2017年12月26日　　20:05

堆是一种特殊的树形结构，其每一个结点都有一个值，通常提到的堆都是指一颗完全二叉树，根节点的值小于（或大于）两个子结点的值，同时根节点的两个子树也分别是一个堆。堆一般分为最大堆和最小堆两种不同的类型。对于给定的n个记录序列（r（1），...r(n)）,当且仅当满足（r(i)>=r(2i)且r(i)>=r(2i+1)）时称为最大堆，堆顶元素为最大值，而最小堆则是r(i)<=r(2i+1)且r(i)<=r(2i)称为最小堆

1.什么是优先队列?

普通队列：先进先出；后进后出（由时间的顺序决定出队的的顺序）

优先队列：出队和入队的顺序无关；和优先级相关（急诊病人）

（1）例如计算机的操作系统会选择优先级最高的任务执行，使用优先队列可以实现这一要求。（**动态**选择优先级最高的任务进行执行）

（2）在人工智能的领域也会使用优先队列，一般会通过选择选择不同类型的敌人进行攻击。

（3）在静态处理中：例如在N个元素中选出前M个元素

①排序：NlogN

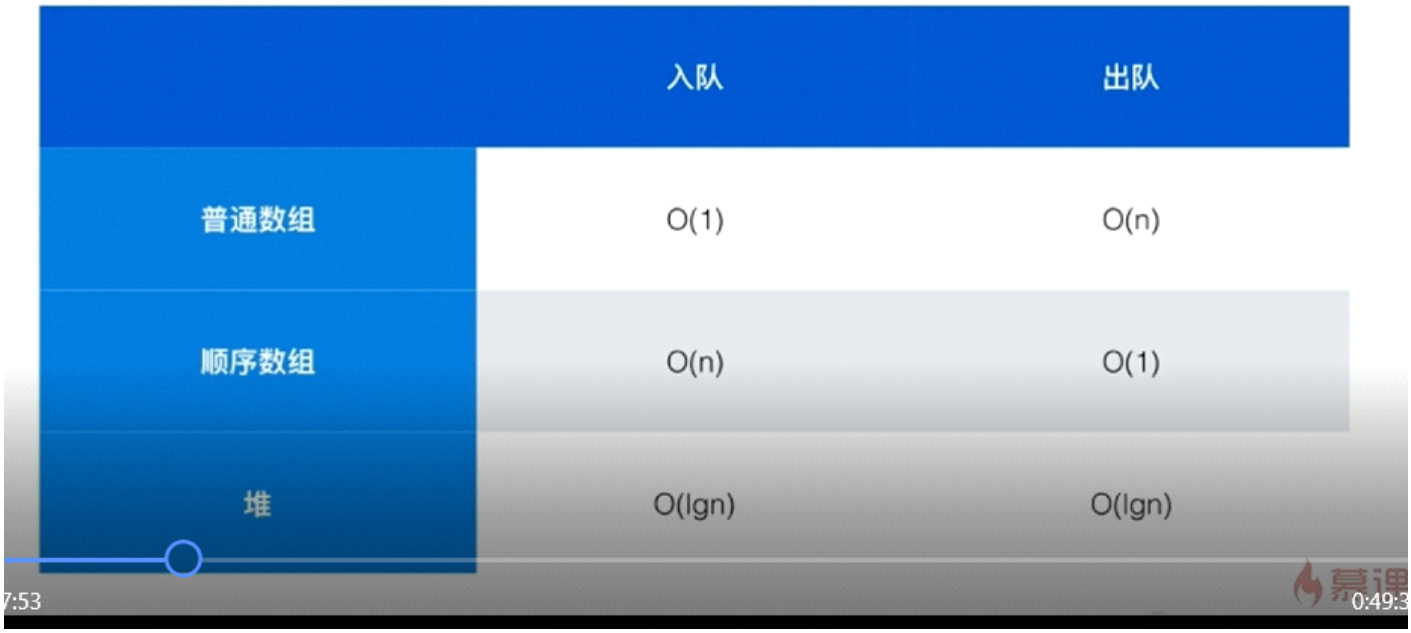
②使用优先队列：NlogM

2.优先队列主要的操作

①入队

②出队（优先队列最大的优点是取出优先级最高的元素）

优先队列的实现



虽然使用堆的操作，入队时间低于普通数组，出队时间低于顺序数组，但是在平均意义上，效率远高于数组。对于总共N个请求：使用普通数组或者顺序数组最差的情况是在O（n^2）级别的，而使用堆，其时间复杂度可以稳定在O（nlgn）级别。n方和nlgn在N很大的时候差别是巨大的。

2017年12月26日 20:22

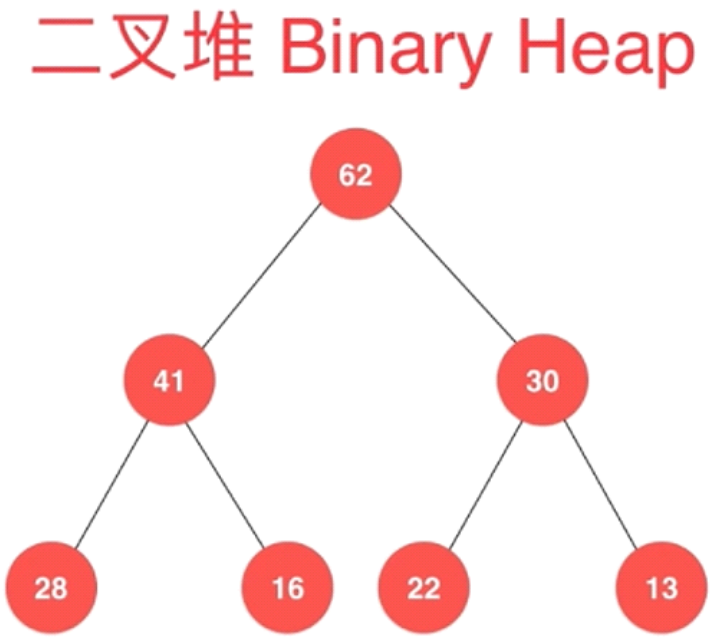
堆一定是一个树的结构

1.二叉堆（Binary Heap）：即一个节点可以分出多个子节点。

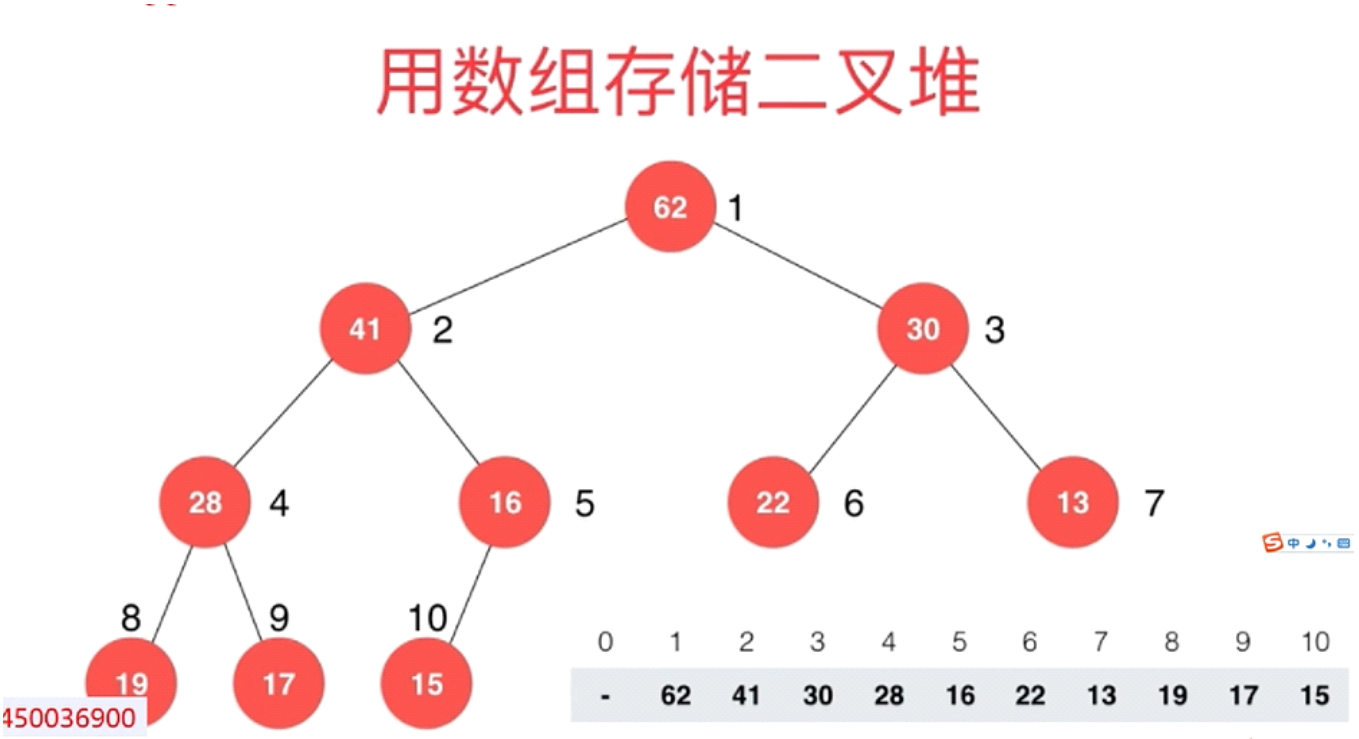
2.性质：在这个二叉树上任何一个节点都不大于其父节点（最大堆）并不意味着层数越高，数值越大；堆总是一棵完全二叉树：二叉树除了最后一层之外，其他层的节点数目必须是最大值。（第一层只有一个节点，第二层有两个节点，第三层有四个节点。。。。最后一层虽然可以不满，但必须是从左左右依次排开的）

