堆

堆是一个完全二叉树结构（包括满二叉树和从左到右变满的二叉树）

二叉树中，对于`i`节点，左子孩子为`2*i+1`，右子孩子为`2*i+2`父节点为`(i-1)/2`用于后续的找

大根堆：完全二叉树里，每一棵子树的最大值是头结点的值

小根堆：每一棵子树的最小值是头结点的值

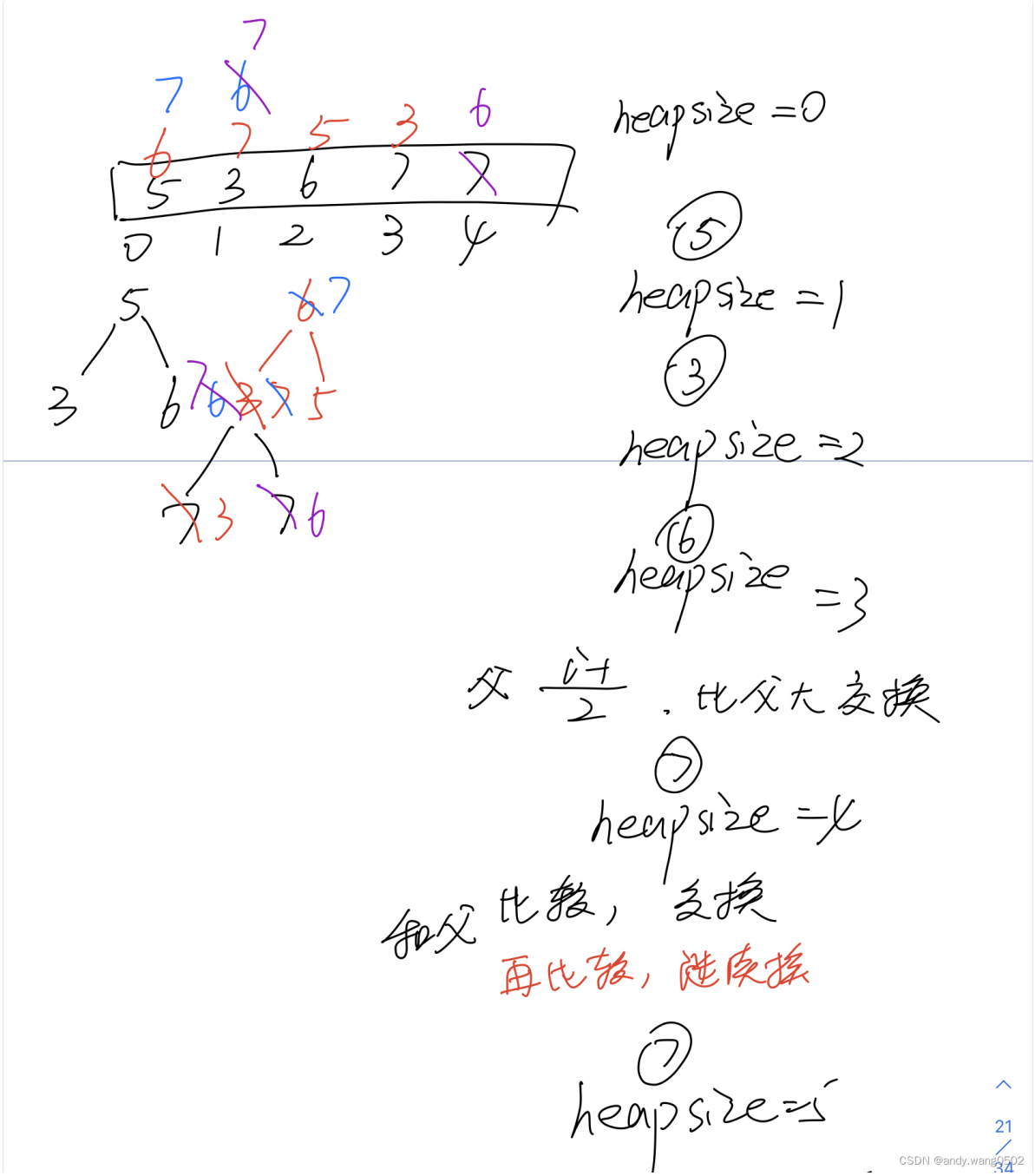
`heapInsert`

给定一组数，找出最大值，大根堆的0位置的数；去掉最大值，先设定一个临时变量存最大值，然后把以及形成的堆结构的最后一个位置的数放到0位置上，`heapsize-1`，然后从头结点开始，与其子节点的最大值进行比较交换，若还有孩子，则继续比较，直到无孩子或不大于则停止

