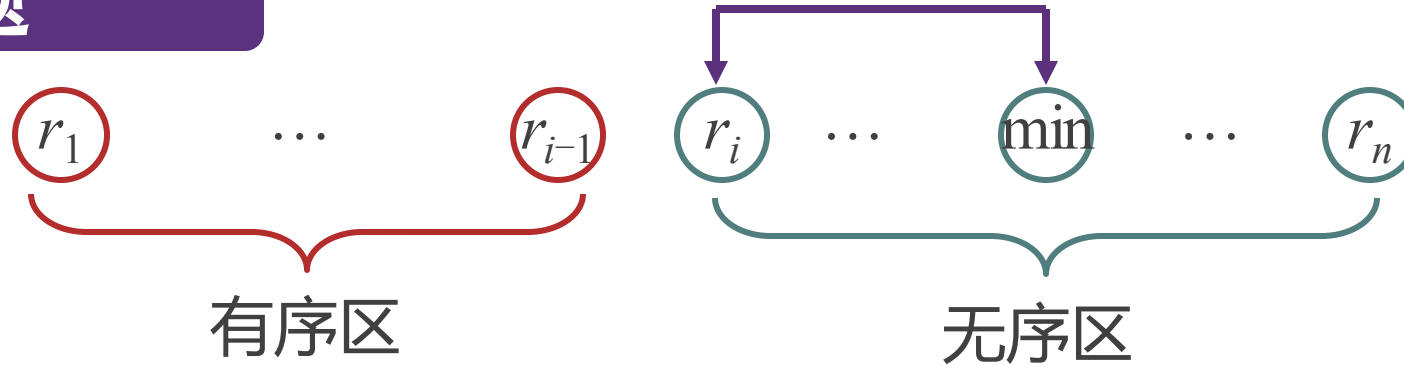


简单选择排序进行多少趟？ $n-1$ 趟

8.4 选择排序

8-4-1 简单选择排序

2. 关键问题



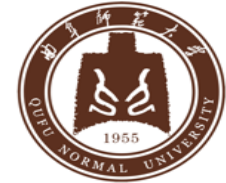
算法描述:

```
index = i;  
for (j = i + 1; j < length; j++)  
    if (data[j] < data[index]) index = j;  
if (index != i) {  
    交换data[i]和data[index];  
}
```



第 i 趟简单选择排序完成什么工作?

- (1) 在 $r[i] \sim r[n]$ 中找最小值
- (2) 将最小记录与 $r[i]$ 交换



3. 算法描述

```
void Sort :: SelectSort( )
{
    int i, j, index, temp;
    for (i = 0; i < length; i++)
    {
        index = i;
        for (j = i + 1; j < length-1; j++)
            if (data[j] < data[index]) index = j;
        if (index != i) {
            temp = data[i]; data[i] = data[index]; data[index] = temp;
        }
    }
}
```



交换语句之前的判断与效率有什么关系？



4. 时间性能分析

```
void Sort :: SelectSort( )
{
    int i, j, index, temp;
    for (i = 0; i < length; i++)
    {
        index = i;
        for (j = i + 1; j < length-1; j++)
            if (data[j] < data[index]) index = j;
        if (index != i) {
            temp = data[i]; data[i] = data[index]; data[index] = temp;
        }
    }
}
```



比较语句？ 执行次数？

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^n 1 = (n-1 + \dots + 2 + 1) = \frac{n(n-1)}{2}$$



移动语句？ 执行次数？

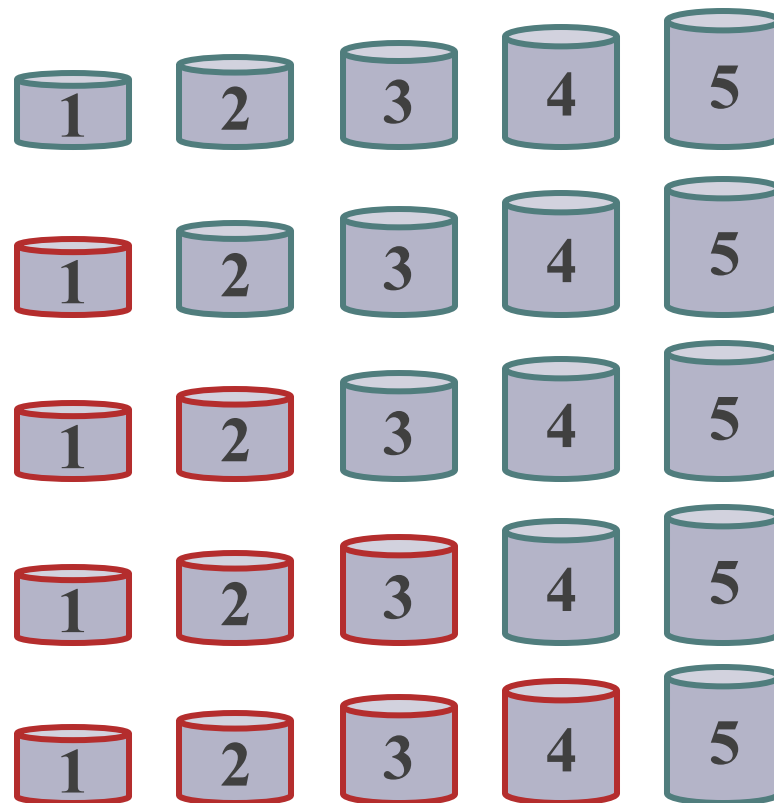
4. 时间性能分析

📜 比较次数: $O(n^2)$

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^n 1 = (n-1 + \dots + 2 + 1) = \frac{n(n-1)}{2}$$

📜 移动次数:

📎 最好情况: 0 次



8.4 选择排序

8-4-1 简单选择排序



4. 时间性能分析

📜 比较次数: $O(n^2)$

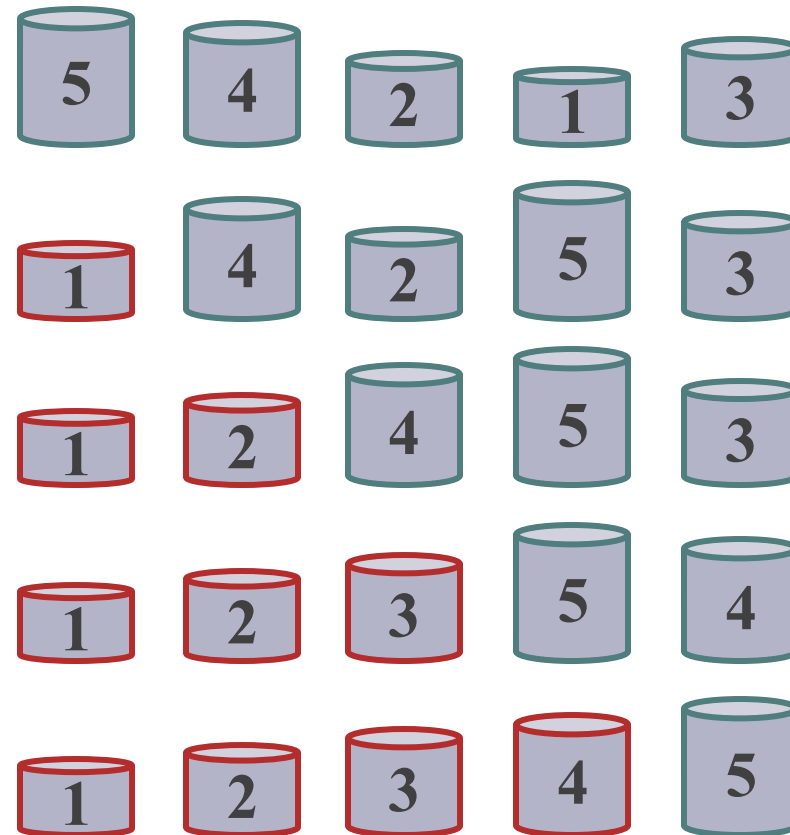
$$\sum_{i=1}^{n-1} \sum_{j=i+1}^n 1 = (n-1 + \cdots + 2 + 1) = \frac{n(n-1)}{2}$$

📜 移动次数:

📎 最好情况: 0 次

📎 最坏情况: $3(n-1)$ 次

📜 最好、最坏、平均情况: $O(n^2)$



8.4 选择排序

8-4-1 简单选择排序

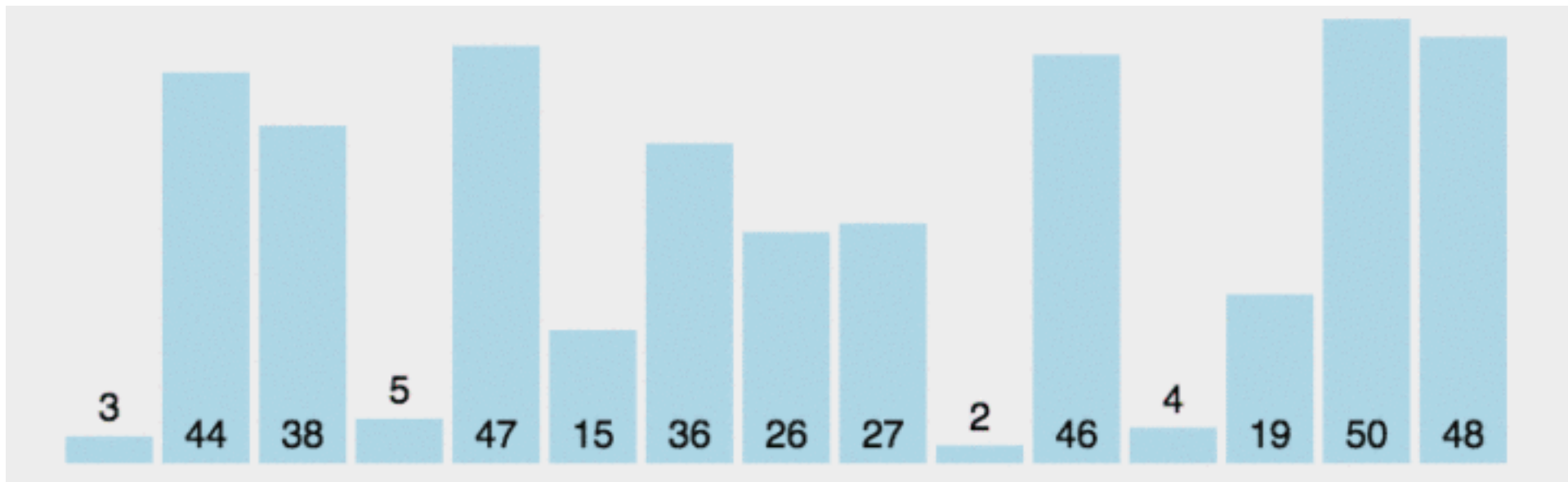


5. 空间性能分析

📜 空间性能: $O(1)$



📜 稳定性: 不稳定



8.4 选择排序

8-4-2 堆排序

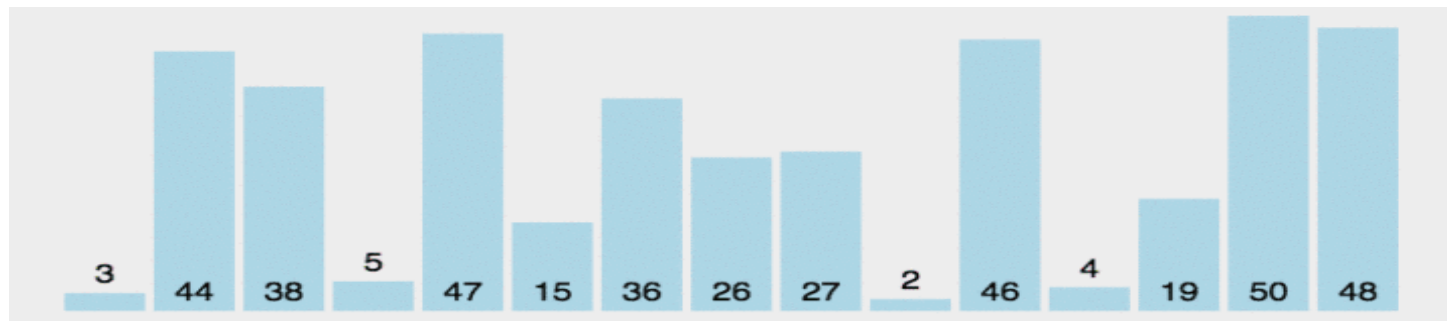
Heap Sort

8.4 选择排序

8-4-2 堆排序



简单选择排序



 **缺点：**简单选择排序的时间主要耗费在哪了呢？

 对无序序列扫描一趟（ $n-1$ 次比较）只做了一件事——找最小值

 **优点：**移动次数较少，最坏情况 $O(n)$

提高整个排序的效率

减少后面选择所用的比较次数



利用每趟比较后的结果

查找最小值的同时找出并保存较小值



8.4 选择排序

8-4-2 堆排序

1. 堆的定义

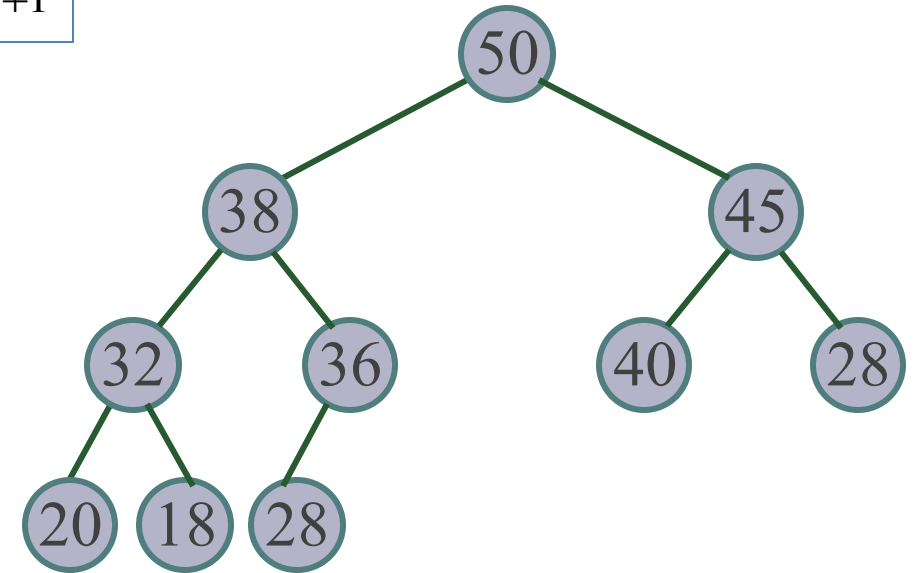
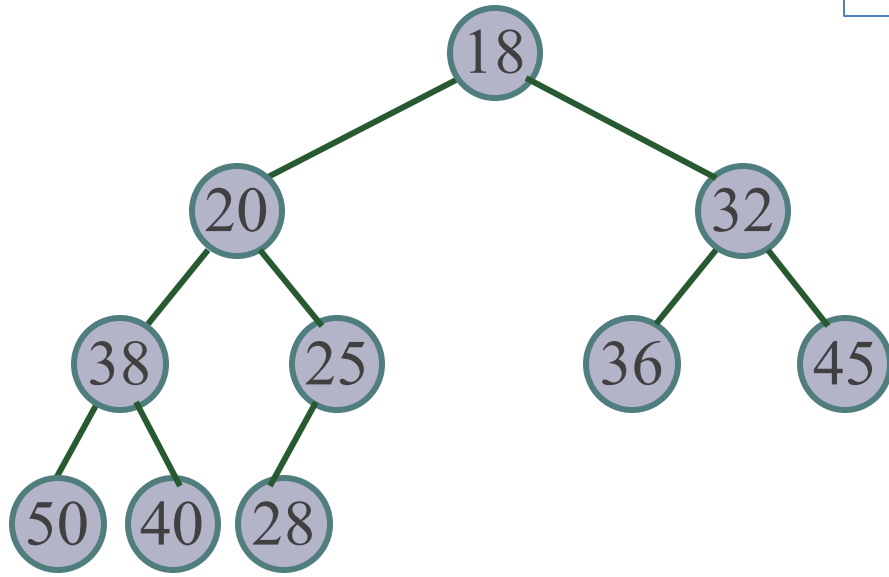
n 个元素的序列 $\{k_1, k_2, \dots, k_n\}$, 当且仅当满足下列关系时, 称为堆:

小根堆: 每个结点的值都**小于等于**其左右孩子结点的**完全二叉树**。

$$\begin{cases} k_i \leq k_{2i} \\ k_i \leq k_{2i+1} \end{cases} \text{ 或 } \begin{cases} k_i \geq k_{2i} \\ k_i \geq k_{2i+1} \end{cases}$$

大根堆: 每个结点的值都**大于等于**其左右孩子结点的**完全二叉树**。

小根堆和大根堆统称为堆。



(1) 根结点 (称为**堆顶**) 的值是所有结点的最大值;