注意：并不意味着层数越高，数值越大。

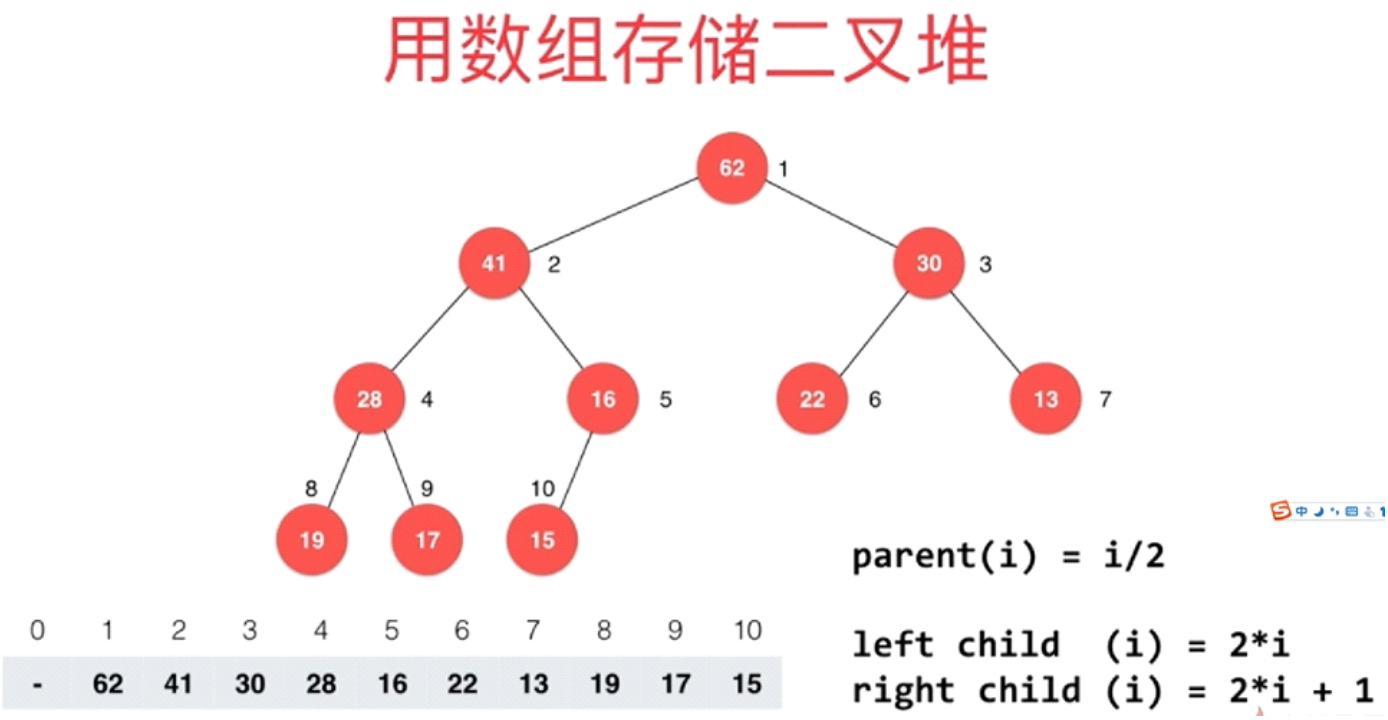
3.堆中某个节点的值总是不大于其父节点的值称为最大堆。



4.用数组存储二叉堆（因为堆是一个完全的二叉树）对于对来说最经典的实现就是根节点从1开始标记，也可以将根结点用0来表示

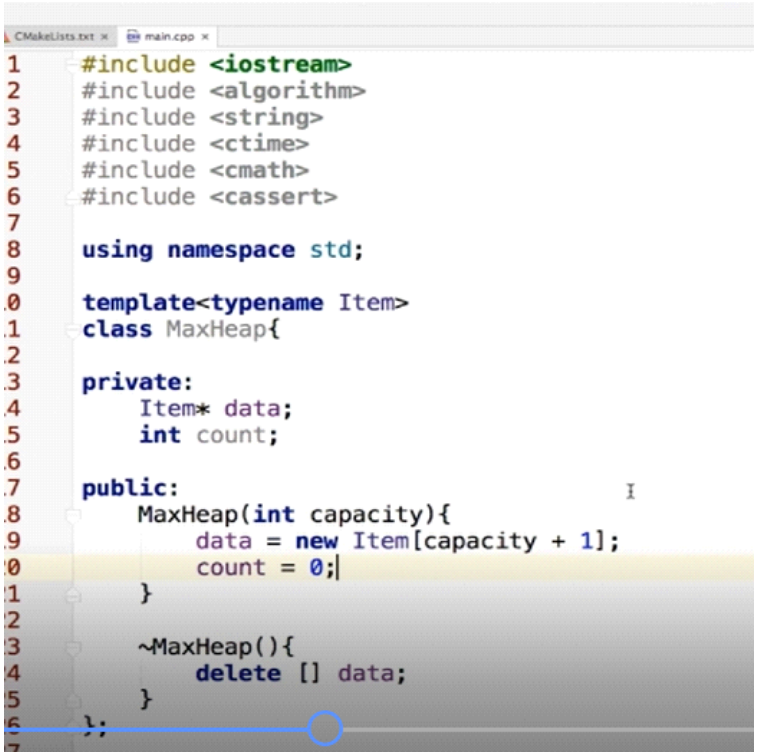


可以看出左节点是父节点的2倍。右节点是父节点的2倍加1。（0号索引不使用）



这个除法使用的是计算机的除法，除不开就取整

定义一个私有变量，在构造函数中对这个变量进行初始化，开辟相应的空间

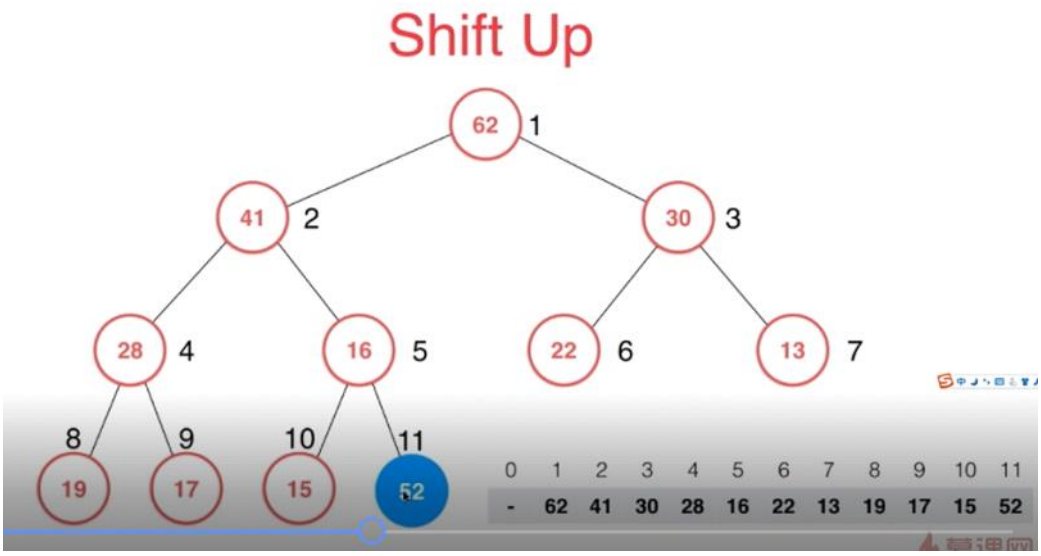


```cpp
#include <iostream>
#include <algorithm>
#include <string>
#include <ctime>
#include <cmath>
#include <cassert>

using namespace std;

template<typename Item>
class MaxHeap{

private:
    Item* data;
    int count;

public:
    MaxHeap(int capacity){
        data = new Item[capacity + 1];
        count = 0;
    }

    ~MaxHeap(){
        delete [] data;
    }
};
```

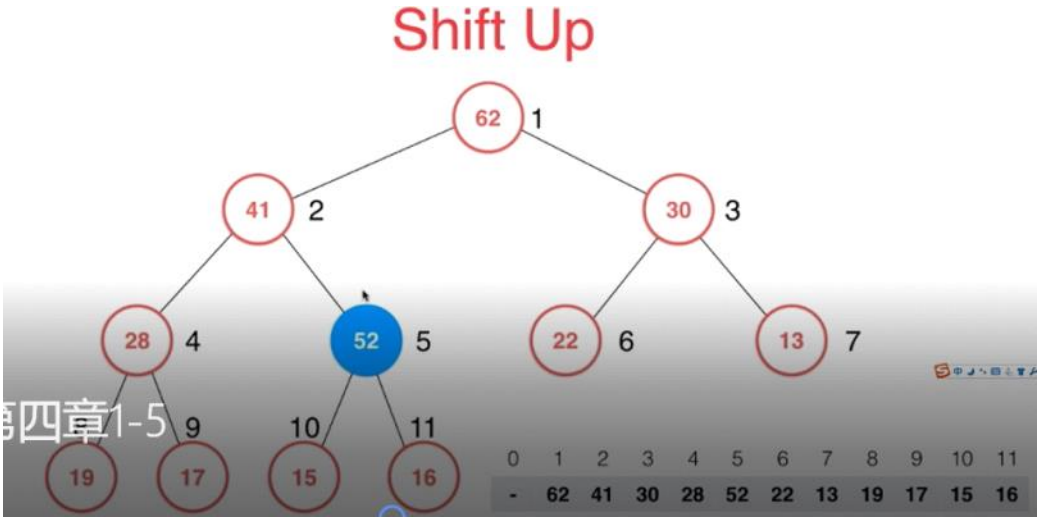
5.代码分析

```
public class maxHeap<Item>{
    private Item[] data;
    private int count;

    //构造函数，构造一个空堆，可容纳capacity个元素
    public maxHeap(int capacity){
        data=(Item[])new Object[capacity+1];
        count=0;
    }

    //返回堆中的元素个数
    public int size(){
        return count;
    }

    //返回一个布尔值，表示堆中是否为空
    public boolean isEmpty(){
        return count==0;
    }
    public static void main(String[] args) {
        maxHeap<Integer> maxHeap1=new maxHeap<Integer>(100);
        System.out.println(maxHeap1.size());
    }

}
```

