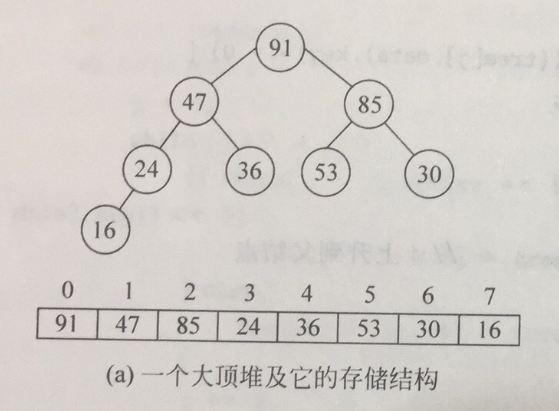
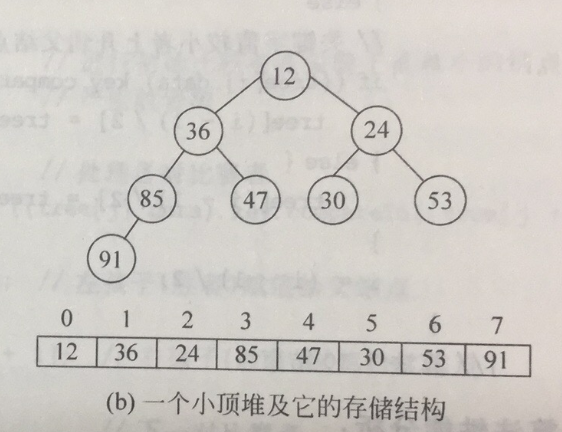
堆排序是一种重要的选择排序方法，它只需要一个记录大小的辅助存储空间，在每个待排序的记录仅占用一个记录大小的存储空间，因此弥补了树形选择排序的弱点。





以小顶堆为例，由堆的特点可知，虽然序列中的记录无序，但是在小顶堆中，堆顶记录的关键字值是最小的。

**基本思想**

先将这n条记录按关键字值的大小建成堆（称为初始堆），将堆顶元素nums[0]与

nums[n-1]交换（或输出），然后将剩下的nums[0]…nums[n-2]序列调整成为堆，再将nums[0]与nums[n-2]交换，再把剩下的nums[0]…nums[n-3]序列调整成为堆，如此反复，便可以得到一个按关键字值有序的记录序列。

面临问题

1.如何将n条记录的序列按关键字值的大小建成初始堆

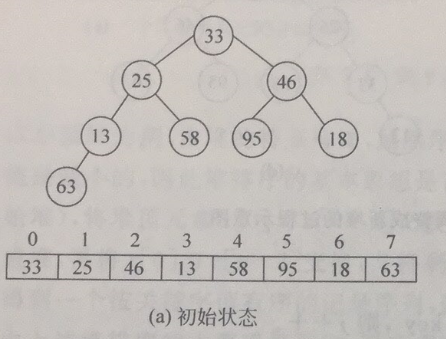
2.将堆顶记录nums[0]与nums[i]交换后，如何将堆调整成一个新的满足条件的堆

