# [九种排序算法分析与实现](https://www.cnblogs.com/Glory-D/p/7884525.html)

　　简介：总的来说，排序算法共有八大类，即冒泡排序、选择排序、快速排序、插入排序、希尔排序、归并排序、基数排序以及堆排序等，本文另外也介绍了桶排序。编程语言使用了C/C++（其实主要用的C），3个经常出现的函数形参，arr - 待排序数组名（首元素地址）、bgn - 待排序数组起始排序元素位置（有时我们仅需要对数组中某一段元素进行排序，但通常bgn = 0，即arr首元素位置）、end - 待排序数组截止排序尾元素的下一个位置（即该位置无效，不可引用）。文中均已升序为例，降序原理相同。

　　时间复杂度：描述该算法在处理大量数据时，总的来说其时间效率的参考；　　稳定性：描述算法对原始序列处理前后，该序列相等大小的元素前后位置是否发生改变

　　两个常用的函数：1、获取数组最大元素值； 2、交换两个整形元素。代码如下：

 1、**冒泡排序** - 依次比较相邻两元素，若前一元素大于后一元素则交换之，直至最后一个元素即为最大；然后重新从首元素开始重复同样的操作，直至倒数第二个元素即为次大元素；依次类推。如同水中的气泡，依次将最大或最小元素气泡浮出水面。

时间复杂度：**O(N2)**　　稳定性：**稳定**

2、**选择排序** - 首先初始化最小元素索引值为首元素，依次遍历待排序数列，若遇到小于该最小索引位置处的元素则刷新最小索引为该较小元素的位置，直至遇到尾元素，结束一次遍历，并将最小索引处元素与首元素交换；然后，初始化最小索引值为第二个待排序数列元素位置，同样的操作，可得到数列第二个元素即为次小元素；以此类推。

时间复杂度：**O(N2)** 　　稳定性：**不稳定**

3、**快速排序** - （类似于选择排序的定位思想）选一基准元素，依次将剩余元素中小于该基准元素的值放置其左侧，大于等于该基准元素的值放置其右侧；然后，取基准元素的前半部分和后半部分分别进行同样的处理；以此类推，直至各子序列剩余一个元素时，即排序完成（类比二叉树的思想，from up to down）

时间复杂度：**O(NlogN)** 　　稳定性：**不稳定**

4、**插入排序** - 数列前面部分看为有序，依次将后面的无序数列元素插入到前面的有序数列中，初始状态有序数列仅有一个元素，即首元素。在将无序数列元素插入有序数列的过程中，采用了逆序遍历有序数列，相较于顺序遍历会稍显繁琐，但当数列本身已近排序状态效率会更高。

时间复杂度：**O(N2)** 　　稳定性：**稳定**

5、**希尔排序** - 插入排序的改进版。为了减少数据的移动次数，在初始序列较大时取较大的步长，通常取序列长度的一半，此时只有两个元素比较，交换一次；之后步长依次减半直至步长为1，即为插入排序，由于此时序列已接近有序，故插入元素时数据移动的次数会相对较少，效率得到了提高。

时间复杂度：**通常认为是O(N3/2) ，未验证**　　稳定性：**不稳定**

6、**桶排序** - 实现线性排序，但当元素间值得大小有较大差距时会带来内存空间的较大浪费。首先，找出待排序列中得最大元素max，申请内存大小为max + 1的桶（数组）并初始化为0；然后，遍历排序数列，并依次将每个元素作为下标的桶元素值自增1；最后，遍历桶元素，并依次将值非0的元素下标值载入排序数列（桶元素>1表明有值大小相等的元素，此时依次将他们载入排序数列），遍历完成，排序数列便为有序数列。

时间复杂度：**O(x\*N)** 　　稳定性：**稳定**

7、**基数排序** - 桶排序的改进版，桶的大小固定为10，减少了内存空间的开销。首先，找出待排序列中得最大元素max，并依次按max的低位到高位对所有元素排序；桶元素10个元素的大小即为待排序数列元素对应数值为相等元素的个数，即每次遍历待排序数列，桶将其按对应数值位大小分为了10个层级，桶内元素值得和为待排序数列元素个数。