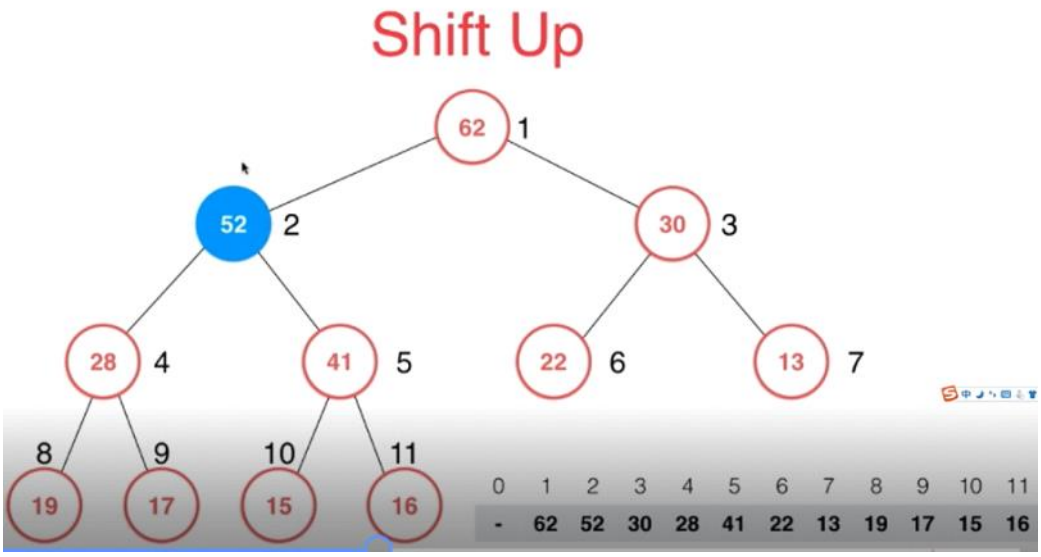
6.如何向一个堆中添加元素（shift up操作）



将新加入的元素和其父节点进行比较



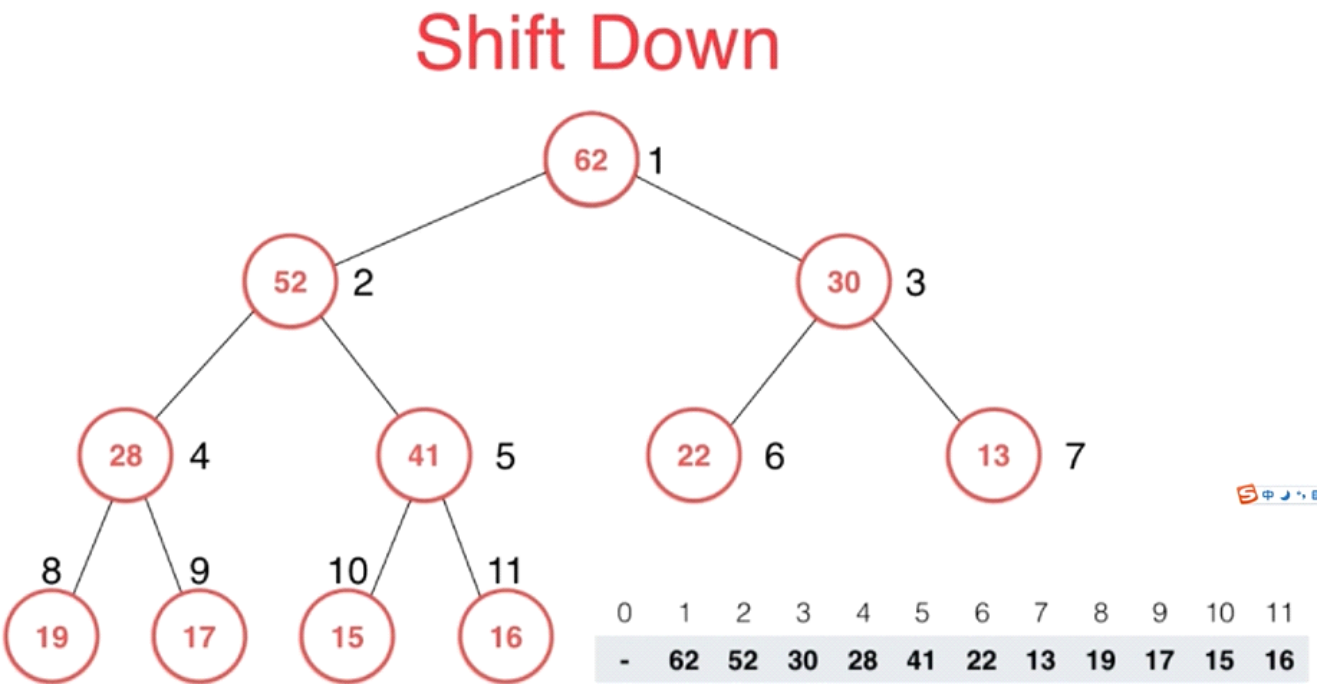
再接着比较52的新位置和其父节点的大小



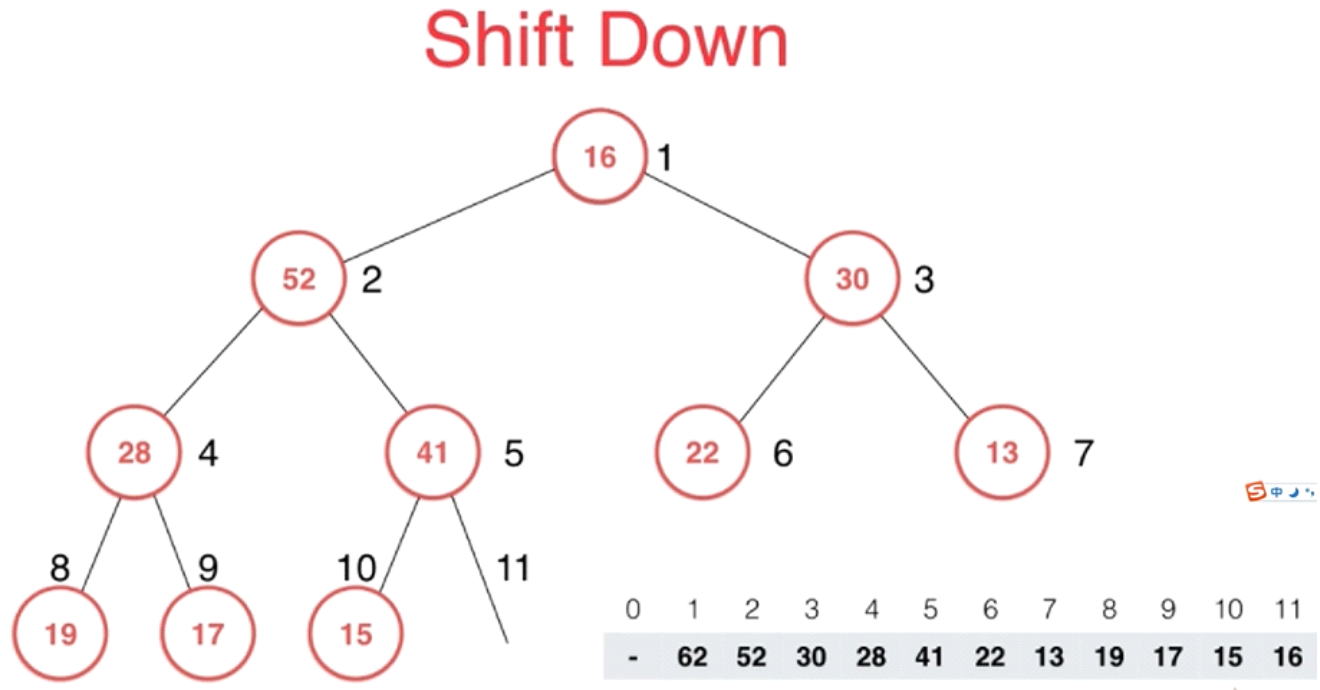
再将52是不是比62小，不用继续交换了。

7.代码分析：

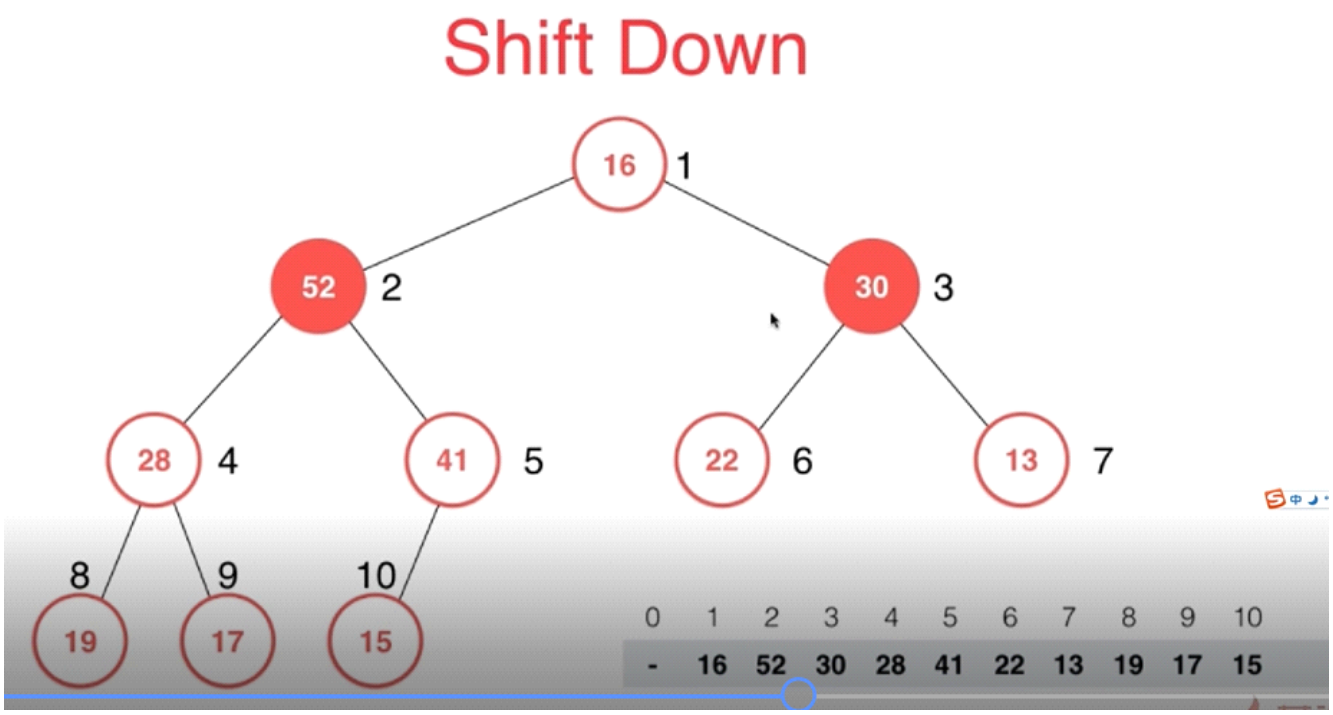
8.Shift Down（从堆中取出元素只能取出根节点的元素，即优先级最大的点的元素）



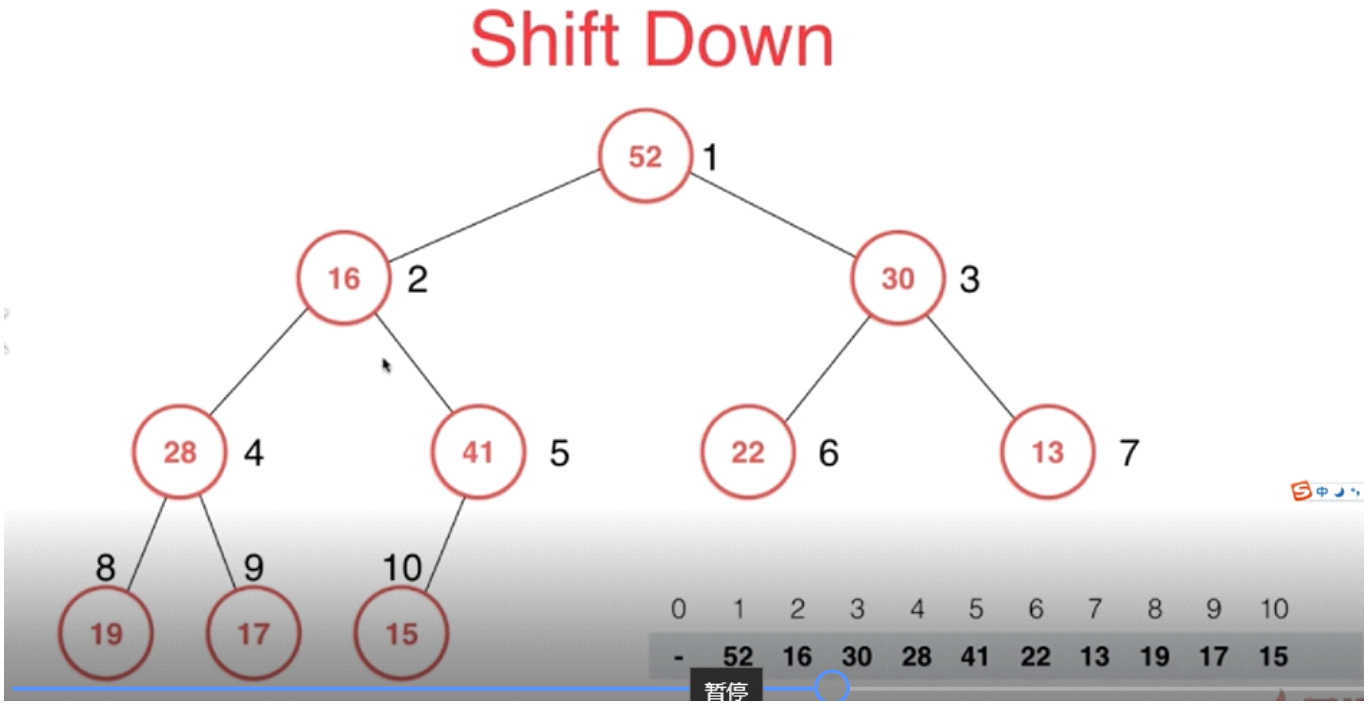
首先将根结点的位置和数组中最后一个元素进行交换



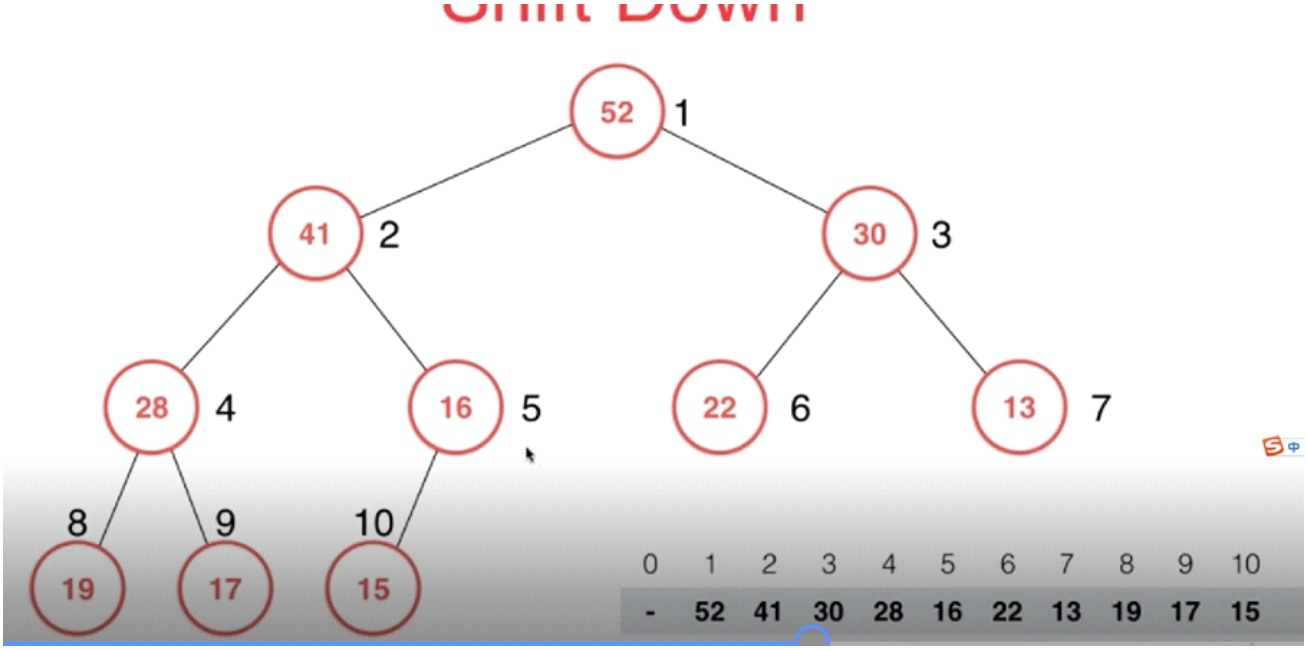
Count--；但是此时完全二叉树不是最大树，接着就调整元素的位置。



比较左右两个孩子，谁大和谁换，所以和52换



再接着比较28和41的大小，由于16比他的左右孩子都要小，将16和41进行交换



发现此时只有一个孩子，因此比较16和15的小，在此情景下不用进行交换。

以上就是最大堆的所有实现方法，**曾经有很多大公司的面试过程中就叫你使用白板编程的方式实现堆的一系列操作。**

已经可以使用堆来进行一些数组的排序操作了

9.代码分析：

```
public class Maxheap<Item extends Comparable> {

    protected Item[] data;
    protected int count;
    protected int capacity;

    //定义构造函数，构造一个空堆，可以容纳capacity个元素
    public Maxheap(int capacity){
        data=(Item[])new Comparable[capacity+1];
        count=0;
        this.capacity=capacity;
    }
    //返回堆中的元素个数
    public int size(){
        return count;
    }
    //判断堆时否为空
    public boolean isEmpty(){
        return count==0;
    }
    //向最大堆中插入一个新的元素值item
    public void insert(Item item){
        assert count+1 <=capacity;
        data[count+1]=item;
        count++;
        shiftUp(count);
    }
```

```
private void shiftUp(int m) {
        while(m>1&&data[m].compareTo(data[m/2])>0){
                swap(m, m/2);
                m=m/2;
        }

}
//从最大堆中取出堆顶元素，即堆中存储的最大数据
public Item extractMax(){
        assert count>0;
        Item ret=data[1];
        swap(1,count);//将根结点的元素和数组中最后一个元素交换
        count--;
        shiftDown(1);
        return ret;
}

private void shiftDown(int k) {
        //首先判断节点是否有孩子，方法是判断是不是有左孩子
        while(2*k<=count){
                int j=2*k;//在次循环中，data[k]有可能和data[j]互换
                if(j+1<=count&&data[j+1].compareTo(data[j])>0){//如果存在右孩子
                        j++;
                }
                if(data[k].compareTo(data[j])>=0)
                        break;
                swap(k,j);
                k=j;
        }
}

//交换索引位置是i和j的两个元素
private void swap(int i, int j) {
        Item temp=data[i];
        data[i]=data[j];
        data[j]=temp;

}
public static void main(String[] args) {
        Maxheap<Integer> maxheap=new Maxheap<Integer>(10);
        int N=10;//堆中元素的个数
        int M=10;//堆中元素的取值范围[0,M)
        Integer[] arr=new Integer[N];
        for(int i=0;i<N;i++){
                maxheap.insert(new Integer((int) (Math.random()*M)));
                arr[i]=new Integer((int) (Math.random()*M));
                System.out.print(arr[i]+"  ");
        }
        System.out.println();
        //将Maxheap中的数据逐渐使用extractMax取出来
        //取出来的顺序应该是按照从大到小的顺序取出来的
   for(int i=0;i<N;i++){
          arr[i]=new Integer(maxheap.extractMax());
          System.out.print(arr[i]+"  ");
   }
   System.out.println();
   //确保arr数组是从大到小排列的
   for(int i=1;i<N;i++)
          assert arr[i-1]>=arr[i];
```

```
    }
}
```