**初始建堆**

为一个无序序列建堆的过程就是对完全二叉树从下往上反复筛选的过程。

完全二叉树的最有一个非叶子结点的编号为n/2（向下取整）-1，所以筛选只需要从编号为n/2（向下取整）-1的结点开始，

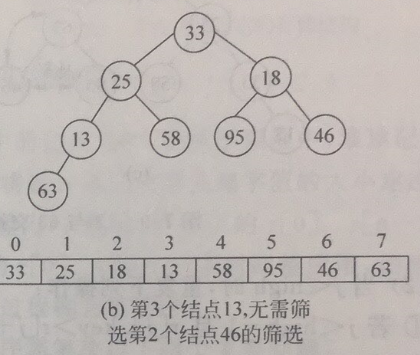
示例



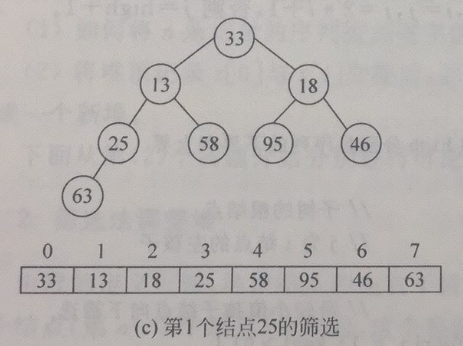
n=8,n/2-1=3,所以我们从第三个开始。

首先考察以13为根结点的子树，13<63,满足堆定义，无需筛选。考察第二个。

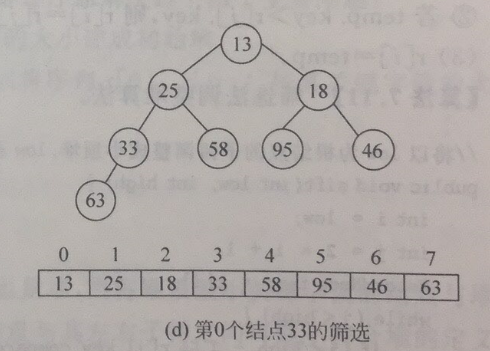
考察以46为根结点的子树，46>18，需要堆结点46进行筛选，就是把46和18互换位置。筛选后的状态



然后考察序号为1的结点25.25>13,不符合堆的定义，筛选。25和13互换位置



最后考察0，13>18,33>13，不符合堆的定义，筛选。13和33互换位置。

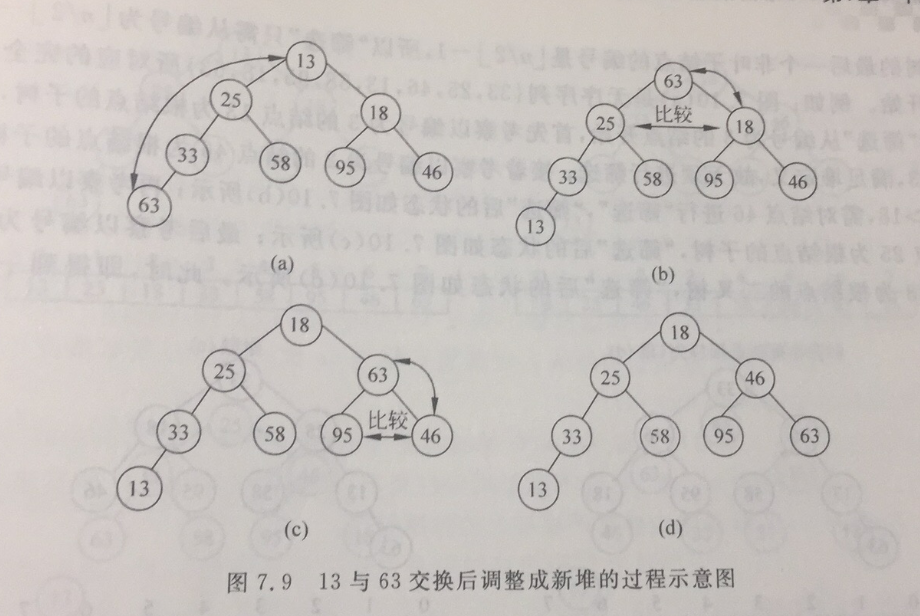


**筛选法调整堆**

由堆的定义可知，堆顶结点的左右子树也是堆，当将堆顶最小关键字值的结点与堆最后一个结点（第n-1个结点）交换后，这个根结点与其左右子树之间可能不符合堆的定义。因此，此时需要调整根结点与其左右两个堆顶结点之间的大小关系，使之符合堆的定义。

**调整方法**

将根节点nums[0]与左右孩子中关键字较小的结点进行交换，若与左孩子交换，则左子树堆被破坏，且仅左子树的根结点不满足堆的性质；若与右孩子交换，则右子树堆被破坏，且仅右子树的根结点不满足堆的性质。继续堆不满足堆性质的子树进行上述交换操作，直到叶子结点或者堆被建成为止。这种从堆顶到叶子结点的调整过程也成为筛选。



步骤

假设待调整成堆的完全二叉树存放在nums[low … high]中

1.设置初值i=low，j=2\*i+1,temp=nums[i]