时间复杂度：**O(x\*N)** 　　稳定性：**稳定**

8、**归并排序** - 采用了分治和递归的思想，递归&分治-排序整个数列如同排序两个有序数列，依次执行这个过程直至排序末端的两个元素，再依次向上层输送排序好的两个子列进行排序直至整个数列有序（类比二叉树的思想，from down to up）。

时间复杂度：**O(NlogN)** 　　稳定性：**稳定**

9、**堆排序** - 堆排序的思想借助于二叉堆中的最大堆得以实现。首先，将待排序数列抽象为二叉树，并构造出最大堆；然后，依次将最大元素（即根节点元素）与待排序数列的最后一个元素交换（即二叉树最深层最右边的叶子结点元素）；每次遍历，刷新最后一个元素的位置（自减1），直至其与首元素相交，即完成排序。

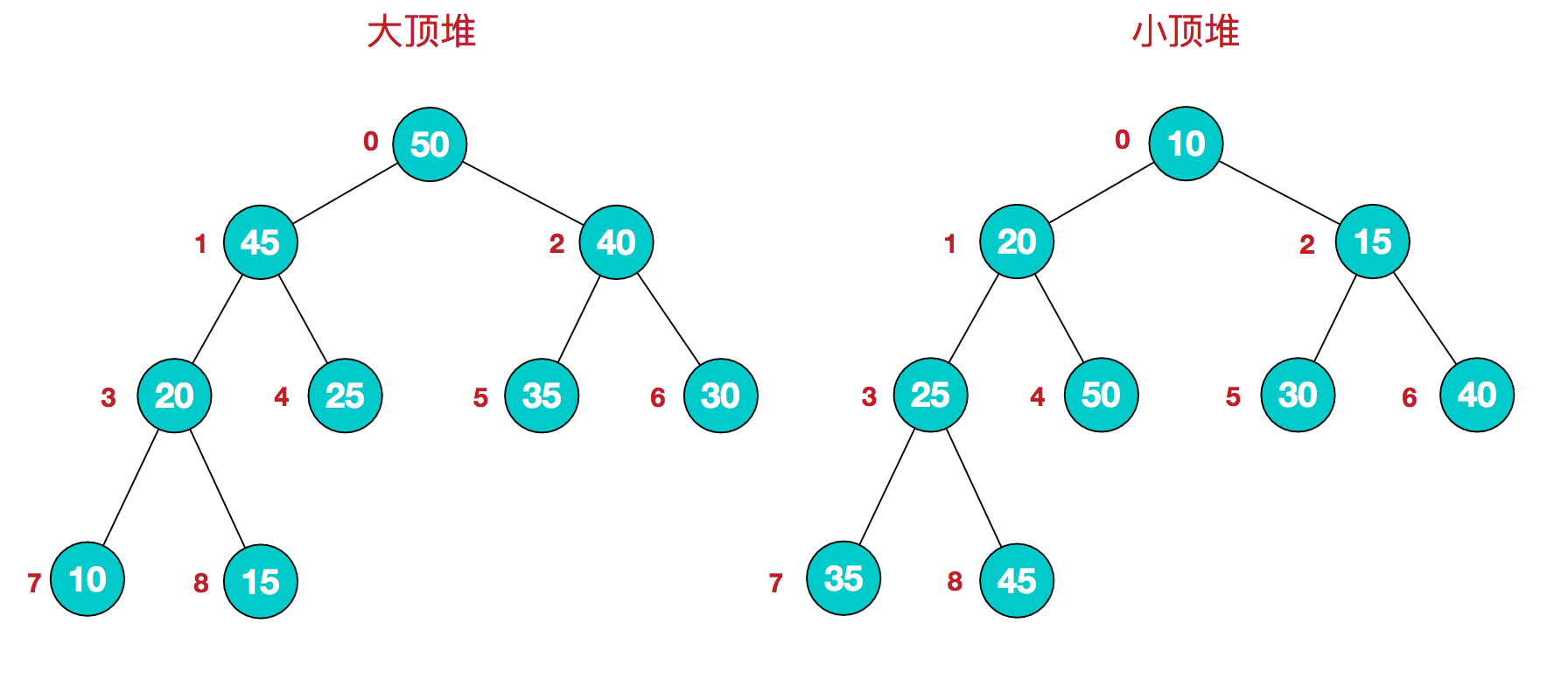
时间复杂度：**O(NlogN)** 　　稳定性：**不稳定**

**堆排序**

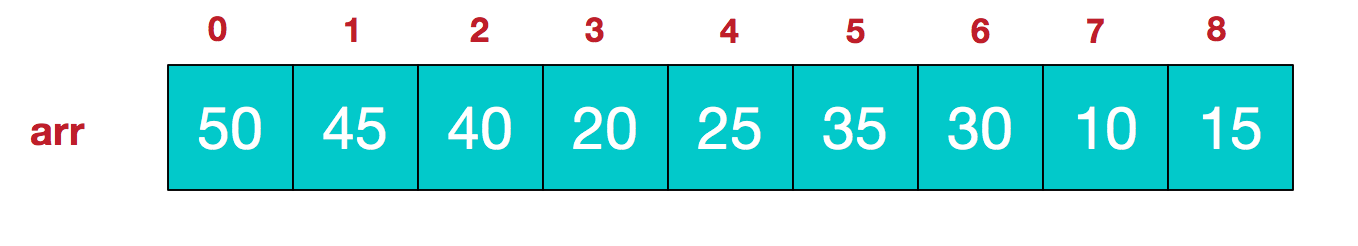
　　堆排序是利用**堆**这种数据结构而设计的一种排序算法，堆排序是一种**选择排序，**它的最坏，最好，平均时间复杂度均为O(nlogn)，它也是不稳定排序。首先简单了解下堆结构。