其中程序中有一个特别需要注意的地方是

在插入数值时随机赋值的时候

```
For(int i=0;i<N;i++)
Maxheap.insert(new Integer( (int )
(Math.random()*M)) )
```

这里的括号问题要处理好不然输出会全部都是零

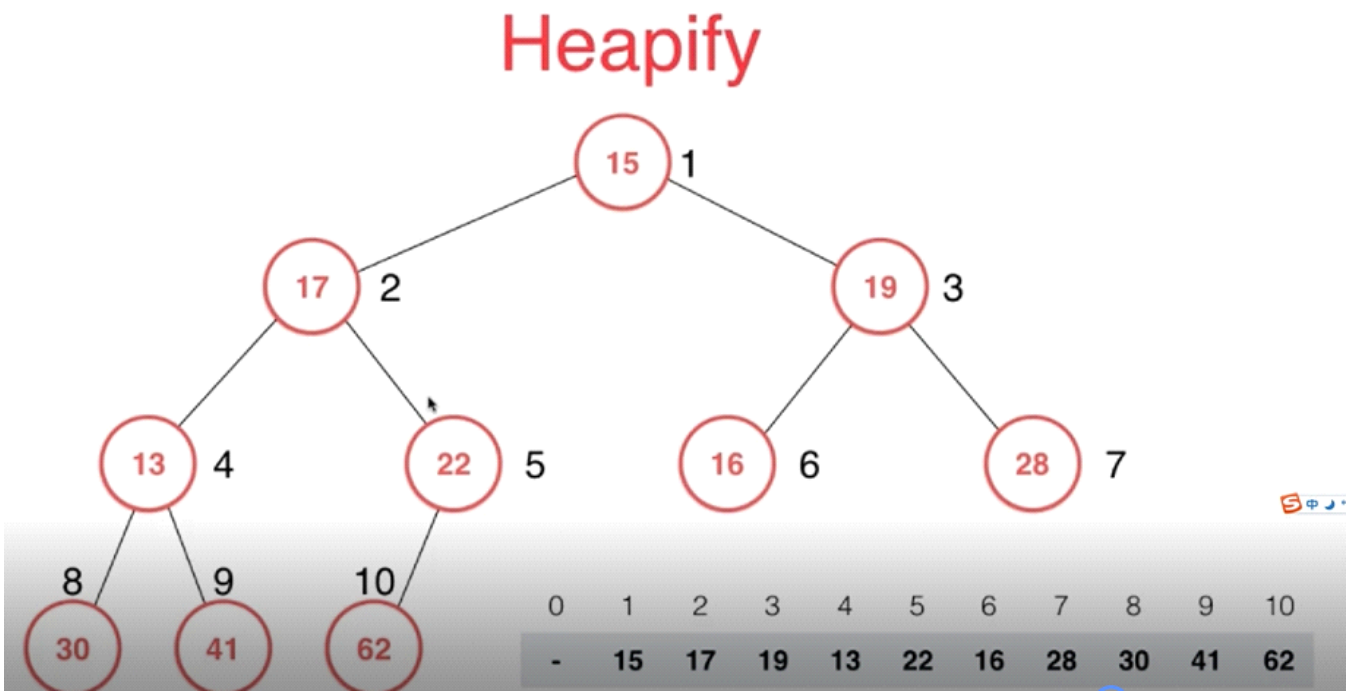
2017年12月27日 14:13

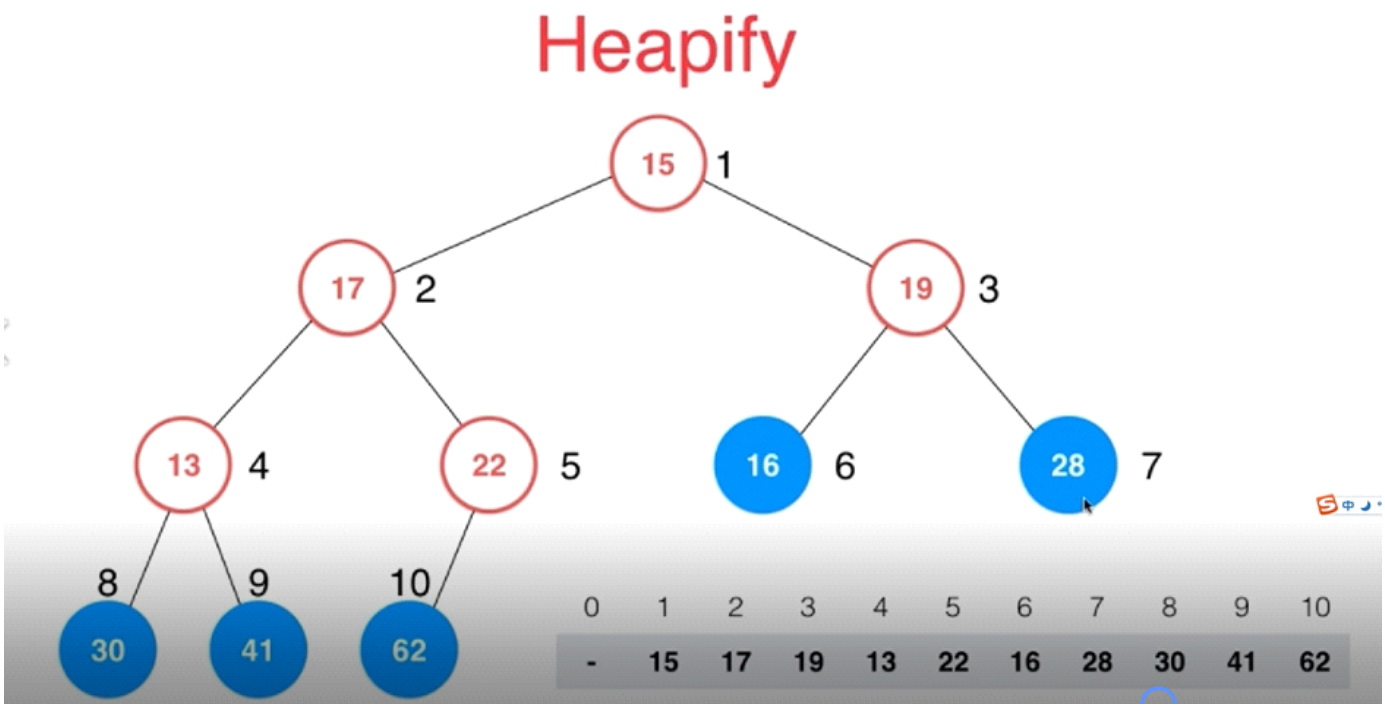
注意只要有索引就必须考虑索引越界的问题

1.调用shiftUp和shiftDown操作，将数组写入堆中之后，再从堆中取出，此时数组就已经排好了相应的顺序。

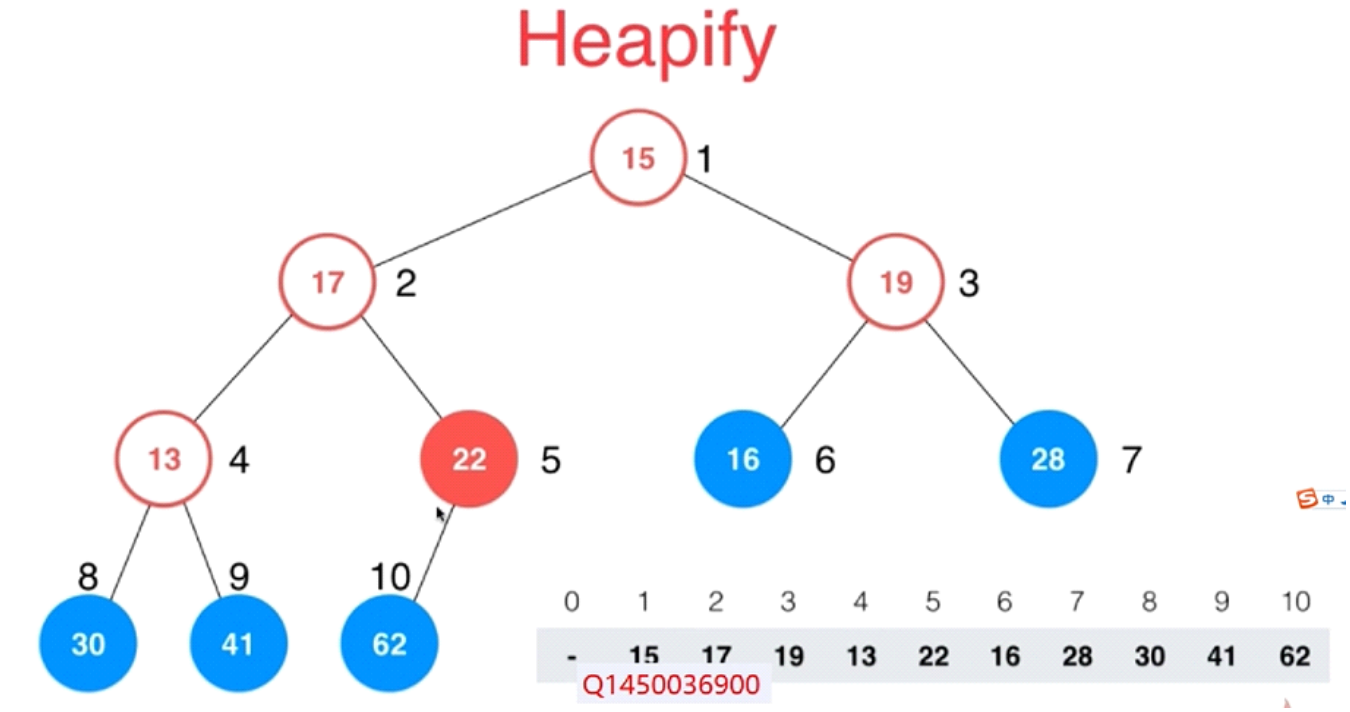
```
public class HeapSort1{
    public static void sort(int[] arr){
        Int n=arr.length;
        Maxheap<Comparable> maxheap=new Maxheap<Comparable>(n);
        //将数组写入堆中
        for(int i=0;i<n;i++){
            maxheap.insert(arr[i]);
            System.out.print(arr[i]+"  ");
        }
        System.out.println();
        //依次从数组中取出相应的元素，出队
        for(int i=n-1;i>0;i--){
            arr[i]=(int)maxheap.extractMax();
            //syso(arr[i]+"  ");
        }
        //打印输出数组
        for(int i=0;i<n;i++){
            System.out.print(arr[i]+"  ");
        }

        //生成随机序列
    public static int[] generateRandomArray(int n,int rangeL,int rangeR){
        int arr[]=new int[n];
        for(int i=0;i<n;i++){
            arr[i]=(int)(Math.random()*(rangeR-rangeL))+rangeL;
        }
        return arr;
    }
    //主函数
    Public static void main(String[] args){
        int[] arr=generateRandomArray(10,0,10);
        sort(arr);
    }
```