(2) 较大值的结点靠近根结点, 但不绝对。

8.4 选择排序

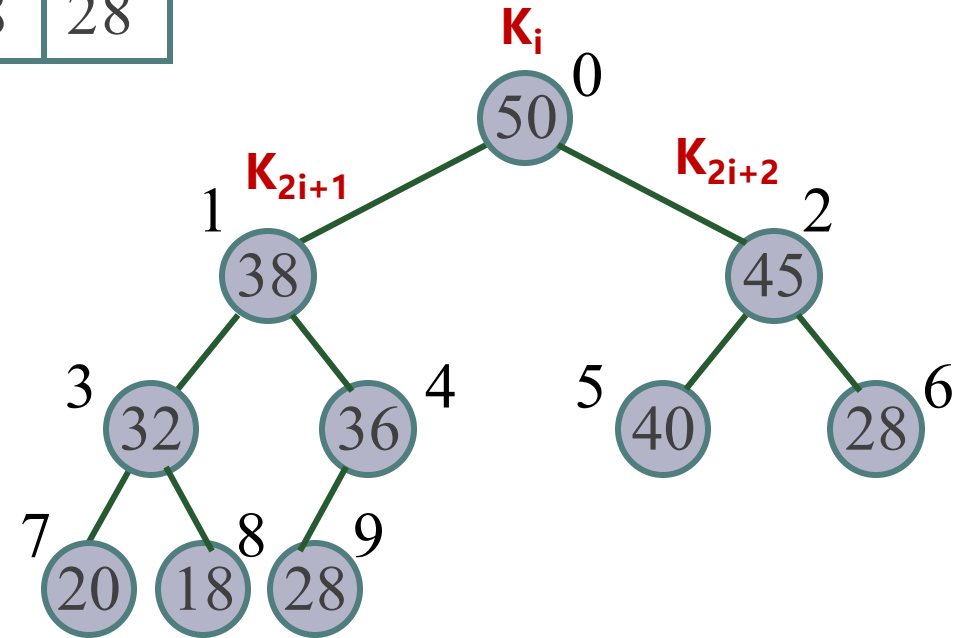
8-4-2 堆排序

2. 堆与序列的关系

📎 堆采用顺序存储，则对应一个（**无序**）序列

0	1	2	3	4	5	6	7	8	9
50	38	45	32	36	40	28	20	18	28

↑ 顺序存储，
以编号作为下标



8.4 选择排序

8-4-2 堆排序

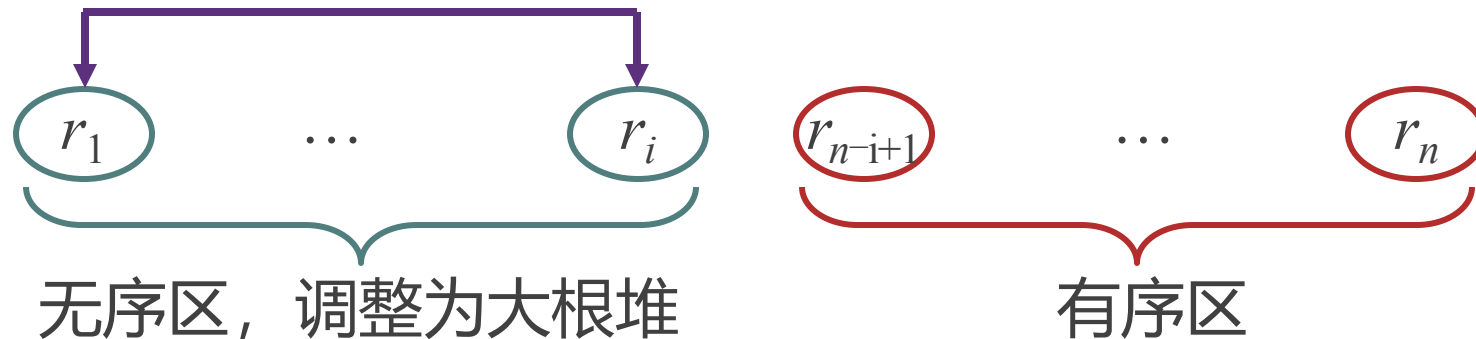
3. 堆排序的基本思想

基本思想：

- ✓ 将无序序列**建成**一个堆
- ✓ 输出**堆顶**的最小（大）值
- ✓ 使剩余的 $n-1$ 个元素又**调整**成一个堆，则可得到 n 个元素的次小值
- ✓ **重复**执行，得到一个有序序列

如何建？

如何调整？



第 i 趟堆排序将 $r_1 \sim r_i$ 调整成大根堆，再将堆顶与 r_i 交换

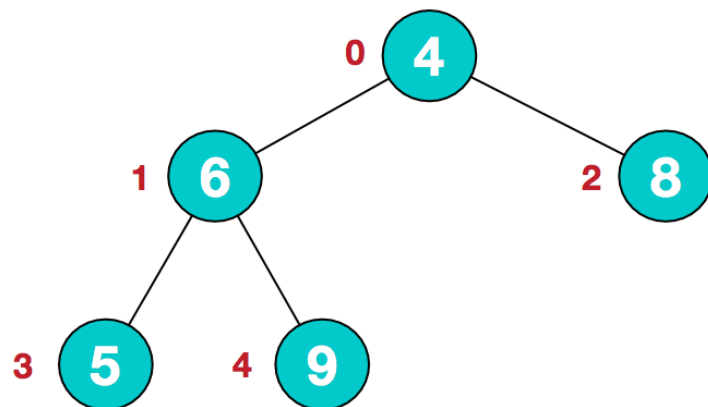


3.1 构造初始堆

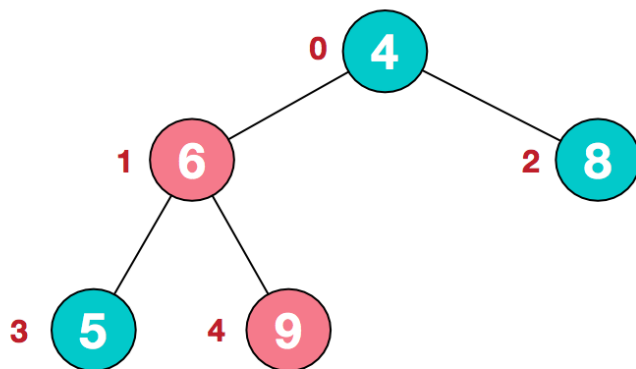
步骤一 构造初始堆。 将给定无序序列构造成一个大顶堆（一般升序采用大顶堆，降序采用小顶堆）。

给定无序序列

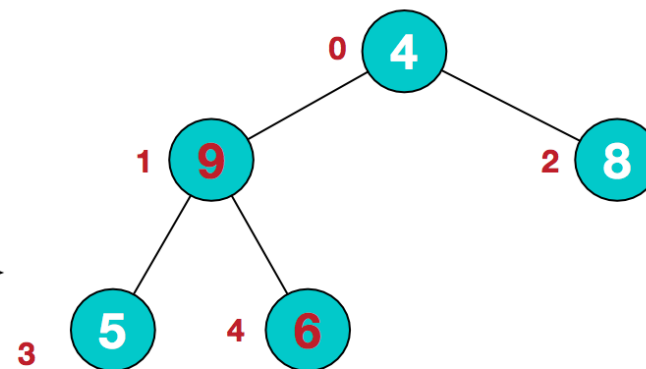
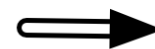
Step1: 从最后一个非叶子结点开始从下至上进行调整



