**堆排序讲解**

1. 基本概念讲解

首先，讲解堆排序之前，我们需要先了解一下，一些关于树的基本概念：

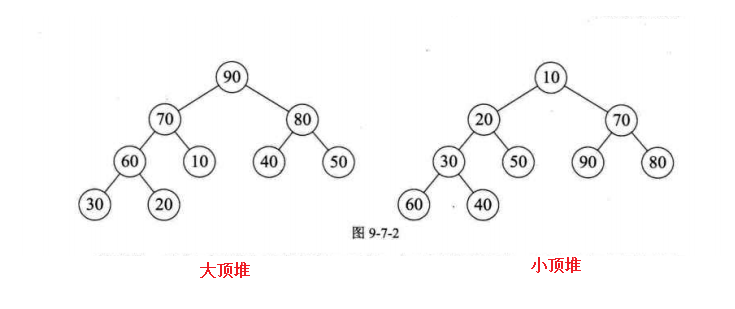
**二叉堆：**二叉堆是一种特殊的堆，二叉堆是完全二元树（二叉树）或者是近似完全二元树（二叉树）。它分为两种一种叫做**大顶堆**，还有一种叫做**小顶堆**。

**大顶堆：每个节点的值都大于或则等于其左右孩子的值。**

**小顶堆：每个节点的值都小于或则等于其左右孩子的值。**

**我们的堆排序中一定需要先把树构建成大顶堆或则小顶堆，然后在进行排序。如下图，**

**就是我们常见的大顶堆和小顶堆。**



**堆排序的思路：**

**堆排序就是利用堆(这里假设利用大顶堆)进行排的方法。一般是将我们的待排序的序列构成一个大顶堆。此时，整个序列的最大值就是堆顶的根结点。然后将它移走(就是将其与堆数组的末尾元素进行交换，此时末尾元素就是最大的值。)，然后将剩余的n-1个序列重新构建成一个堆，这样就会得到n个元素的次小值。依次反复进行，就可以得到一个有序序列。**

**操作步骤：**

**[1]** **初始化堆：将R[1..n]构造为大顶堆；**

**[2]将当前无序区的堆顶元素R[1]同该区间的最后一个记录交换，然后将新的无序区调整为新的堆。**

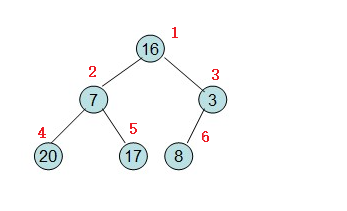
**因此对于堆排序，最重要的两个操作就是构造初始堆和调整堆，其实构造初始堆事实上也是调整堆的过程，只不过构造初始堆是对所有的非叶节点都进行调整。**