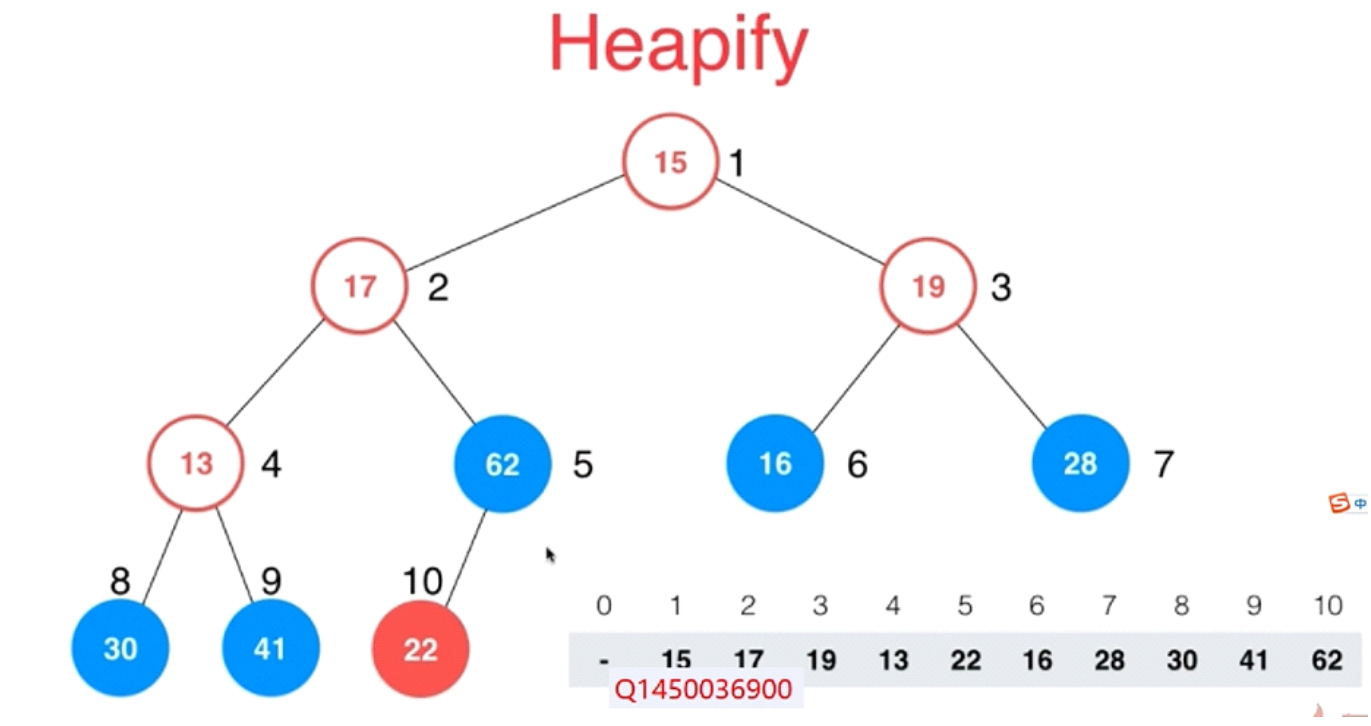
2、给定一个数组，使数组的排列以堆的形式表现出来，这种操作叫做Heapify。



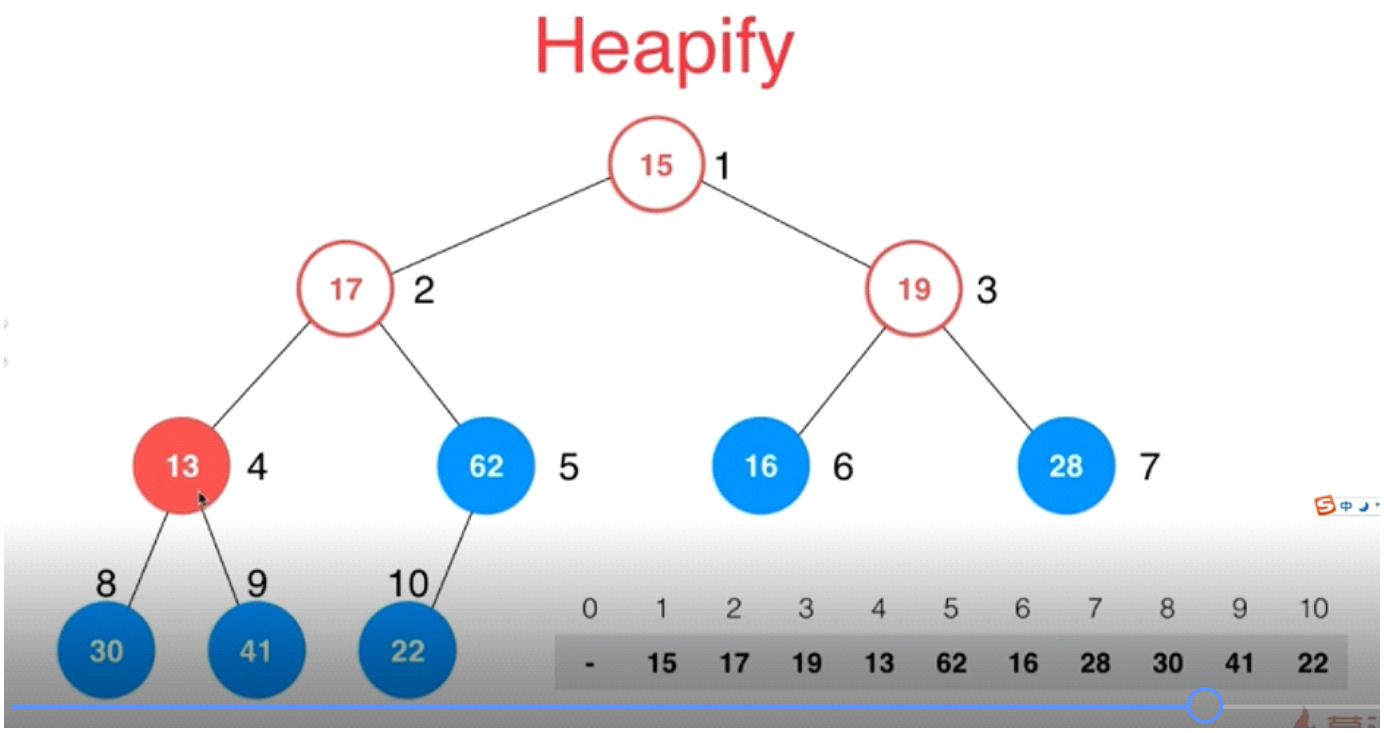


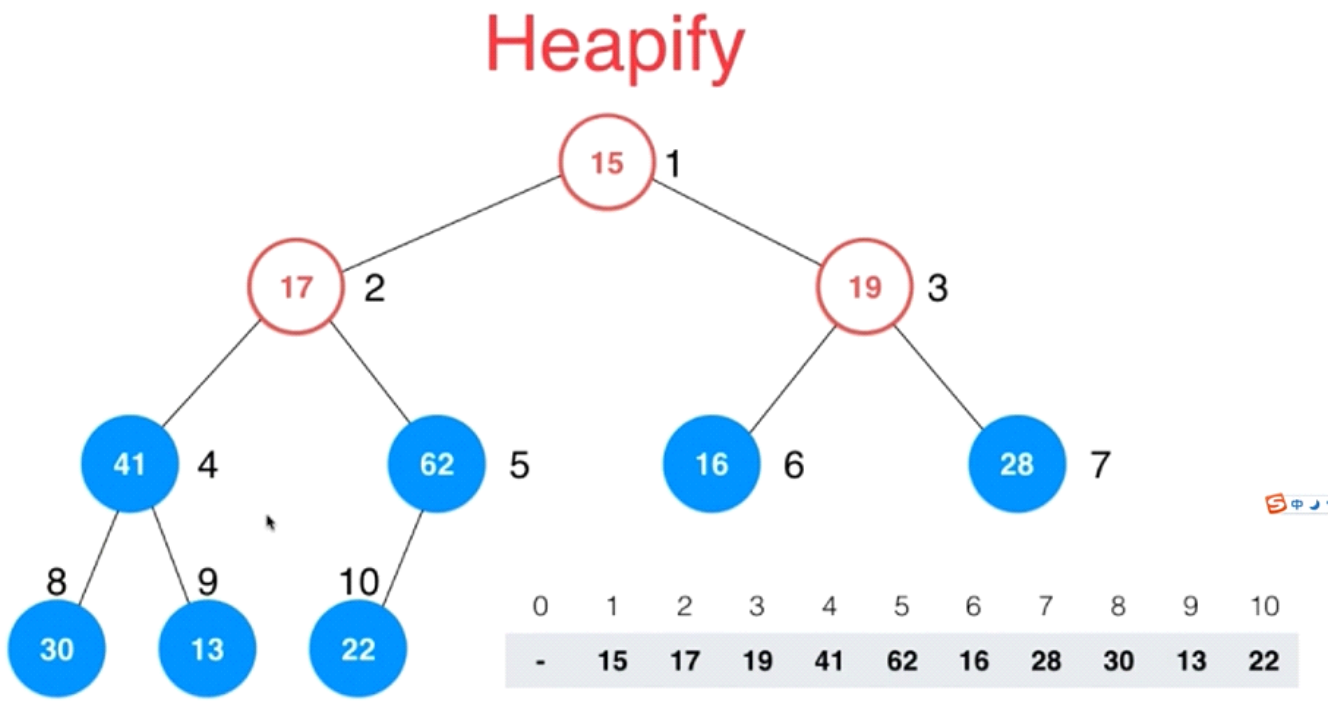
现在这个完全二叉树还不是一个最大堆，所有的叶子结点（没有孩子的节点即蓝色的部分）本身就是一个最大堆，只是每个堆中都只有一个元素。注意，对于一个完全二叉树来说，第一个非叶子节点的索引是（10/2=5），即节点5是对应的第一个不是叶子的元素对应的索引。则第一个要考察的是22，操作很简单就是在22元素所在的位置上执行shiftDown操作即可。



