package sortdemo;  
  
public class DuiPaiXu {  
  
 public static void main(String[] args) {  
  
 //小顶堆 所有的父都小于左右两个儿子,但儿子关系不限制  
  
 //arr[i] < arr[2i+1] && arr[i] < arr[2i+2]  
  
  
 //大顶堆 所有父都大于左右两个儿子,但儿子大小不限制  
  
 //arr[i] > arr[2i+1] && arr[i] > arr[2i+2]  
  
 //堆排序是不稳定的，因为他不是相邻比较的，而是分成树那样了，左边大树和右边大树前后相邻的在树里面可能都没关系,  
 //堆排序的时间复杂度稳定为n\*logn 第一个n是因为要挨个获取n个数，后面的n是获取一个n在树结构中要对比logn次，建  
 //堆得时间是n，但是比小太多，忽略了。  
 int[] chi = { 2, 44, 5, 65, 7, 12, 53, 19, 9, 10, 22, 4 };  
 int len = chi.length;  
 // 将这个数组搞成堆的样子 注意这里是个循环.  
  
  
 //对于堆而下,堆是完全二叉树,也就是上面是满的,如果最后一行不是满的话,也会把子叶节点靠在左边  
 //但是堆本质上还是个数组,那么下面这个是从最左下非子叶节点开始调整的,最左下非子叶节点就是  
 //i=len/2-1这个索引的点哦,这个节点往下的就是非子叶节点,其实也只有非子叶节点才需要调整  
 for(int i=len / 2 - 1;i>=0;i--){  
 //从第一个非叶子结点从下至上，从右至左调整结构  
 gaoChengDui(chi,i,len);  
 }  
   
 while (len > 0) {  
 System.***out***.println(chi[0]);  
  
 //还有一点要注意,对于小顶堆来说 0索引在的地方就是最小的顶点  
 //len-1的地方就是最后下面最靠右面的也就是最后一个  
  
  
 chi[0] = chi[len - 1];  
 gaoChengDui(chi, 0, len);  
 len--;  
 }  
 }  
 //下面这个算法是假设下面的是堆属性的，只有顶点的不是的时候才能用，所以我们在第一次创建堆的时候  
 //需要从下到上，从左到右把非叶节点全部走一遍才可以形成堆，而当调整 的时候由于只有顶点刚被换成最大的，只有顶点是  
 //非堆结构的，所以可以直接gaoChengDui(chi, 0, len);  
 public static void gaoChengDui(int[] chi, int i, int length) {// 由于在后来交换的时候要不停的缩小数组length  
 // 要减小，所以传了进来。  
 int temp = chi[i];  
 int k = 0;  
  
  
 //注意k = 2k +1 说明这个是循环的下一个节点是往下去的!!!!!!!!!!!!!!!!!!!  
  
  
  
  
 for (k = 2 \* i + 1; k < length; k = 2 \* k + 1) {  
  
 //先从  
 if ((k + 1) < length && chi[k] > chi[k + 1]) {  
 k++;// 注意这里要先比  
 }  
  
  
 //这个就是再一个父子三人节点组如果temp是最小的了,那么就让他在这里当爹就可以了  
 //如果temp不是最小的,就一直往下找,有没有小的,  
 //所有最后有一个chi[i]=temp  
  
 if (chi[k] > temp) {// 因为这里是和temp比，所以可以到最后chi[i] = temp;  
 break;  
 } else {  
 // 因为父节点和k换了，所以就有可能影响k节点的堆结构，所以要对k节点判断下，所以k=2\*k+1循环判断。  
 chi[i] = chi[k];  
 i = k;// 现在的k就是后来的i，因为现在由于k节点被换了，要对k节点的情况进行判断。  
 }  
 }  
  
 chi[i] = temp;  
 }  
  
}