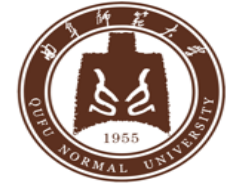
	0	1	2	3	4
arr	4	6	8	5	9



	0	1	2	3	4
arr	4	6	8	5	9



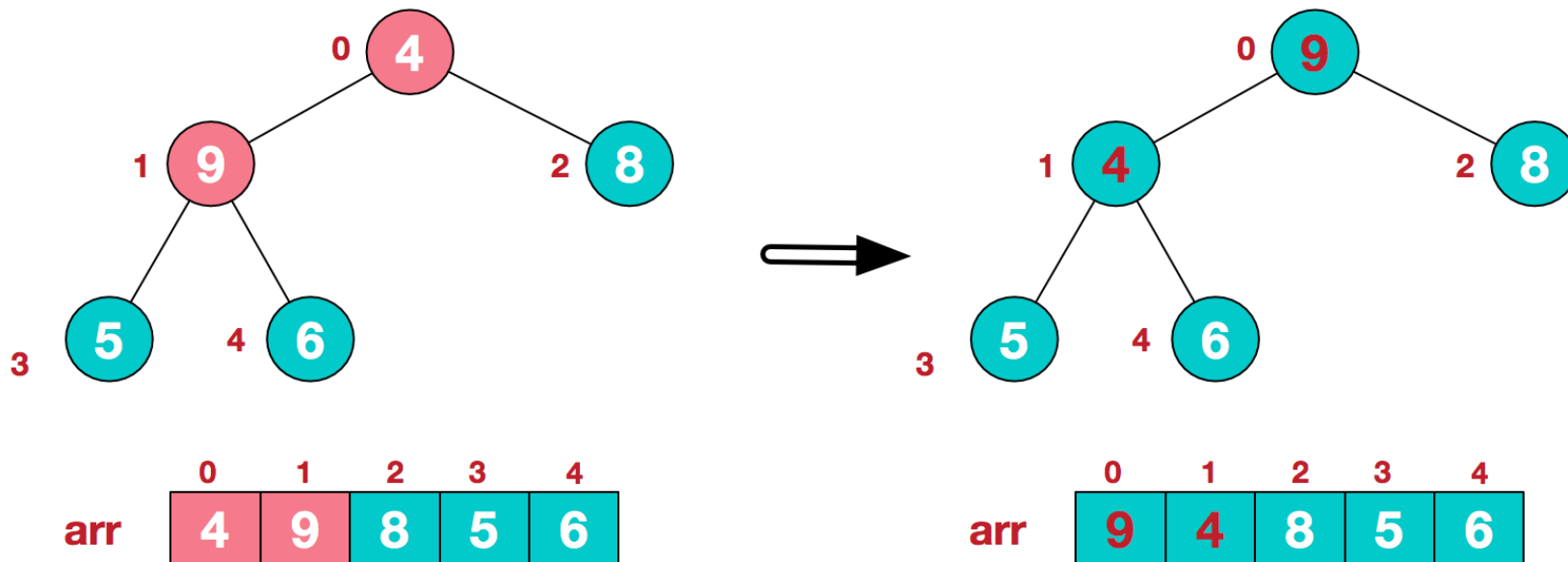
	0	1	2	3	4
arr	4	9	8	5	6

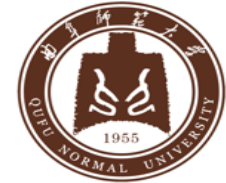


3.1 构造初始堆

步骤一 构造初始堆。 将给定无序序列构造成一个大顶堆（一般升序采用大顶堆，降序采用小顶堆）。

Step2: 找到第二个非叶节点4，由于[4,9,8]中9元素最大，4和9交换。

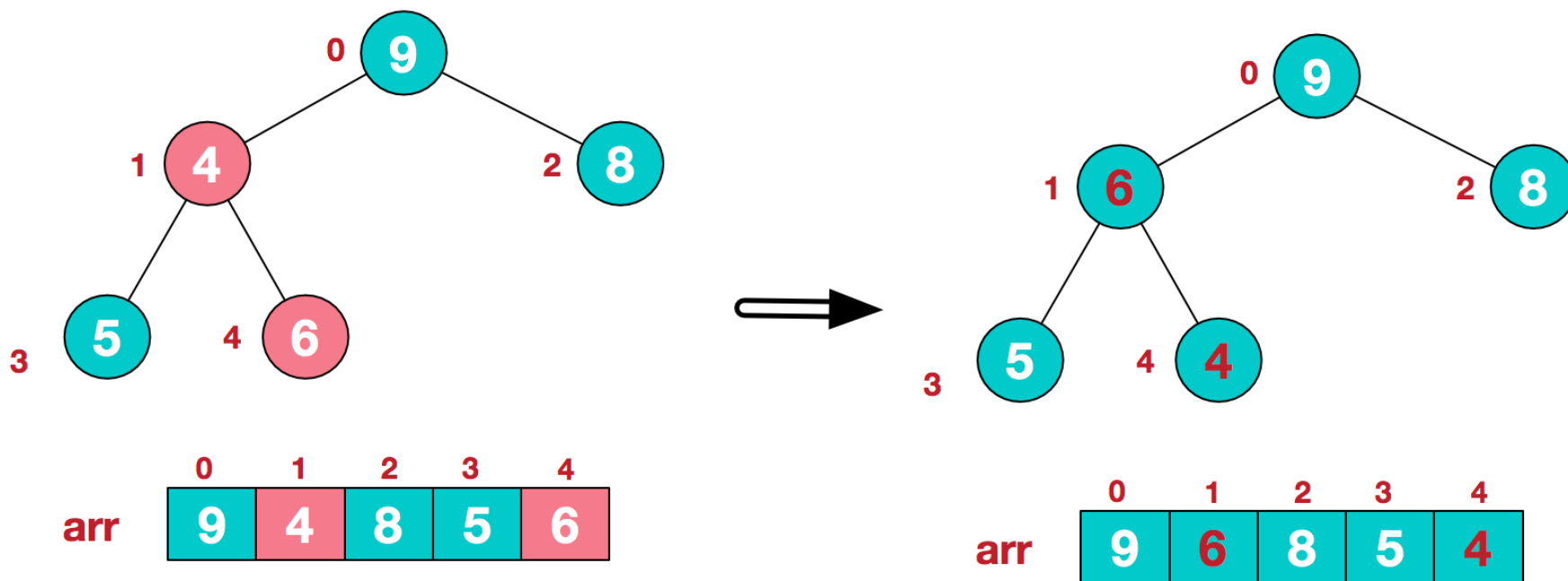




3.1 构造初始堆

步骤一 构造初始堆。 将给定无序序列构造成一个大顶堆（一般升序采用大顶堆，降序采用小顶堆）。

Step3: 交换导致了子根[4,5,6]结构混乱，继续调整，[4,5,6]中6最大，交换4和6。

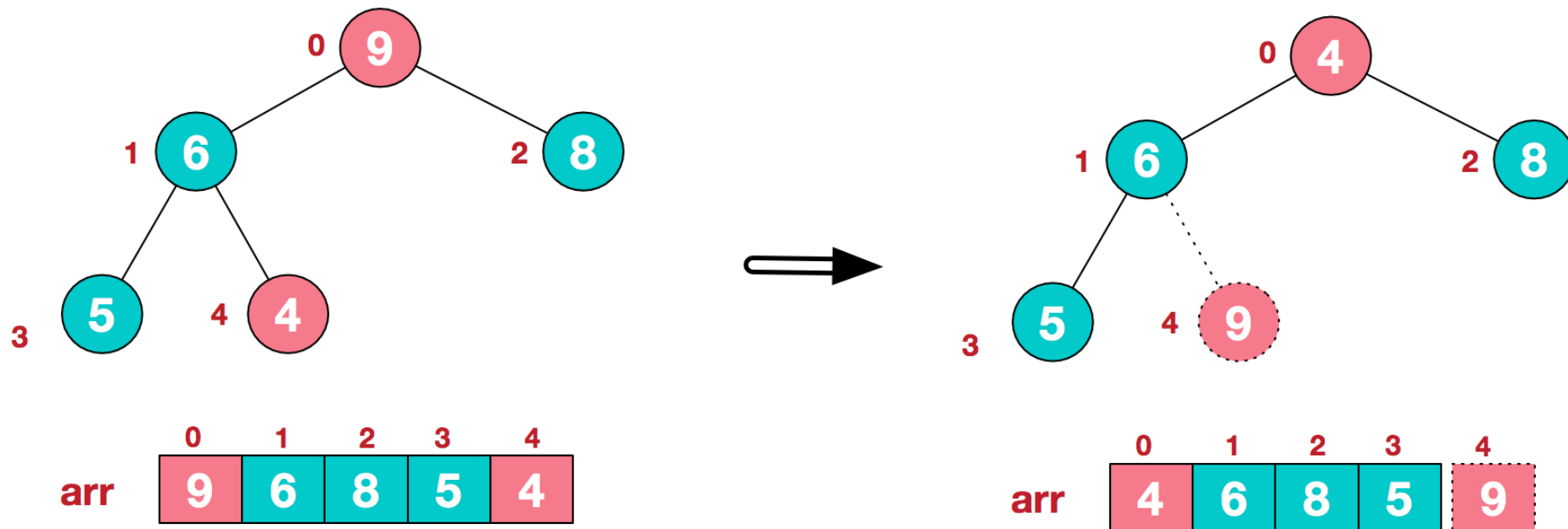




3.2 堆调整

步骤二 将堆顶元素与末尾元素进行交换，使末尾元素最大。然后继续调整堆，再将堆顶元素与末尾元素交换，得到第二大元素。如此反复进行交换、重建、交换。

Step1: 将堆顶元素9和末尾元素4进行交换

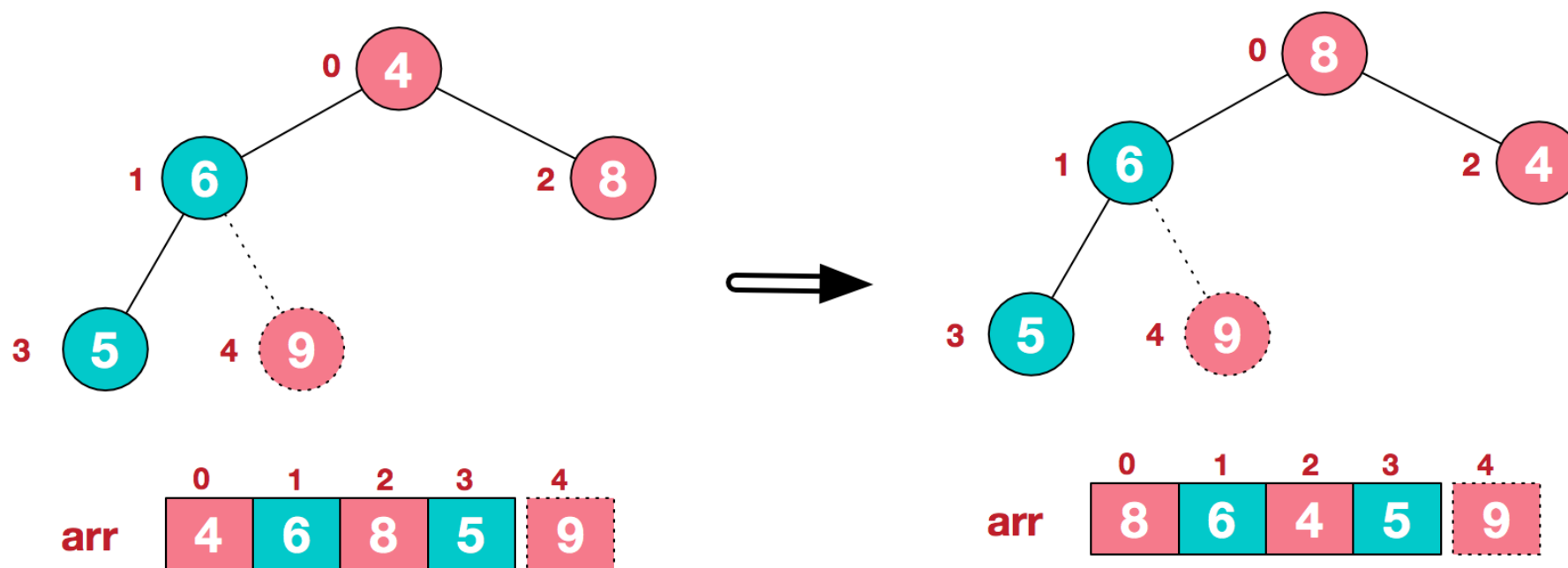




3.2 堆调整