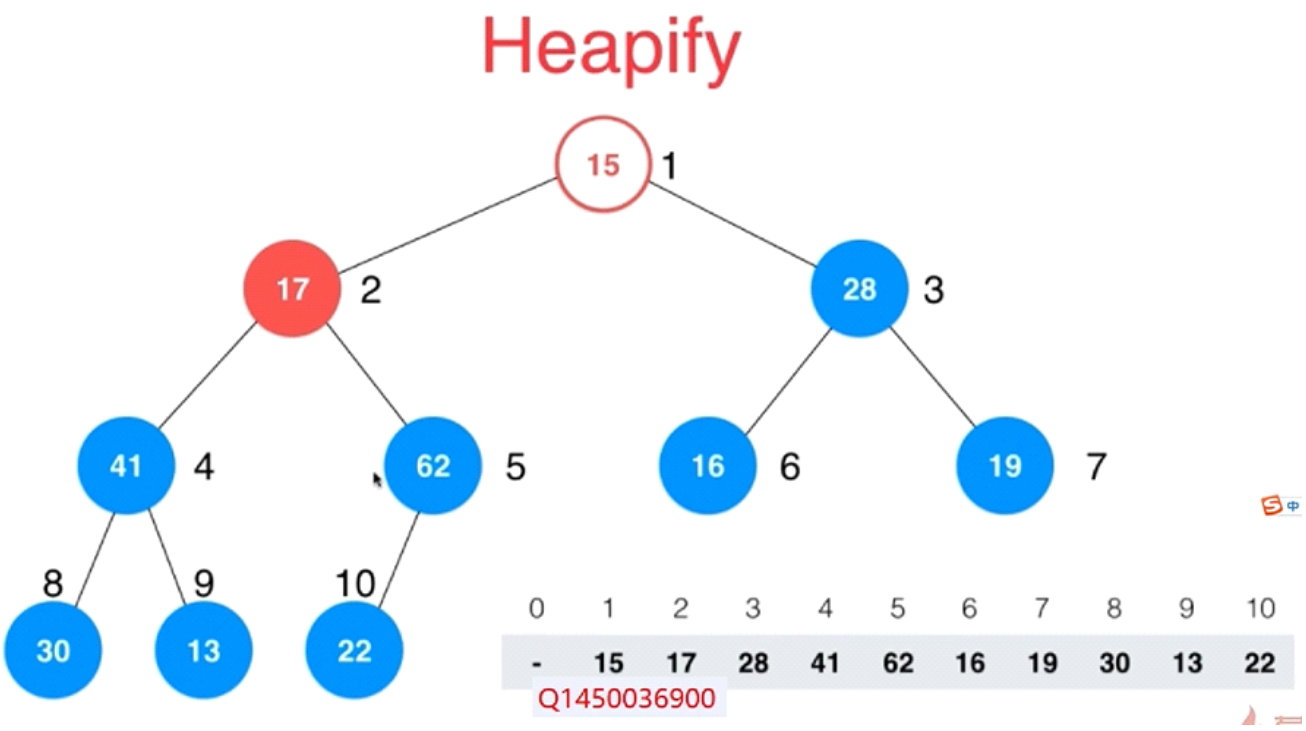


接下来考察13这个元素，它比两个孩子都要小，但是最大的是41，所以将13和41交换下位置

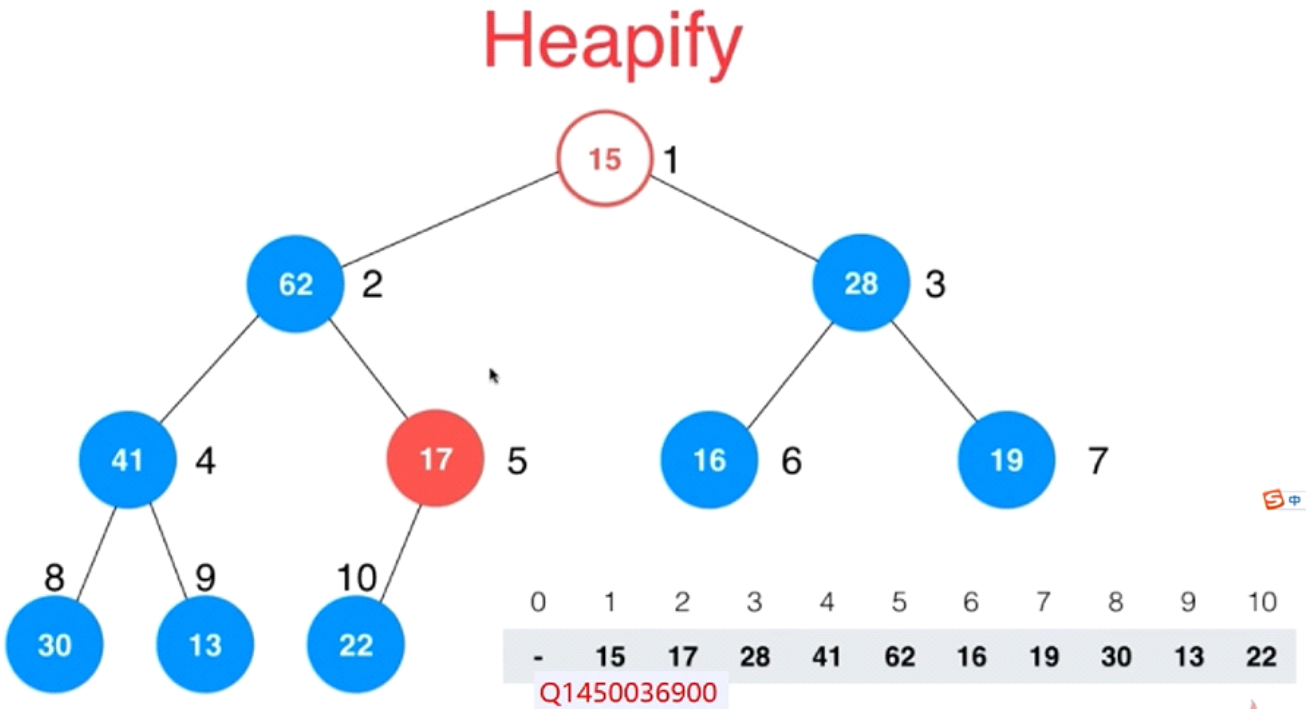


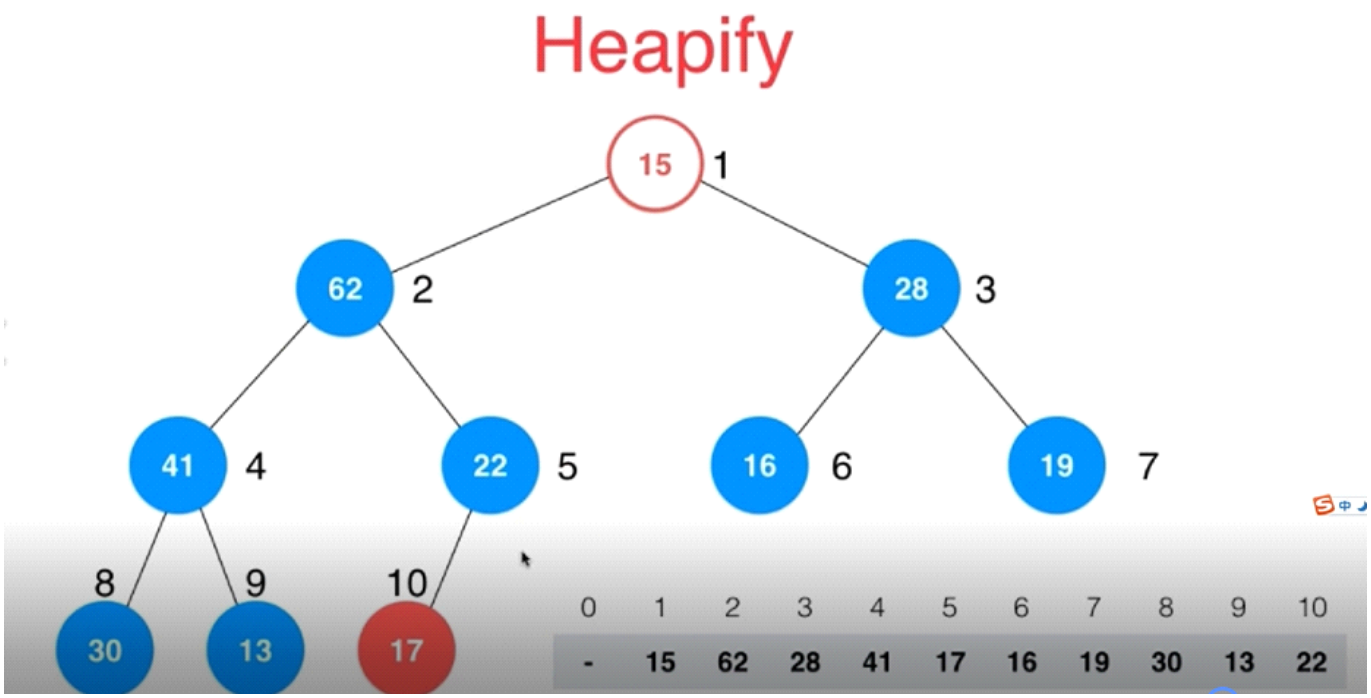


接着考察索引3的元素，同样的道理，将19和28调换一下位置

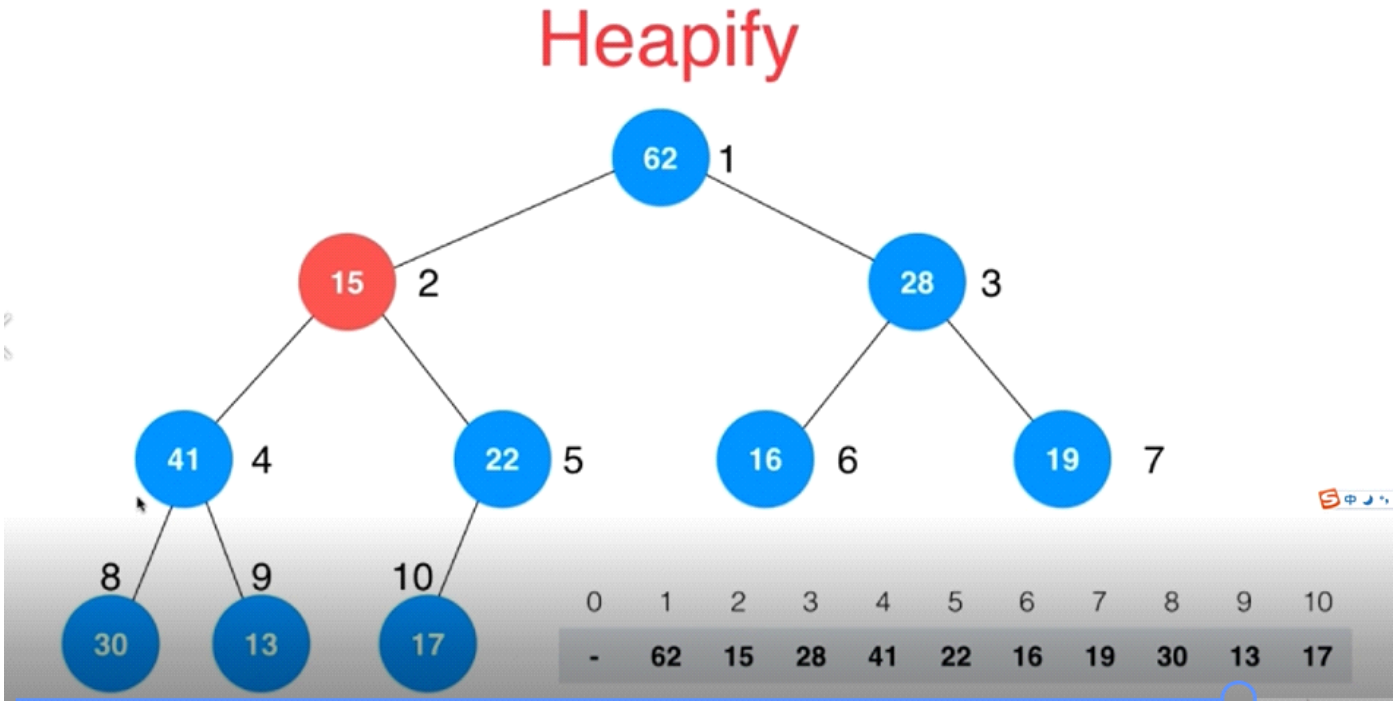


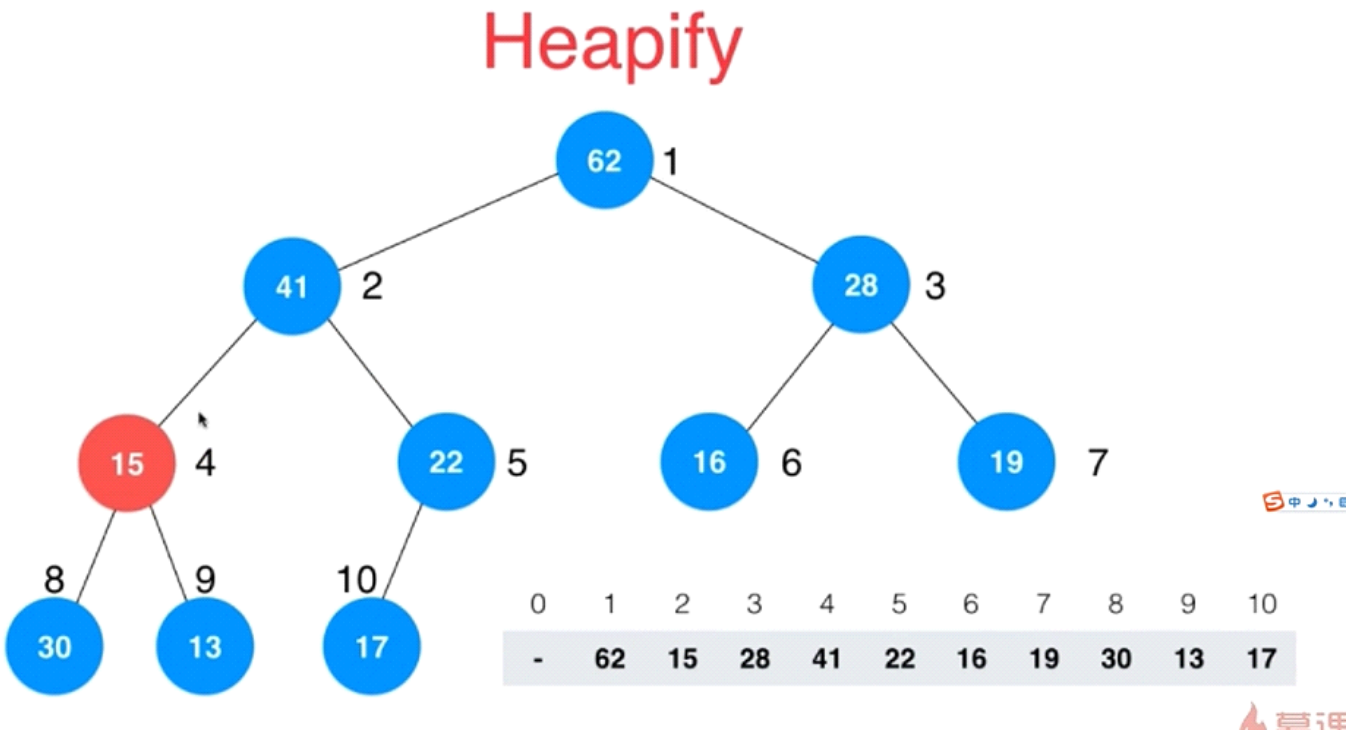
接着考察索引2所在的元素，17<62,所以交换一下位置，之后没完，因为作为索引5的元素比索引10指向的22还要小，所以交换一下位置，此时以索引2为节点的索引也满足了堆的性质。