**堆**

**堆是具有以下性质的完全二叉树：每个结点的值都大于或等于其左右孩子结点的值，称为大顶堆；或者每个结点的值都小于或等于其左右孩子结点的值，称为小顶堆。如下图：**



同时，我们对堆中的结点按层进行编号，将这种逻辑结构映射到数组中就是下面这个样子



该数组从逻辑上讲就是一个堆结构，我们用简单的公式来描述一下堆的定义就是：

**大顶堆：arr[i] >= arr[2i+1] && arr[i] >= arr[2i+2]**

**小顶堆：arr[i] <= arr[2i+1] && arr[i] <= arr[2i+2]**

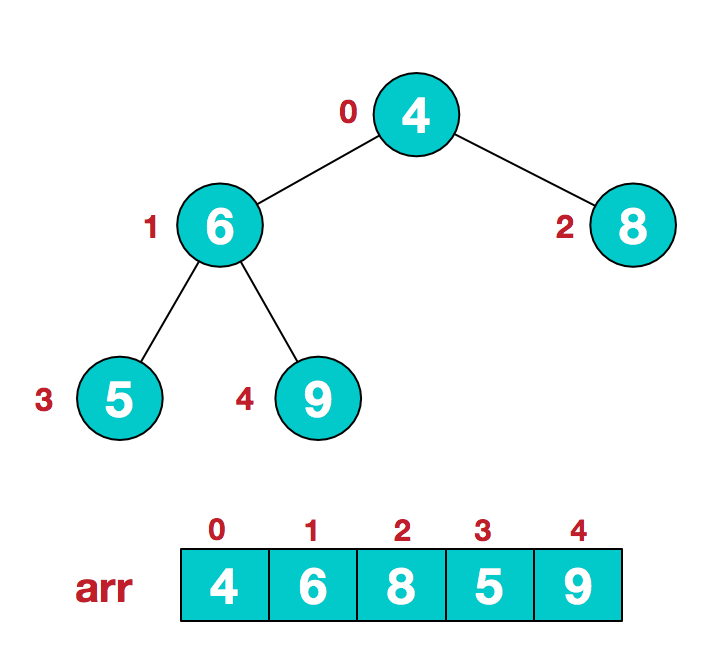
ok，了解了这些定义。接下来，我们来看看堆排序的基本思想及基本步骤：

# 堆排序基本思想及步骤

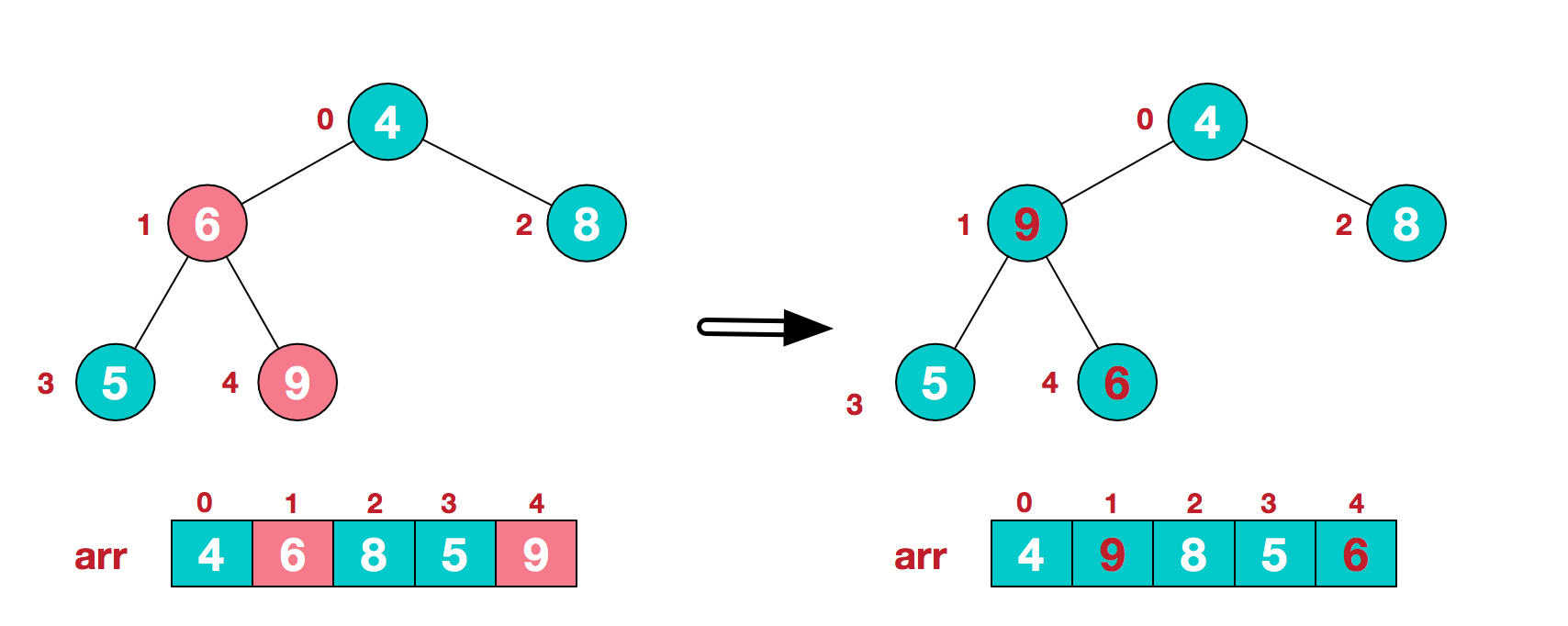
**堆排序的基本思想是：将待排序序列构造成一个大顶堆，此时，整个序列的最大值就是堆顶的根节点。将其与末尾元素进行交换，此时末尾就为最大值。然后将剩余n-1个元素重新构造成一个堆，这样会得到n个元素的次小值。如此反复执行，便能得到一个有序序列了**