**二、举例说明**

**假如，我们要把数组中的一些数据进行排序。**

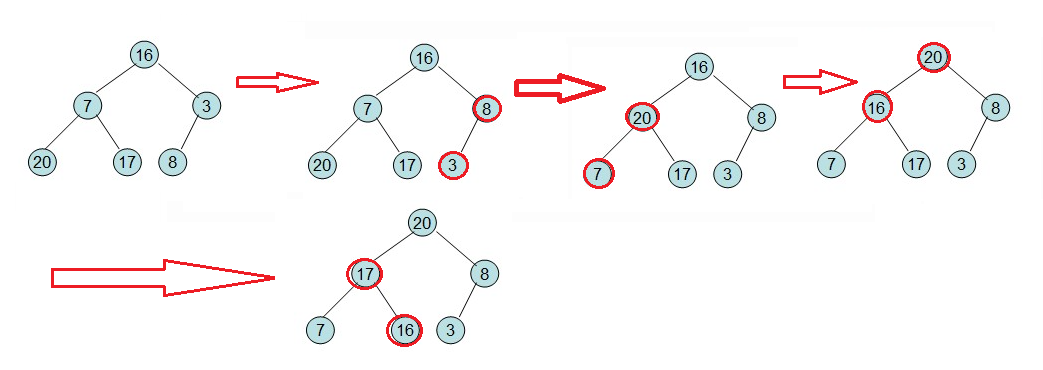
**int a[] = {16,7,3,20,17,8}.进行堆排序。**

1. **首先，把上述的数据构成一个棵完全二叉树。如下图。层次遍历**

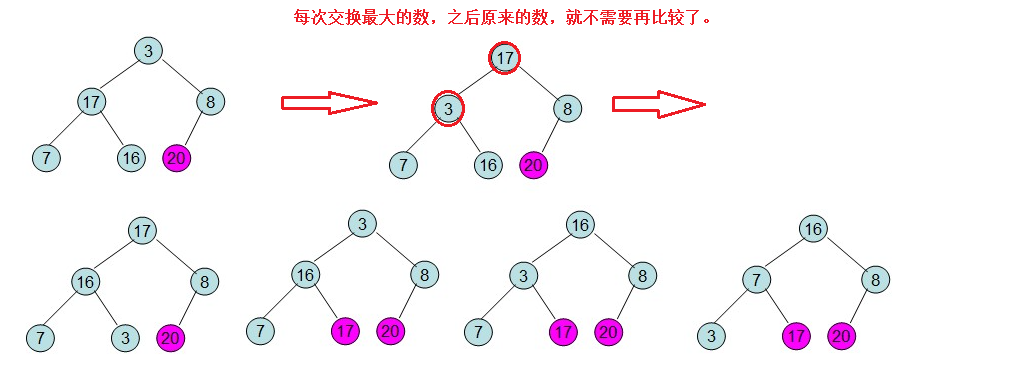


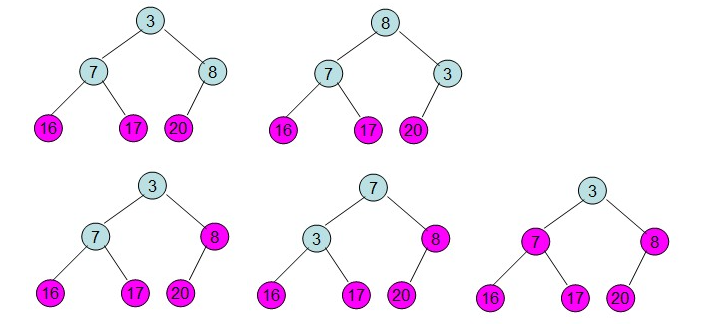
1. **然后把二叉树，构造成大顶堆。从最后一个非叶子结点开始调整，调整成我们的大顶堆，步骤如下：（由于排序之后，我们的根结点一定是最大的结点，而知道非叶子结点，可以知道我们的子结点，故我们知道1,2,3这几个结点，就可以访问其子结点，而我们要构建成大顶堆，故从下到上，从右到左进行访问。非叶子结点的个数。**

**所有结点/2，可得。**



1. **这样就得到了我们的初始的大顶堆，然后对初始堆进行排序。每次调整都是拿根结点和我们的最后一个元素进行交换。交换之后可能就不满足我们大顶堆的性质了。故每次交换完成之后，我们需要重新对被交换的孩子结点进行调整。有初始堆就可以排序了。**





**由上述过程可知，堆排序其实也是一种选择排序。而且是一种树形选择排序。**

**由上述过程可知，若是想要升序，则建立大顶堆，若是想要降序，则建立小顶堆。**

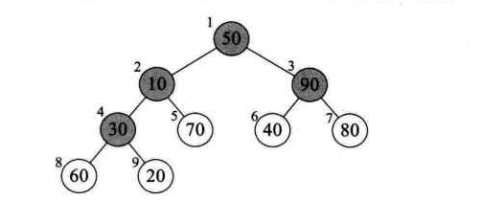
**代码简介：**

**1.假设排列**

**Int a[ ] = {-1,50,10,30,70,40,80,60,20};//注为了让结点编号和下标对应，故下标为0的位置**

**//不进行排序。排序个数为9个。**

**构建成完全二叉树如下。 （我们只需要堆1,2,3,4这个几个下标的子结点进行比较即可。）**



**int a[ ] = {-1,50,10,90,30,70,40,80,60,20};**

**int len = sizeof(a) / sizeof(a[0]) – 1;**

**//把a[i]这个结点和其两个子节点比较，把最大的值赋值给父节点**

**void build\_heap(int a[ ],int i,int len)**

**{**

**temp = a[i];**

**int k = 0;**

**for(k = 2 \* i; k <= len;k = k \* 2)**

**{**

**//找到i的子节点k 和k + 1中较大值的结点。**

**//k记录了两个子结点中较大的值得下标。K < len说明是左结点**

**if(k < len && a[k] < a[k + 1])**

**k++;**

**//拿我们的a[i]结点和其较大的子结点a[k]比较，若是temp大**

**//说明它在三个结点中是最大的，不需要交换，直接退出。**

**if（temp >=a[k]）**

**break;**

**//否则说明不是最大的，需要交换**

**a[i] = a[k]; //把较大的值赋给a[i]**

**i = k; //记录其结点，然后方面交换**

**}**

**a[i] = temp; //把temp的值给其子结点。**

**}**

**Void heap\_sort()**

**{**

**//只访问4,3,2,1这几个结点，然后让其子结点和该结点进行比较**

**for(i = len / 2; i >= 1;i--)**

**{**

**bulid\_heap(a,i,len); //把当前的数据构造成大顶堆**

**}**

**//下面我们来进行我们的堆排序。**

**for(i = len;i > 1;i--)**

**{**

**a[1] ^= a[len]; //将堆顶的元素和数组最后一个元素进行交换。**

**a[len] ^= a[1];**

**a[1] ^= a[len];**

**Build\_heap(a,1,i-1);//将剩余的元素重新排列为大顶堆。**

**}**

**}**

**它的运行时间主要消耗在初始建堆和重复建堆的时候。我们是从完全二叉树的最下层和最右边的非终端节点开始构建的。对于每个非终端节点来说，其实最多进行两次比较和交换的**

**操作，因此构建堆的时间复杂度为O（n）.而第I次取堆顶记录重建堆需要O（logi）的时间。重建堆的时间复杂度O(nlogn).**

**它在效率上要好过冒泡、简单选择排序，和直接插入排序。**