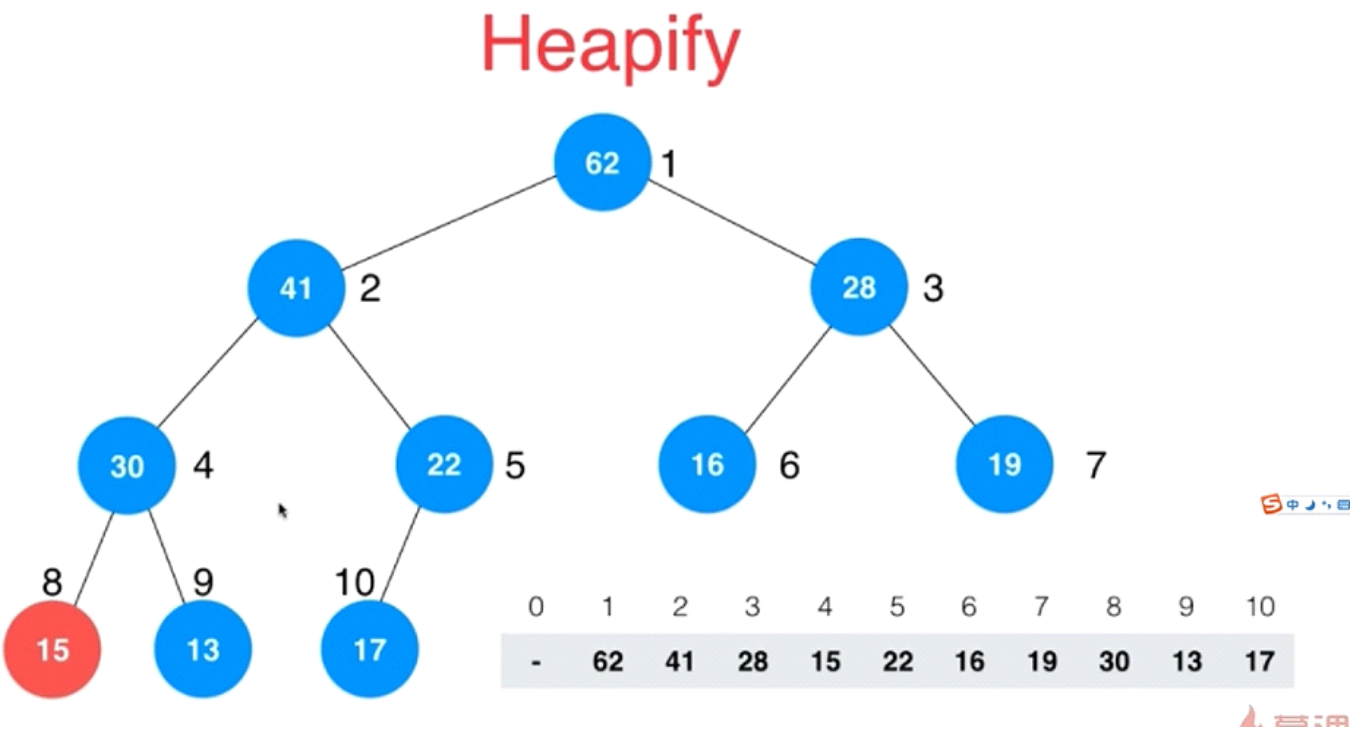




接着考察索引1指向的元素15，由于15<62,所以交换一下位置，索引2指向的15小于41和22，所以交换下位置，此时索引2指向元素41.索引4指向元素15，由于15仍小于30，再次交换位置，得到最后结果







代码实现：

用Heapity实现堆更快；

将n个元素逐个插入一个空堆中的算法复杂度是O（nlogn）

heapify的过程，算法复杂度使O（n）（不是数学，证明忽略，记住结论即可），这个过程上来就直接将n/2的元素都抛出去了，所以应该更快一

些。

```
            k = j;
        }
    }

public:
    MaxHeap(int capacity){
        data = new Item[capacity + 1];
        count = 0;
        this->capacity = capacity;
    }

    MaxHeap(Item arr[], int n){

        data = new Item[n+1];
        capacity = n;
        for( int i = 0 ; i < n ; i ++ )
            data[i+1] = arr[i];
        count = n;

        for( int i = count/2 ; i >= 1 ; i -- )
            shiftDown(i);
    }

```

执行完这个操作之后，可以将arr数组转化成一个最大堆。

```
template<typename T>
void heapSort2(T arr[], int n){

    MaxHeap<T> maxheap = MaxHeap<T>(arr, n);
    for( int i = n-1 ; i >= 0 ; i -- )
        arr[i] = maxheap.extractMax();
}
```

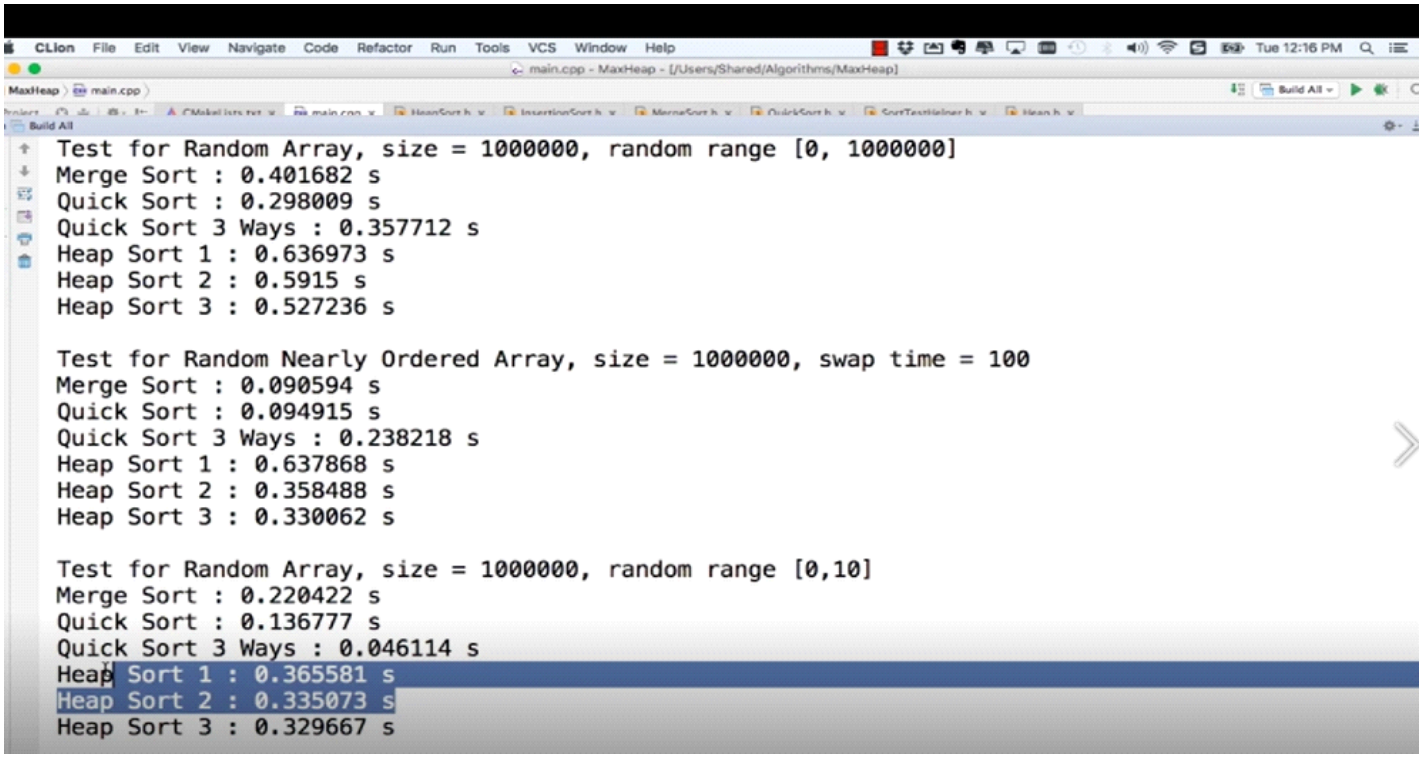
public class HeapSort2 {	
	// 我们的算法类不允许产生任何实例
	private HeapSort2(){}
	// 对整个arr数组使用HeapSort2排序
	// HeapSort2, 借助我们的heapify过程创建堆
	// 此时, 创建堆的过程时间复杂度为O(n), 将所有元素依次从堆中取出来, 实践复杂度为O(nlogn)
	// 堆排序的总体时间复杂度依然是O(nlogn), 但是比HeapSort1性能更优, 因为创建堆的性能更优
	public static void sort(Comparable[] arr){
	int n = arr.length;
	MaxHeap<Comparable> maxHeap = new MaxHeap<Comparable>(arr);
	for(int i = n-1 ; i >= 0 ; i --)
	arr[i] = maxHeap.extractMax();
	}
	// 测试 HeapSort2
	public static void main(String[] args) {

```
        int N = 1000000;
        Integer[] arr = SortTestHelper.generateRandomArray(N, 0, 100000);
        SortTestHelper.testSort("bobo.algo.HeapSort2", arr);
        return;
    }
}
```

来自 <https://github.com/liuyubobobo/Play-with-Algorithms/blob/master/04-Heap/Course%20Code%20(Java)/05-Heapify/src/bobo/algo/HeapSort2.java>

2017年12月28日 9:44

这个排序不需要进行空间开辟，因此执行效率比前面两种都还要高




```
//不使用一个额外的最大堆，直接在原数组上进行原地排序
public class heapSort {
    //进行heapify过程将数组构建一个堆，索引位置从最后一个叶子节点开始
    public static void sort(int[] arr){
        int n=arr.length;
        for(int i=(n-1)/2;i>=0;i--)
            shiftDown(arr,n,i);
    //进行排序操作
    for(int i=n-1;i>0;i--){
        swap(arr,0,i);//将当前堆中最大的元素放在合适的位置中
        shiftDown(arr,i,0);//每次循环，堆中应该有i个元素，对其中第0个操作进行shiftDown操作即可
    }
    System.out.println();
    //打印输出
    for(int i=0;i<arr.length;i++)
        System.out.print(arr[i]+"   ");
}

    private static void swap(int[] arr,int i, int j) {
        int temp=arr[i];
        arr[i]=arr[j];
```

```
            arr[j]=temp;
        }

        private static void shiftDown(int[] arr, int n, int k) {
            while(2*k+1<n){
                int j=2*k+1;
                if(j+1<n&&arr[j+1]>arr[j])
                    j=j+1;//至此得到了第k个节点左右孩子中比较大的一个孩
                    子
                if(arr[k]<arr[j])
                    swap(arr,k, j);
                else
                    break;
                    k=j;
            }
        }

        public static int[] generateRandonArray(int n,int rangeL,int rangeR){
            int[] arr=new int[n];
            for(int i=0;i<n;i++){
                arr[i]=(int) (Math.random()*(rangeR-rangeL)+rangeL);
                System.out.print(arr[i]+"   ");
            }
            return arr;
        }

        //打印输出数组
//      public static void    printSort(int[] arr){
//              for(int i=0;i<arr.length;i++)
//                  System.out.print(arr[i]+"   ");
//
//      }
        public static void main(String[] args) {
            int[] arr=generateRandonArray(10, 0, 100);
            sort(arr);
            System.out.println();
            //printSort(arr);
        }
}
```

排序算法的比较

2017年12月28日　　10:35

注意是：平均时间复杂度，例如对于插入排序而言，如果序列本身是很有序的情况下，有可能退化到o（n）级别，对于快速排序而言有可能退化成O（n^2）级别的。

①总体来说，对于三种nlogn的来说，快速排序也是很快的

②原地排序是指直接在数组上进行操作进而实现排序过程，而归并排序必须开辟额外的空间才能完成排序的过程，其他三种都可以进行原地排序。

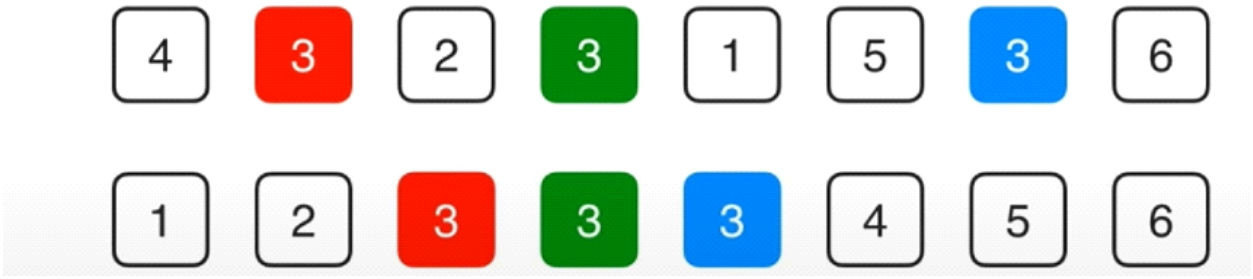
③快速排序采用递归的方式进行排序，这个过程一共有logn这么多层，这么多层的递归中栈空间就需要有这么多个递归临时变量，一共递归返回的时候进行使用