**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)。**

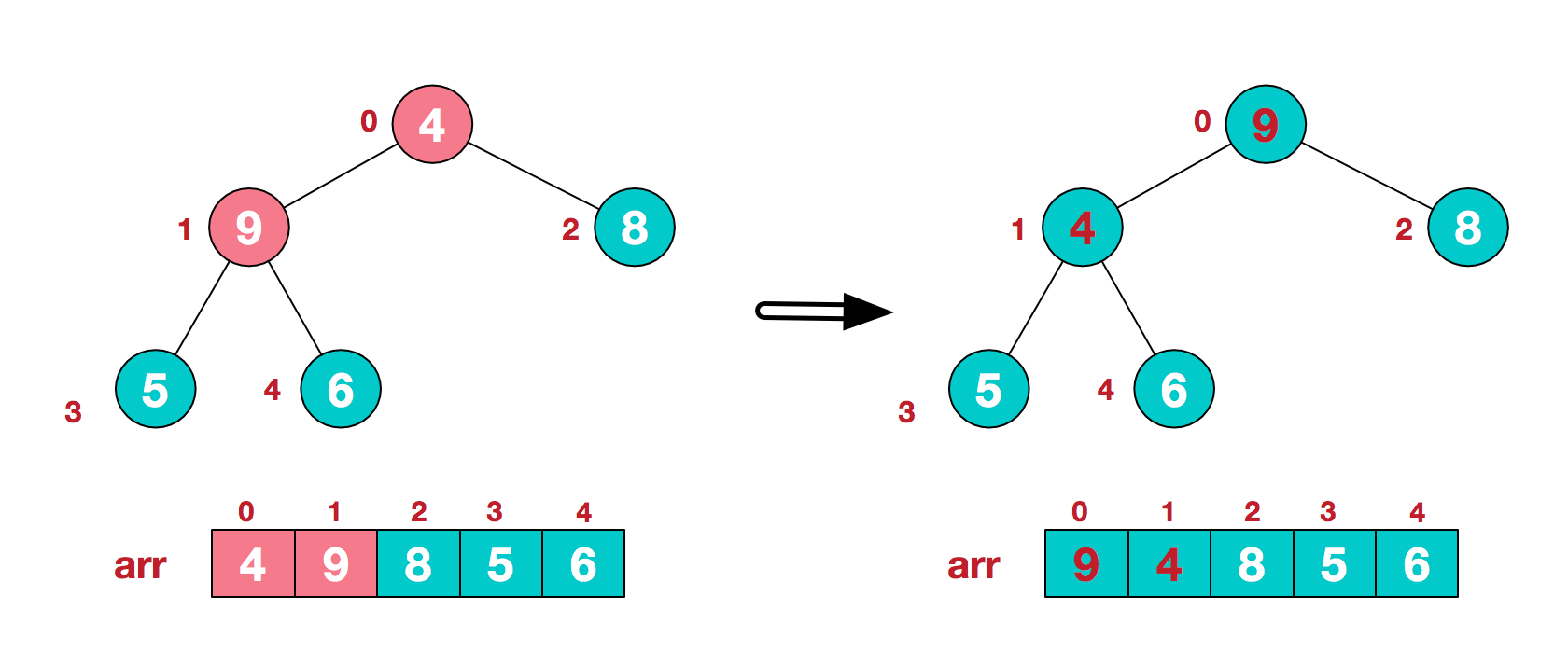
　　a.假设给定无序序列结构如下