步骤二 将堆顶元素与末尾元素进行交换，使末尾元素最大。然后继续调整堆，再将堆顶元素与末尾元素交换，得到第二大元素。如此反复进行交换、重建、交换。

Step2: 重新调整结构，使其继续满足堆定义

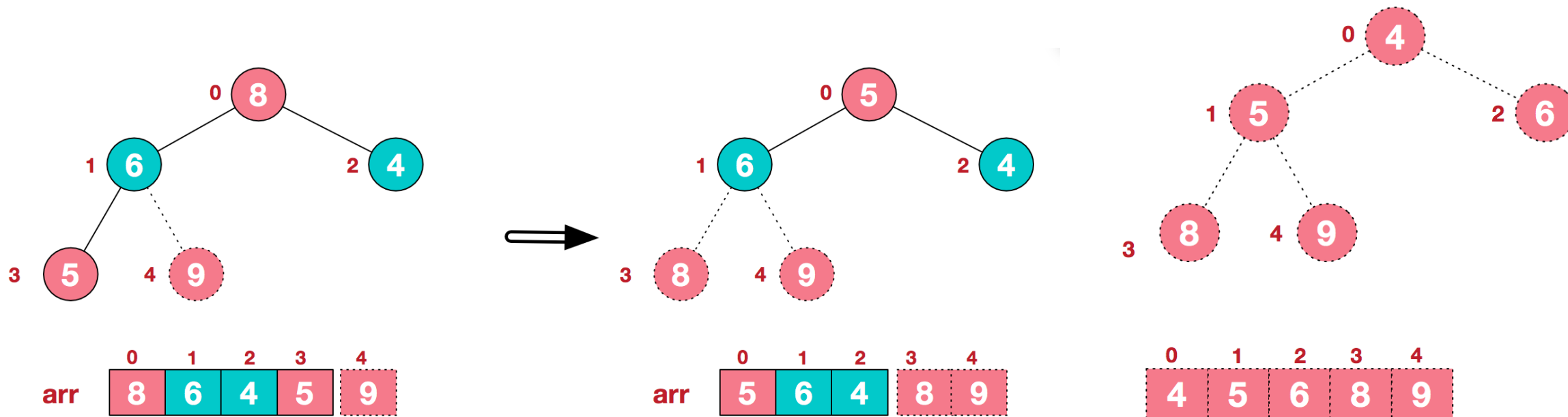




3.2 堆调整

步骤二 将堆顶元素与末尾元素进行交换，使末尾元素最大。然后继续调整堆，再将堆顶元素与末尾元素交换，得到第二大元素。如此反复进行交换、重建、交换。

Step3: 再将堆顶元素8与末尾元素5进行交换，得到第二大元素8。



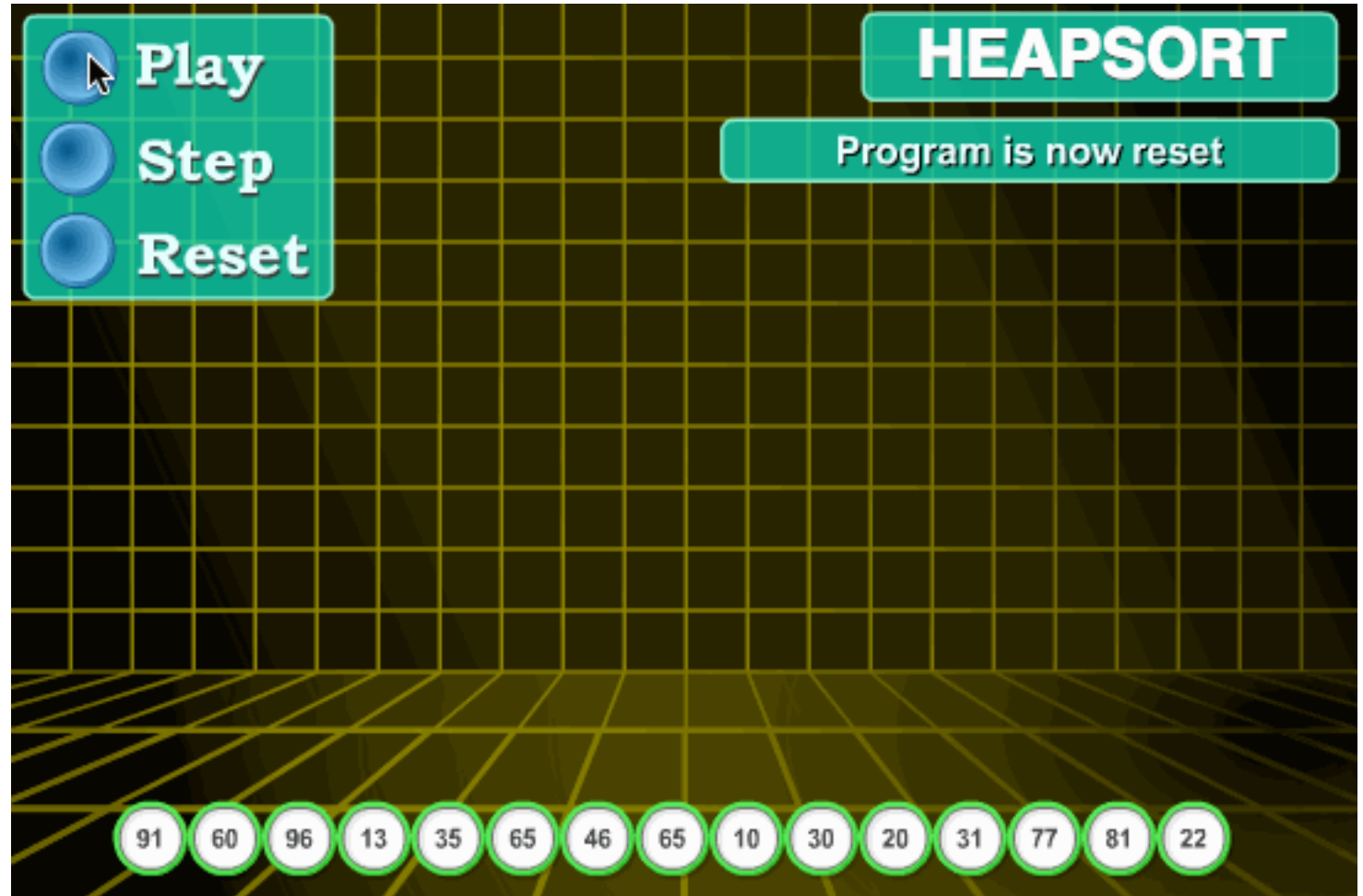
8.4 选择排序

8-4-2 堆排序

3. 堆排序的基本思想

基本思想:

- ✓ 将无序序列**建成**一个堆
- ✓ 输出**堆顶**的最小（大）值
- ✓ 使剩余的 $n-1$ 个元素又**调整**成一个堆，则可得到 n 个元素的次小值
- ✓ **重复**执行，得到一个有序序列



<https://zhuanlan.zhihu.com/p/34644389>

8.4 选择排序

8-4-2 堆排序

4 堆调整的实现

✚ **堆调整**：在一棵完全二叉树中，根结点的左右子树均是堆，**调整根结点**使整个完全二叉树成为一个堆的过程。

🕒 如何设计函数接口？

由于**初始建堆**和**重建堆**均调用此函数，因此，设置形参 k 和 $last$

```
void Sort :: Sift(int k, int last) //根结点的编号为k，最后一个结点的编号为last