④排序算法的稳定性

（1）稳定排序：对于相等的元素，在排序后，原来靠前的元素依然靠前，相等元素的相对位置没有发生改变。

稳定排序：对于相等的元素，在排序后，原来靠前的元素依然靠前。

相等元素的相对位置没有发生改变。



关于插入排序

首先3和8比，3比8小，则3和8交换位置。接着3和6接着比，则3和6接着交换位置；接着红色的3和蓝色的3相比一样大，则不用交换位置，所以插入排序是稳定排序。

关于归并排序

首先比较2和1，1小所以将1放在数组的第一个位置，接着比较2和3的大小，2小，则将2放在

数组的第二个位置，接着比较3和3的大小，因为相等则将前面的3放在a数组的第3个位置，右边的3放在数组的第四个位置，接着比较6和5的大小，所以归并排序算法也是稳定的排序。（在归并排序过程中当n很小的时候，使用插入排序也保证了数组的稳定性）

另外，关于数组的稳定性，是随着实际情况决定的

？？？？？？？？？？？？？？？？？

	平均时间复杂度	原地排序	额外空间	稳定排序
插入排序 Insertion Sort	O(n^2)	✓	O(1)	✓
归并排序 Merge Sort	O(nlogn)	×	O(n)	✓
快速排序 Quick Sort	O(nlogn)	✓	O(logn)	×
堆排序 Heap Sort	O(nlogn)	✓	O(1)	×
神秘的排序算法?	O(nlogn)	✓	O(1)	✓

这种神秘的算法还不存在

2.各种排序算法的优势

各种排序的稳定性，时间复杂度和空间复杂度总结：

类别	排序方法	时间复杂度			空间复杂度	稳定性
		平均情况	最好情况	最坏情况	辅助存储	
插入排序	直接插入	$O(n^2)$	O(n)	$O(n^2)$	O(1)	稳定
	Shell排序	$O(n^{1.3})$	O(n)	$O(n^2)$	O(1)	不稳定
选择排序	直接选择	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(1)	不稳定
	堆排序	$O(nlog_2n)$	$O(nlog_2n)$	$O(nlog_2n)$	O(1)	不稳定
交换排序	冒泡排序	$O(n^2)$	O(n)	$O(n^2)$	O(1)	稳定
	快速排序	$O(nlog_2n)$	$O(nlog_2n)$	$O(n^2)$	$O(nlog_2n)$	不稳定
归并排序		$O(nlog_2n)$	$O(nlog_2n)$	$O(nlog_2n)$	O(n)	稳定
基数排序		O(d(r+n))	O(d(n+rd))	O(d(r+n))	O(rd+n)	稳定

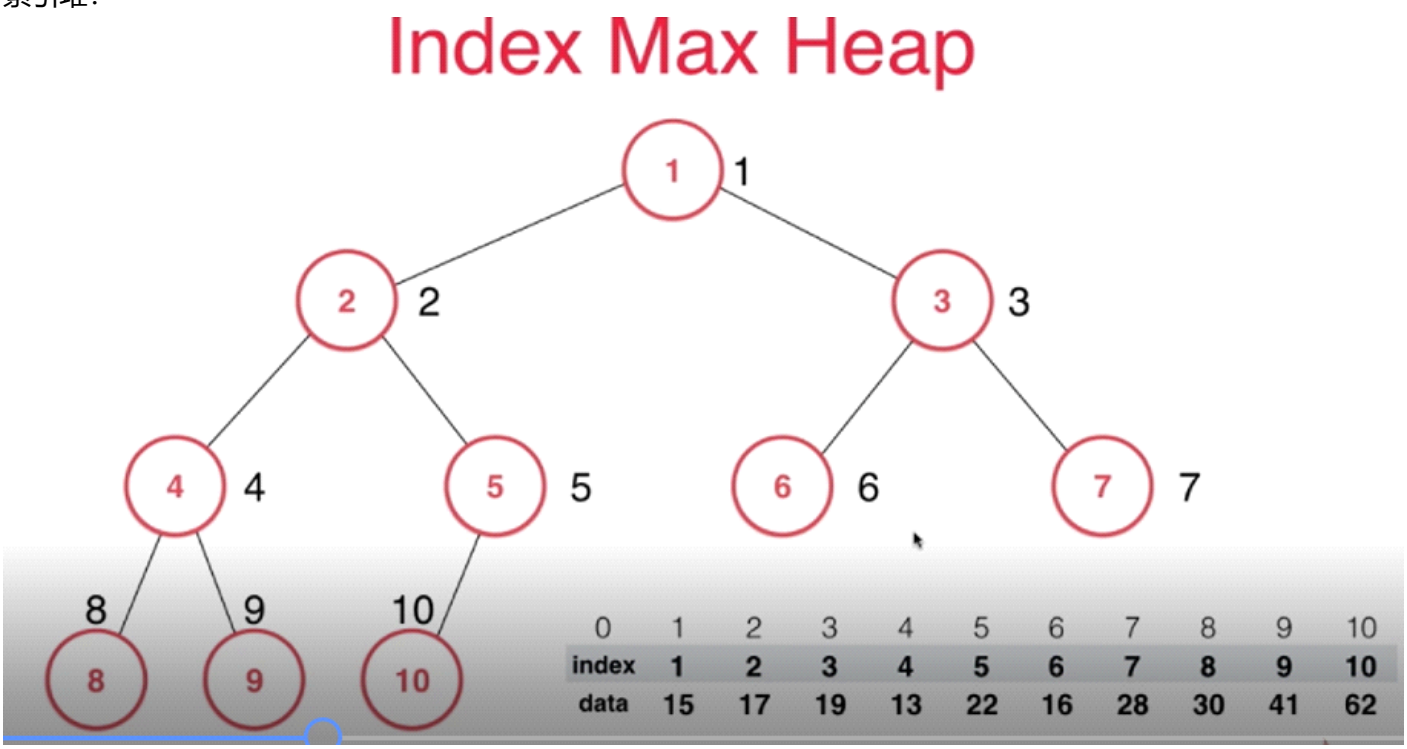
注：基数排序的复杂度中，r代表关键字的基数，d代表长度，n代表关键字的个数。

类别	排序方法	时间复杂度			空间复杂度	稳定性
		平均情况	最好情况	最坏情况		
插入排序	插入排序	O(N^2)	O(N)	O(N^2)	O(1)	稳定
	Shell 排序	O(N^1.3)	O(N)	O(N^2)	O(1)	不稳定
选择排序	选择排序	O(N^2)	O(N^2)	O(N^2)	O(1)	不稳定
	堆排序	O(N*lgN)	O(N*lgN)	O(N*lgN)	O(1)	不稳定
交换排序	冒泡排序	O(N^2)	O(N)	O(N^2)	O(1)	稳定
	快速排序	O(N*lgN)	O(N*lgN)	O(N^2)	O(lgN)	不稳定
归并排序	归并排序	O(N*lgN)	O(N*lgN)	O(N*lgN)	O(N)	稳定

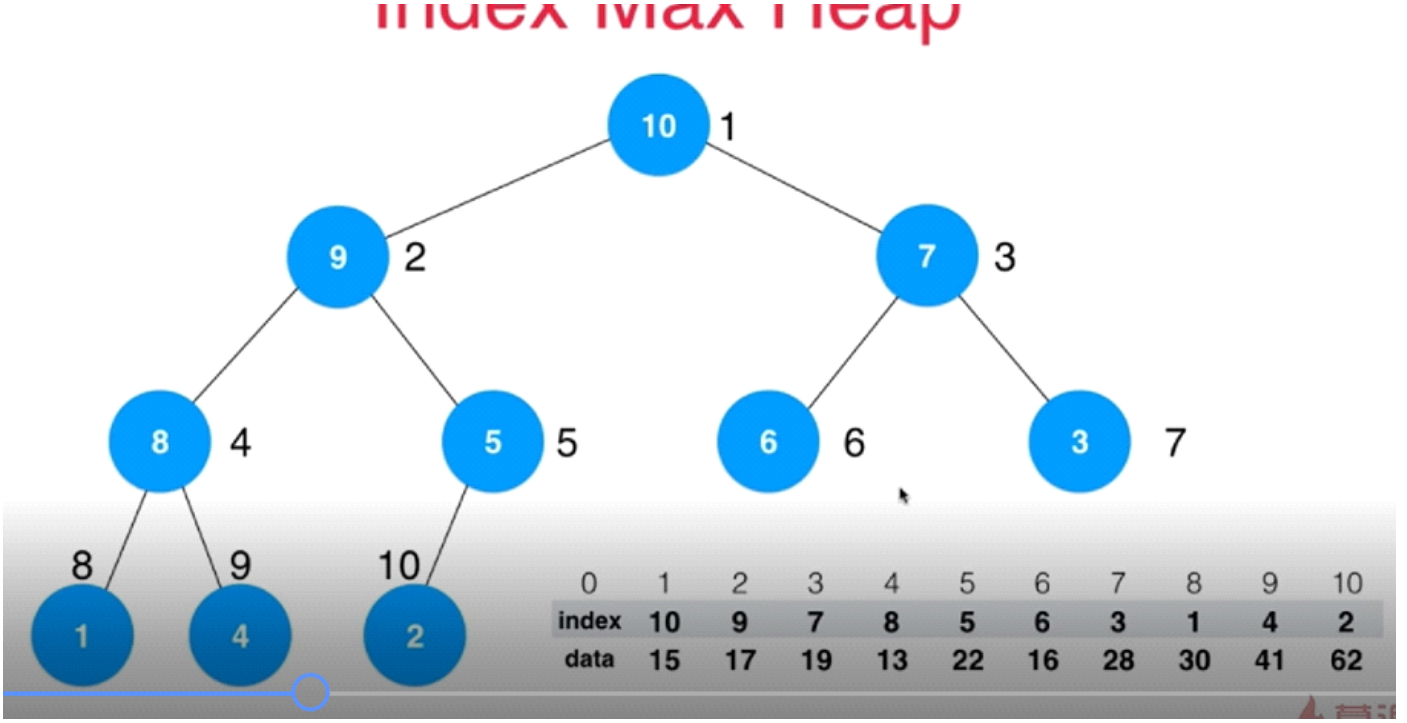
索引堆（Index Heap）

2018年1月2日　　21:55

索引堆：



将数据和索引分开存储，而真正表示数组内容的是他们的索引位置。



对于data域中堆中的元素实际并没有改变，改变的是Index的内容。例如，在上面的表格中，堆顶的元素索引为10，说明堆顶存放的的是10这个位置存放的元素即62，同样的堆的左孩子存放的是9这个位置存放的元素是41，右孩子存放的是7这个位置存放的28.

2018年2月10日 20:23

1.实习简历
（1）介绍自己
（2）突出自己的优点
2.所有相关的基础知识
面试官一般是之后实习的同事或者是之后领导的组长
3.刷题
算法题一般考比较基础的知识（归并、快排）
4.员工内推不筛选简历，除非岗位特别不合适
5.阿里的内推和不内推的区别要式部门来看，实习生内推是没有笔试的，但是面试是一定要的
6.内推挂了，评价是会留在系统里的，内推之后是不能和正常网申同时进行的
7.内推的小技巧：
内推和校招是不一样，内推一般是认识的已经工作的学长进行推荐的，一般是学长的leader进行推荐，那一般是和学长在同一个组里面工作的
8.进BAT的话面试的算法题是不会很难的，算法题最好是把题目的思路记下来
9.校招的话一般早一点投递简历，一般一个星期就能收到通知的

二.银行求职需要做哪些准备
四大行
工行的直属机构：软开，数据等
一般分为三大机构：一般9月中旬总行开始，
比较适合自己的是总行直属机构，包括数据、软开、利润中心（上海）、牡丹卡中心
机构三：各省级分行，分行本部、直属中心、所辖支行
五六月份一般银行实习，实习留用的机会是比较大的
三、看面经
（1）注重时效性
（2）代入感强一点，要自己情景重现，假设自己就在现场
（3）