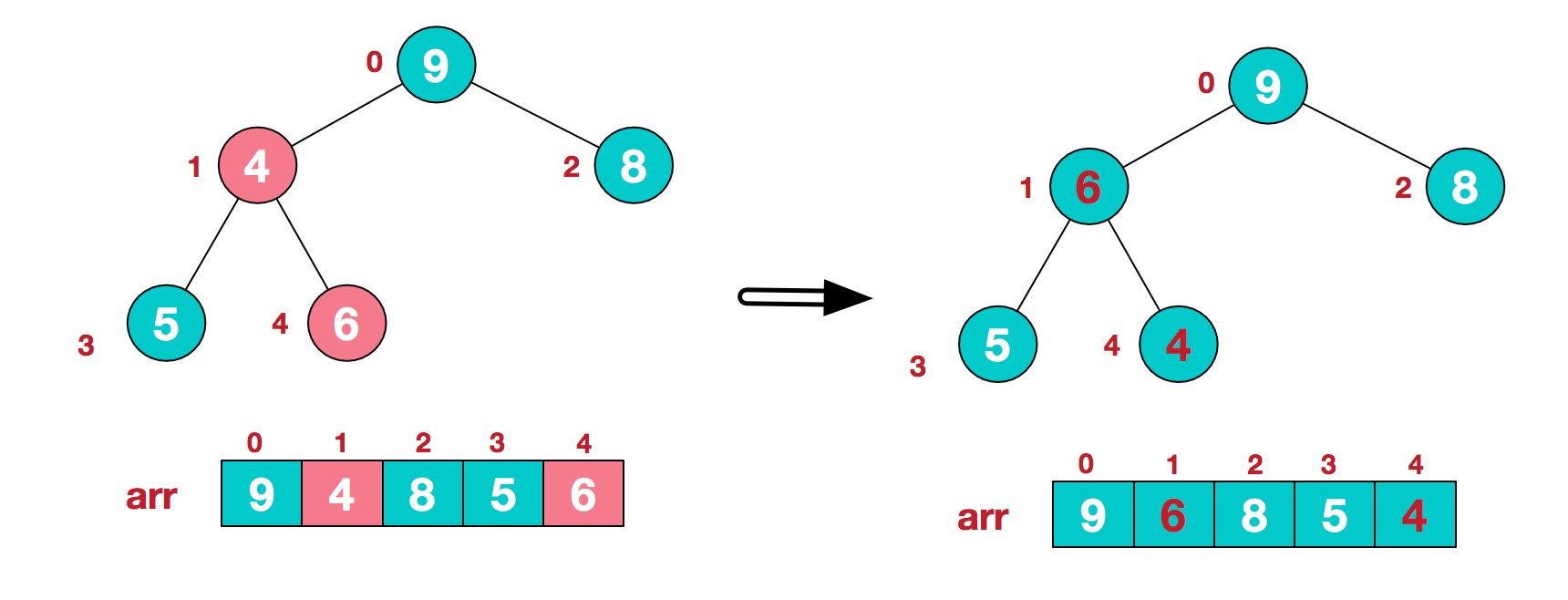
2.此时我们从最后一个非叶子结点开始（叶结点自然不用调整，第一个非叶子结点 arr.length/2-1=5/2-1=1，也就是下面的6结点），从左至右，从下至上进行调整。