{
    // ...
}
```

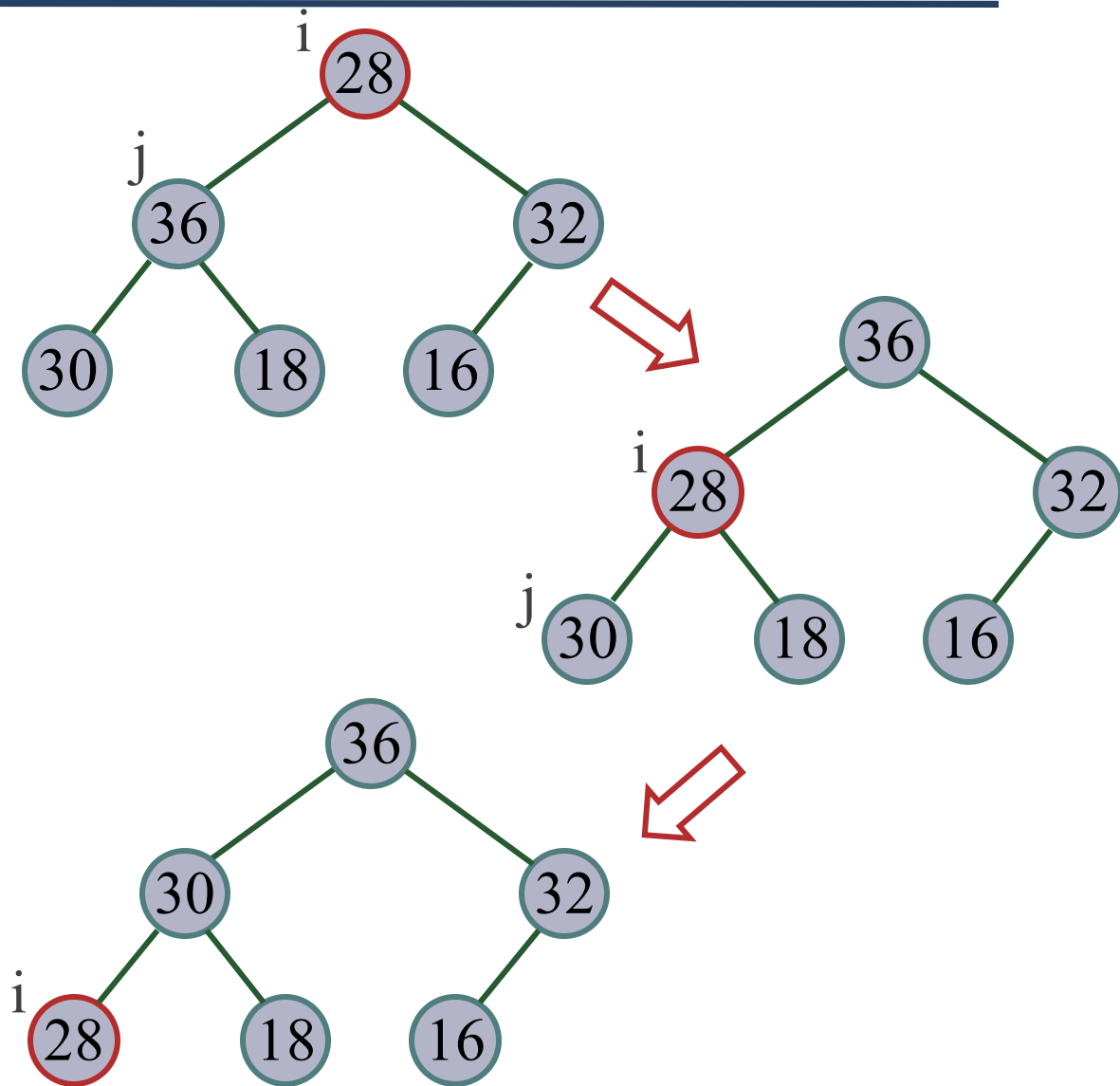
8.4 选择排序



8-4-2 堆排序

4 堆调整的实现

```
void Sort :: Sift(int k, int last)
{
    int i, j, temp;
    i = k; j = 2 * i + 1; // i是被调整结点, j是i的左孩子
    while (j <= last) //还没有进行到叶子
    {
        // j指向左右孩子的较大者
        if (j < last && data[j] < data[j+1]) j++;
        if (data[i] > data[j]) break; //已经是堆
        else {
            temp = data[i]; data[i] = data[j]; data[j] = temp;
            //被调整结点位于结点j的位置
            i = j; j = 2 * i + 1;
        }
    }
}
```

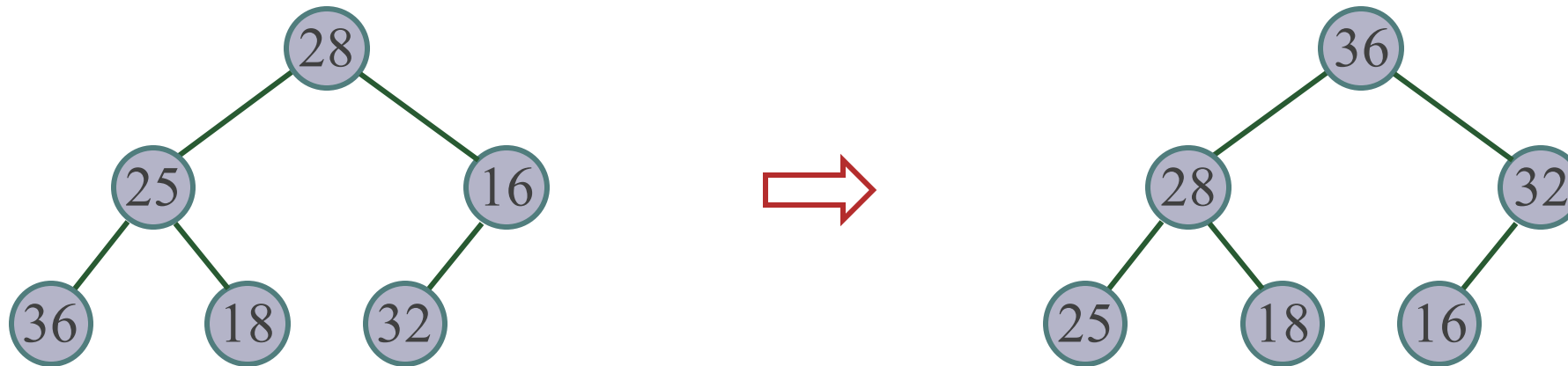


8.4 选择排序

8-4-2 堆排序

5 初始建堆的实现

待排序序列 $\{28, 25, 16, 36, 18, 32\}$ \Rightarrow 初始建堆结果 $\{36, 28, 32, 25, 18, 16\}$