令i为根节点，j=2\*i+1为根节点的左孩子，temp等于根结点

2.当j<high-1时，重复下列操作

（1）j<high-1且nums[j]>nums[j+1]，则j++ //比较孩子结点，取较小的孩子结点

（2）若temp>nums[j]，则nums[i]=nums[j]，i=j，j=2\*i+1，否则j=high+1

//当较小的子结点比当前根结点大的时候，子结点上移动

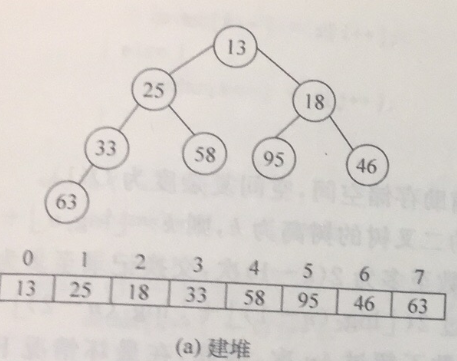
//待调整的根结点下移

//当前待比较的根节点小于较小的子结点，满足小根堆定义，退出

3.nums[i]=temp //退出循环后的i位置就是根节点所应在的位置

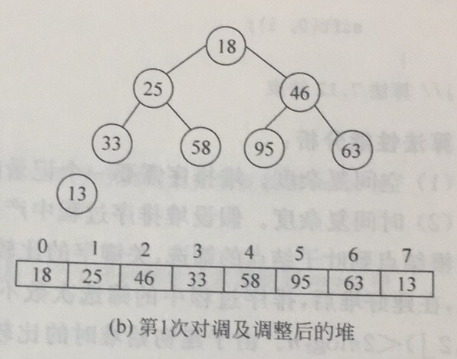
**堆排序**

将一个无序序列构造成堆。

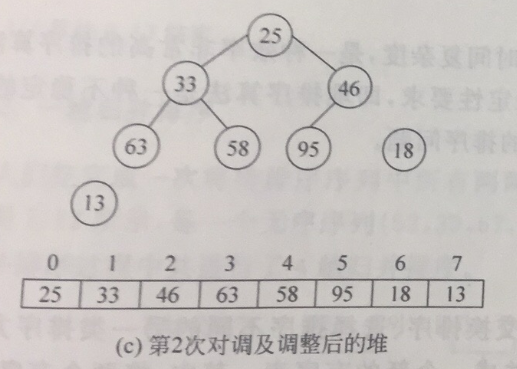


然后将堆顶次小结点与最后第2个（第n-2个结点）交换，再调整构造成第3个堆。如此反复，直至整个序列有序。

13和63交换，然后进行调整



然后18与63交换，然后进行调整



重复直到整个序列有序为止。

步骤

1.将待排序记录nums[]建成一颗完全二叉树

2.将下标为n/2（向下取整）-1的记录作为开始调整的子树的根结点

3.找出此结点的两个孩子结点中的关键字值较小者，将其与父节点比较，若父节点的关键字值较大，则交换，然后以交换后的子结点作为新的父节点，重复此步骤，直到没有子结点为止。

4.以步骤3中原来的父节点所在位置往前一个位置，作为新的调整的子树的根节点。重复步骤3，直到调整到树根。此时初始堆建成

5.堆建成后，将树根与二叉树的最后一个结点交换，再将最后一个结点输出（即原本的根结点）然后比较根结点的两个子结点，若左子结点的关键字较小，则调整左子树，反之右子树，使他再成为堆。

6.重复5，直到二叉树仅剩下一个结点为止。

**算法性能分析**

1.空间复杂度

只用了一个记录的辅助存储空间，空间复杂度为O(1)

2.时间复杂度

假设堆排序过程中产生的二叉树的高度为k，则k= O()(向下取整)+1.从根结点到叶子结点的筛选，关键字bijiao次数至多2(k-1)次，交换记录次数最多k次。由于建堆时候比较次数不超过4n次，因此再最坏情况下，堆排序的算法时间复杂度为O()

3.稳定性：不稳定

**代码**

public static void sift(int low,int high,int[] nums){ //小顶堆的调整

int i = low; //子树的根结点

int j = 2\*i+1; //子树根结点的左孩子

int temp = nums[i];

while(j<high){

if(j<high-1&&nums[j]>nums[j+1])//j为左孩子，j+1为右孩子，找到关键字最小的孩子，令j的等于它

j++;

if(temp>nums[j]){ //如果当前根结点比最小的孩子大

nums[i]=nums[j];//孩子结点上移到当前根结点的位置

i=j;//令当前根结点为刚交换的子结点，继续往下调整

j=2\*i+1;//j为当前根节点的左孩子

}

else {//找到满足堆定义的位置

break;//退出循环，当前堆已经调整完毕

}

nums[i]=temp;//将temp插入到i位置

}

}

public static void heapSort(int[] nums){

int n = nums.length;

for(int i=n/2-1;i>=0;i--){ //i=n/2-1，为完全二叉树的最后一个非叶子结点

sift(i,n,nums); //把nums调整成为堆

}

for(int i=n-1;i>0;i--){ //每趟将最小关键字与当前最后一个无序序列关键字交换

int temp = nums[0];//nums[i]和nums[n-i]交换

nums[0]=nums[i];

nums[i]=temp;

sift(0,i-1,nums);//调整成堆

}

}