4.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。



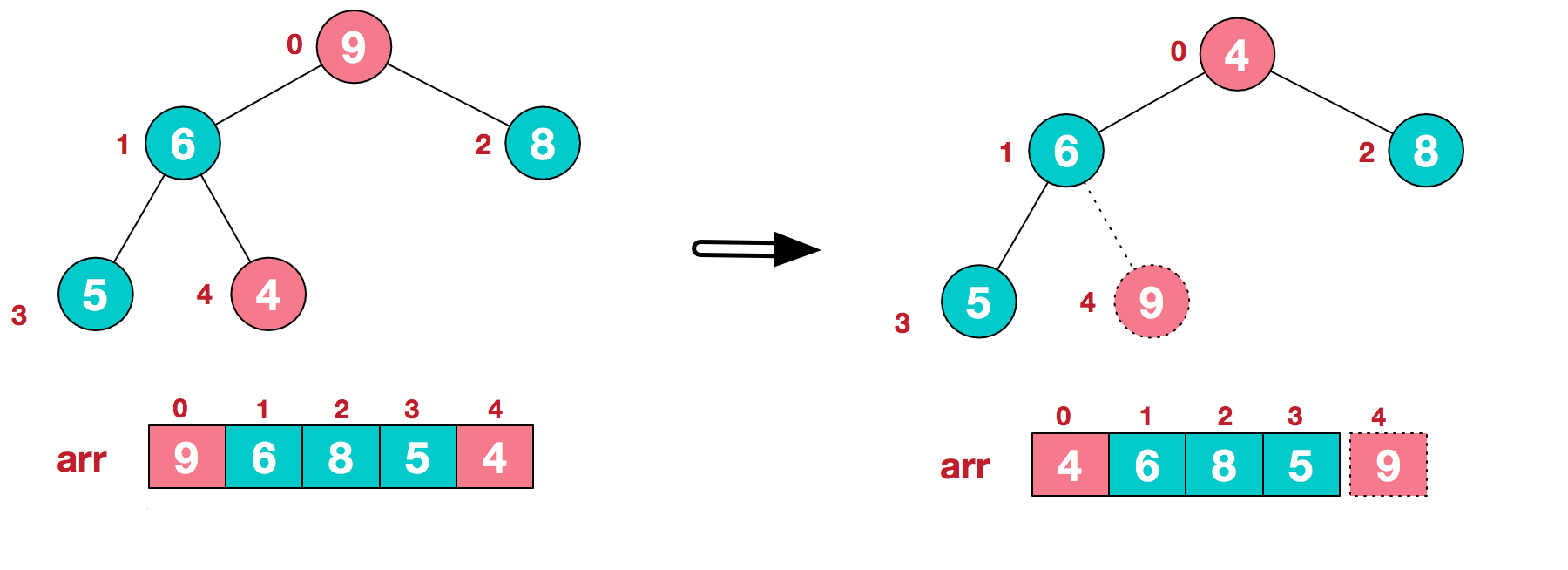
这时，交换导致了子根[4,5,6]结构混乱，继续调整，[4,5,6]中6最大，交换4和6。