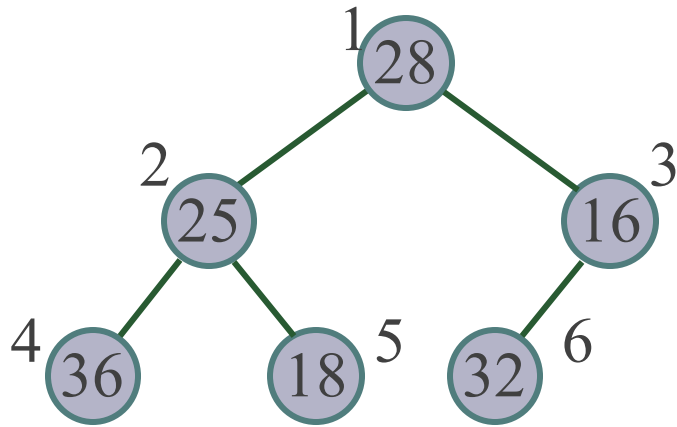
如何将一个无序序列建成一个大根堆——初始建堆？

8.4 选择排序

8-4-2 堆排序

5 初始建堆的实现

待排序序列 {28, 25, 16, 36, 18, 32}



解决办法:

从编号最大的分支结点到根结点进行调整



算法描述:

```
for (i = ceil(length/2) - 1; i >= 0; i--)  
    Sift(i, length-1);           //调整结点 i
```

```
void Sort :: Sift (int k, int last)  
    //根结点的编号为 k, 最后一个结点的编号为 last
```



需要调整叶子结点吗? 分支结点中编号最大的是多少?

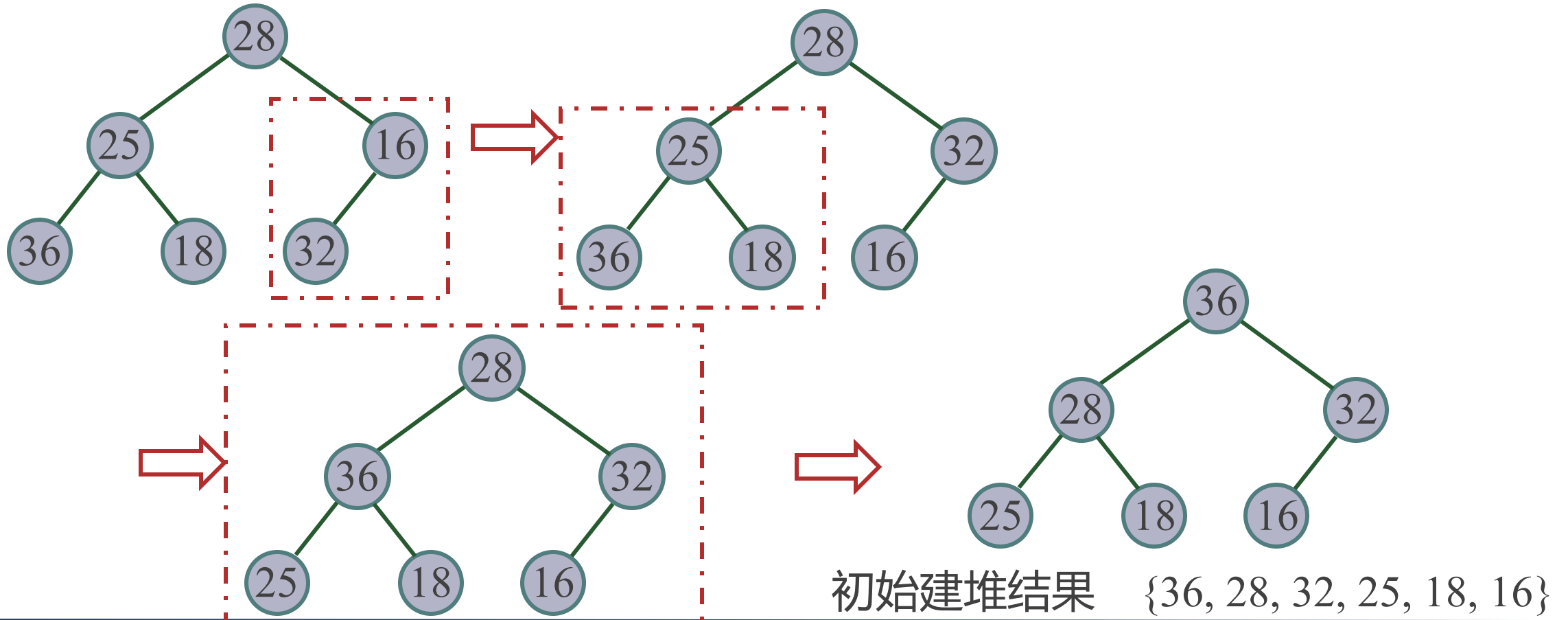
8.4 选择排序

8-4-2 堆排序

5 初始建堆的实现

待排序序列 {28, 25, 16, 36, 18, 32}

```
for (i = ceil(length/2) - 1; i >= 0; i--)  
    Sift(i, length-1);           //调整结点 i
```



8.4 选择排序

8-4-2 堆排序

6 堆排序算法实现

```
void Sort :: HeapSort( )  
{  
    int i, temp;  
    //从最后一个分支结点至根结点  
    for (i = ceil(length/2) - 1; i >= 0; i--)  
        Sift(i, length-1);  
    for (i = 1; i < length; i++)  
    {  
        temp = data[0]; data[0] = data[length-i]; data[length-i] = temp;  
        Sift(0, length-i-1); //重建堆  
    }  
}
```