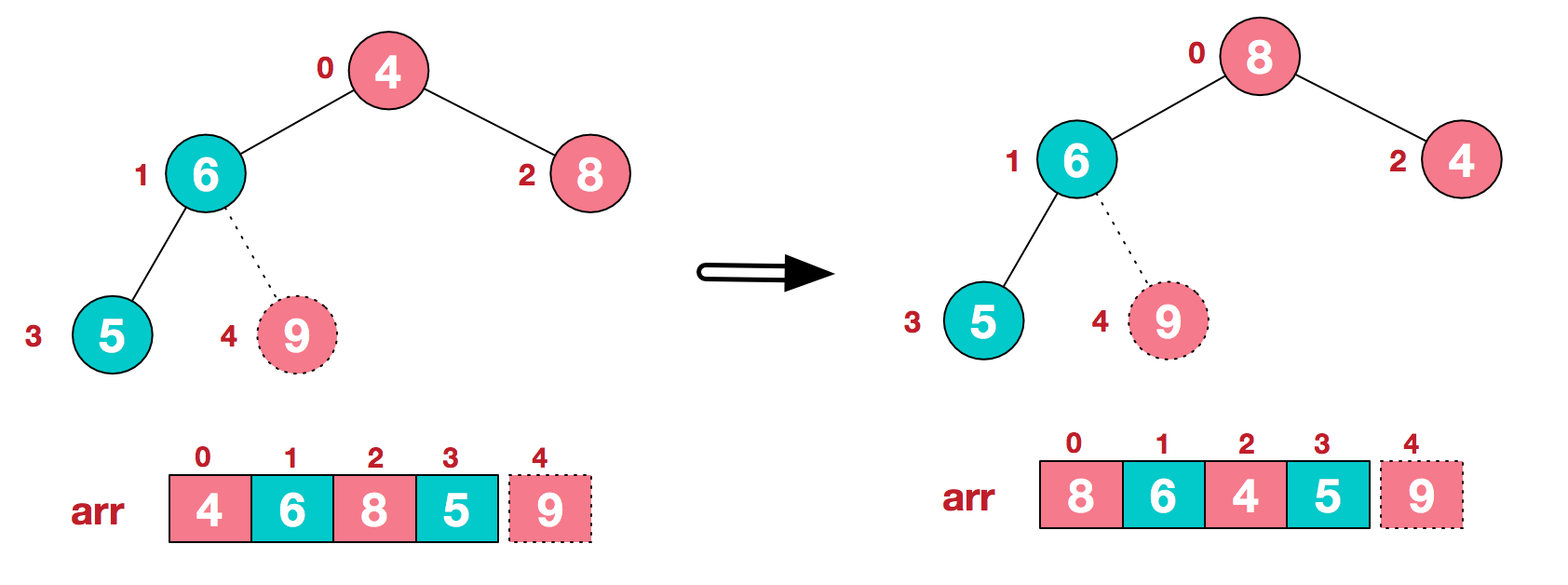
此时，我们就将一个无需序列构造成了一个大顶堆。

**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。**

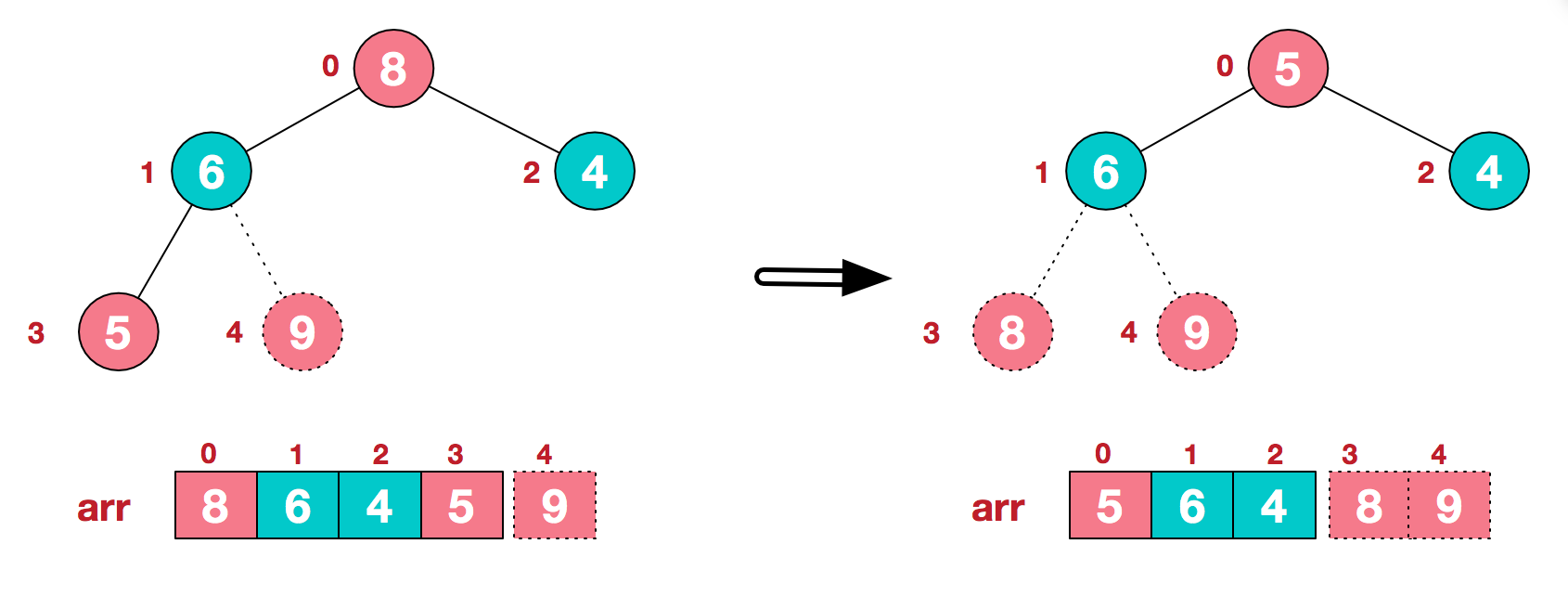
a.将堆顶元素9和末尾元素4进行交换



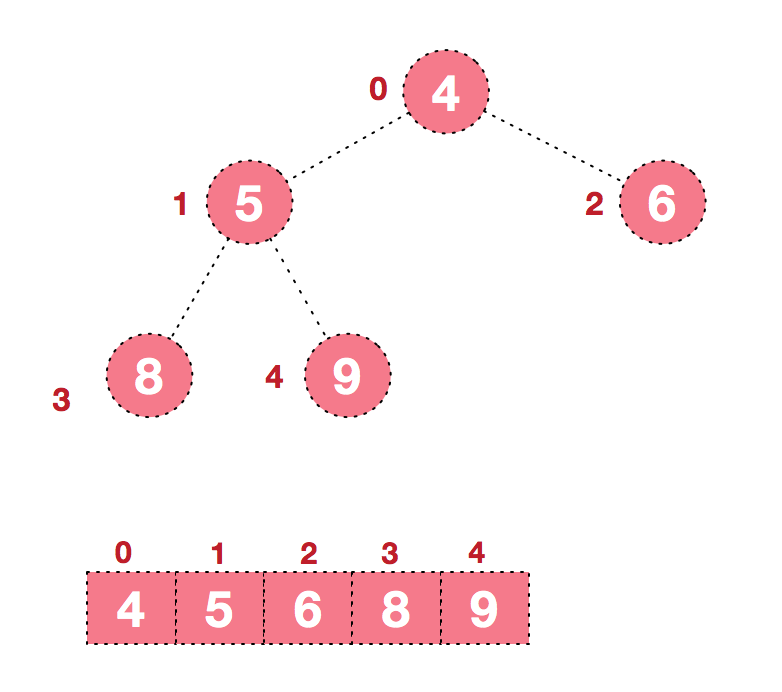
b.重新调整结构，使其继续满足堆定义



c.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.



后续过程，继续进行调整，交换，如此反复进行，最终使得整个序列有序



再简单总结下堆排序的基本思路：

**a.将无需序列构建成一个堆，根据升序降序需求选择大顶堆或小顶堆;**

**b.将堆顶元素与末尾元素交换，将最大元素"沉"到数组末端;**

**c.重新调整结构，使其满足堆定义，然后继续交换堆顶元素与当前末尾元素，反复执行调整+交换步骤，直到整个序列有序。**