 初始建堆: $O(n\log_2 n)$

 重建堆次数: $n-1$

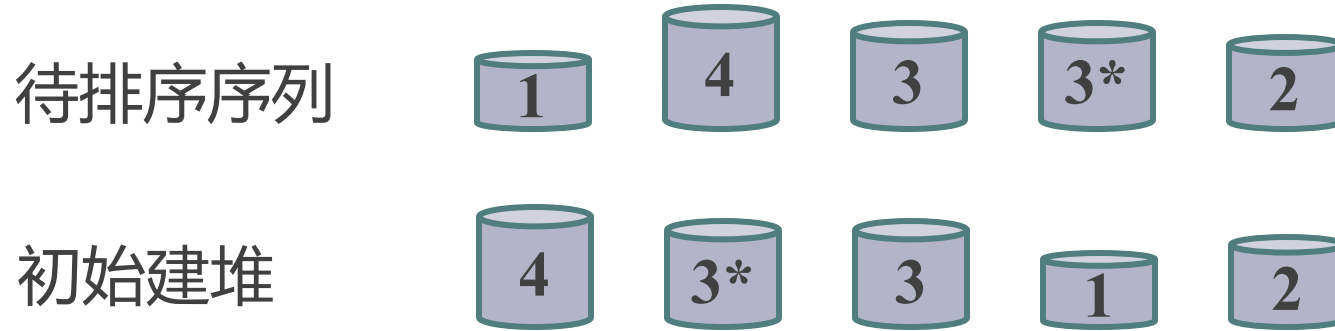
 重建堆: $O(\log_2 i)$

 最好、最坏、平均情况: $O(n\log_2 n)$

8.4 选择排序

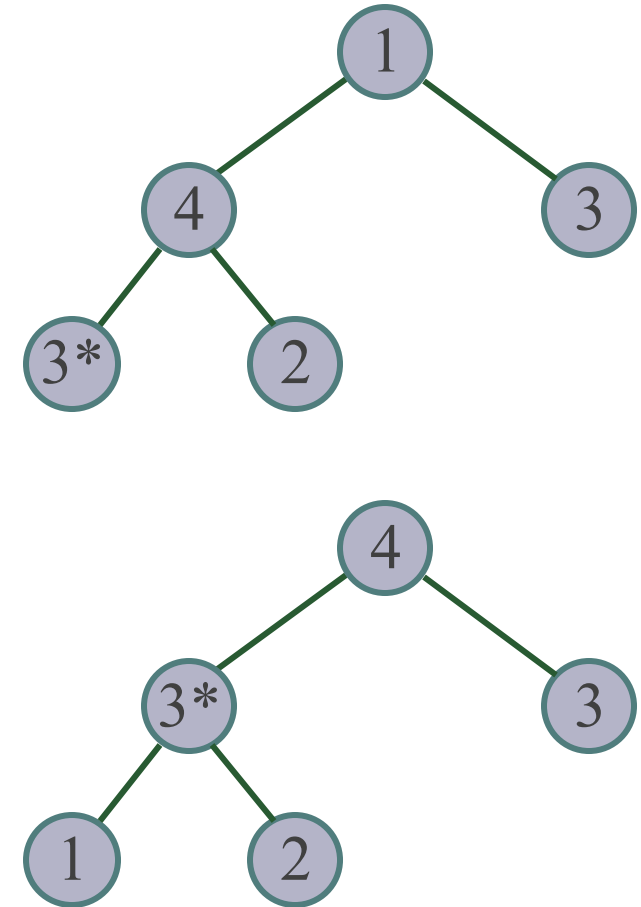
8-4-2 堆排序

7 空间性能分析



📜 空间性能: $O(1)$

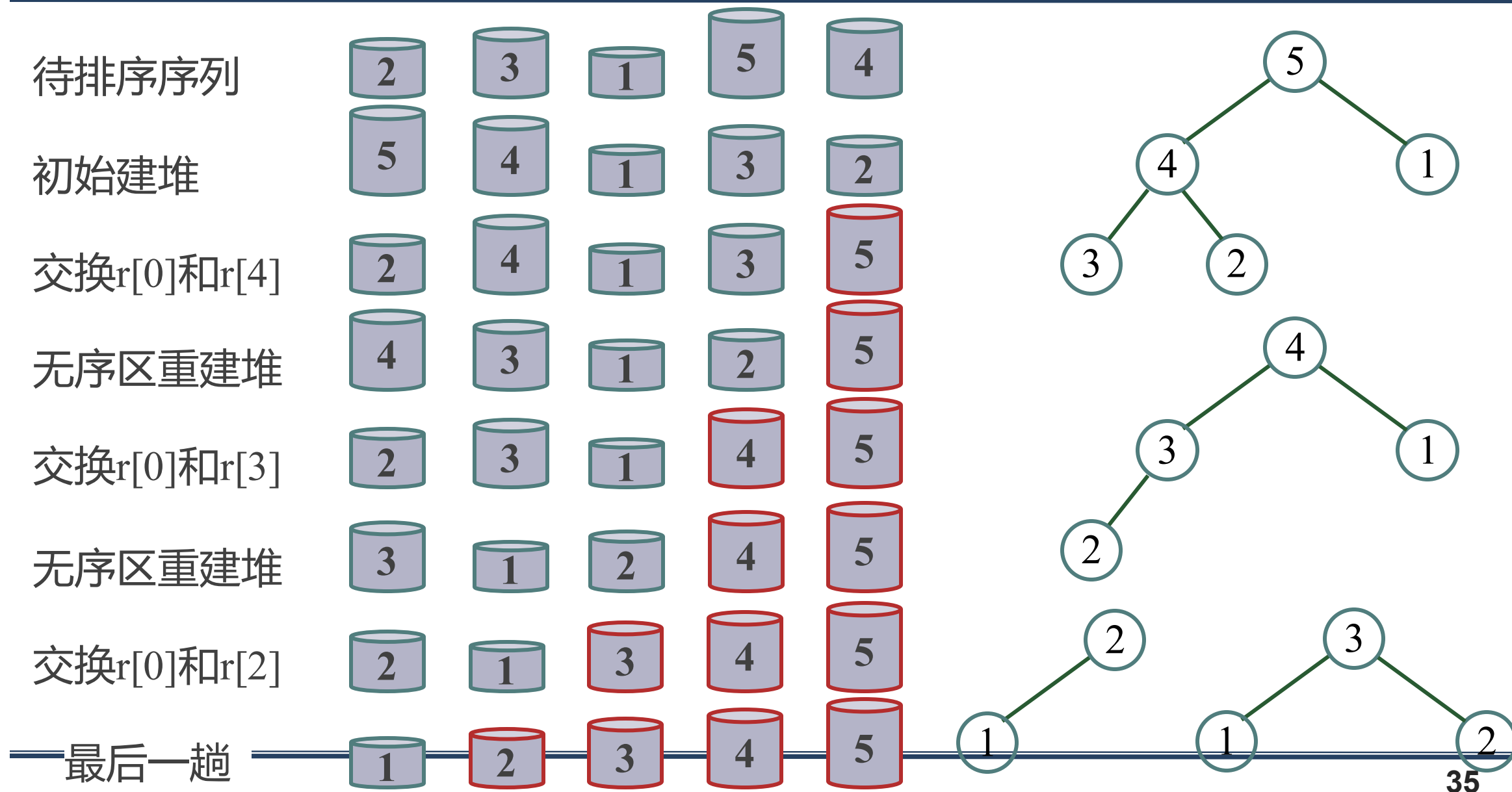
📜 稳定性: 不稳定



8.4 选择排序

运行实例2

8-4-2 堆排序

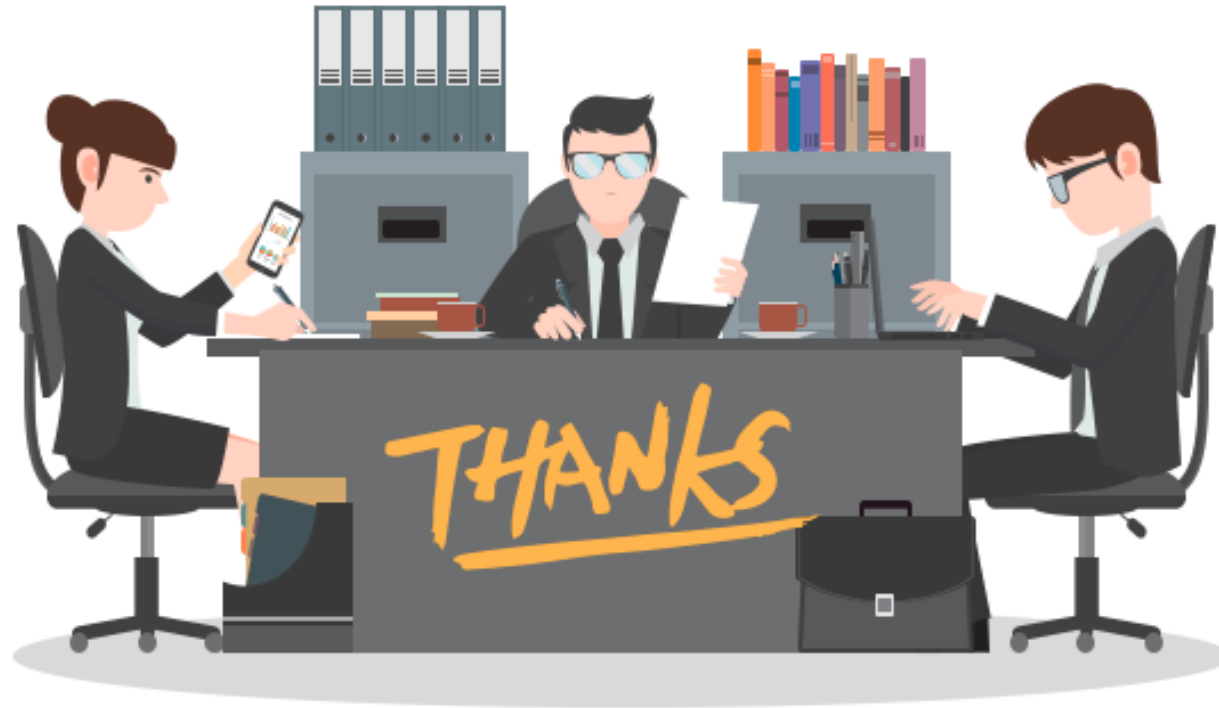


小结

1. 掌握**简单选择排序**算法及实现
2. 掌握**堆排序**算法及实现

作业

1. 已知关键字序列(3, 26, 38, 5, 47, 15, 36, 26*, 2, 4, 19, 50), 使用直接插入排序、希尔排序、起泡排序、快速排序、简单选择排序、堆排序、二路归并排序七种排序算法进行排序, 请分别给出七种排序算法每一趟排序的结果, 并给出时间和空间复杂度量级。



Thank You !

Q & A