- 首先有数组，然后`heapSize`一开始是 0, 这个就是数组中现在几个数是我现在的堆
- 先接收第一个数字 (注意这样就是一个一个传的，而不是整个直接给你让你变成大根堆)
- 把那个数字放到我们`heapSize`现在的值的下标位置上，`heapSize++`, 现在我们数组中有一个数是我们的大根堆
- 接着把下一个数字放到`heapSize`现在的值的下标位置上，`heapSize++`

  现在看这个放的数字的位置会是第一个的子孩子，所以要看现在还是不是大根堆，也就是当前数字有没有他父亲大 (他可以靠`(i-1)/2`下标算出他父亲的下标)

  - 如果没有父亲大或者等于，那就直接下一个
  - 如果有父亲大，那就 bubble up (注意要是比当前父亲大，然后互换了，需要继续比较看换了后比那个时候的父亲是不是更大，如果还是更大，那就需要继续换，这个就是 bubble up 或者 heap insert)
- 把一个一个传进来的数字都这么操作放入数组中，然后当结束时`heapSize==nums.length`
- 我们已经有了一个大根堆结构




```
public static void heapInsert(int[] arr, int index) {
    while(arr[index] > arr[(index-1)/2]) {//与父节点比较，0位置也不成立
        swap(arr, index, (index-1)/2);
        index = (index-1)/2;
    }

}
```

heapify

- 有了一个大根堆我们就可以从数组中取出第一个元素就是最大的元素，然后让剩下的进行 `heapify` 操作，再次形成大根堆，然后再取出第一个元素也就是我们所有的数据的第二个大的，so on...
- 不过我们也可以直接把第一个元素也就是最大的直接跟最后一个元素交换， `heapSize--`(此时 `heapSize=nums.length-1`, 没错就是这，不过我们知道我们接下来就不用管这最后一个元素了，因为已经在正确的位置上了)

- 此时我们第一个元素是当初最后一个元素，我们可以使用 `heapify` 也就是 bubble down, 跟他的左孩子和右孩子 (都是使用数组下标可以算出来的) 之间更大 的那个，如果比我们当前元素更小，那么交换，**注意左孩子或者 == 左孩子和右孩子 (因为堆结构就是往左到右 fill 的，想想也合理，你数组如果有左孩子和右孩子，那么右孩子的下标就是左孩子的下一个，怎么可能会有 左孩子有右孩子没有 的情况)== 可能会是 null!!!** 如果是 null 那就不换呗
- 我们就一直比，直到我们的元素比他此时的 左孩子和右孩子之间更大 的那个更大或者一样的话，那就不换了，到那就行了
- 此时我们就又形成了堆结构，接着把第一个元素也就是数组所有数第二大的放到 `heapSize` 现在的值 - 1 的下标位置，然后 `heapSize--`, so on...
- 直到我们的 `heapSize` 变成 0 代表数组所有的数我们都排序好了

注意在每次我们的把一个数放到了正确的地方后，我们的 `heapSize` 就成为了那个数的下标的位置，我们下次 `heapify` 操作只需要针对数组下标为 0 到 `heapSize-1` 位置就行了

```
public static void heapify(int[] arr, int index, int heapSize){
    int left = index * 2 + 1;//左孩子的下标
    while(left < heapSize){//下方还有孩子时
     //两个孩子中，谁的值大，把下表给largest
        int largest = left + 1 < heapSize && arr[left + 1] > arr[left] ? left +
1 : left;
     //父和孩子之间，谁的值大，把下标给largest
        largest = arr[largest] > arr[index] ? largest : index;
        if(largest == index){
            break;
        }
        swap(arr, largest, index);
        index = largest;
        left = index * 2 + 1;
    }
}
```

如果我们已经形成了一个大根堆，结果有人突然把大根堆里面的一个下标的数改了，我们再怎么形成堆?

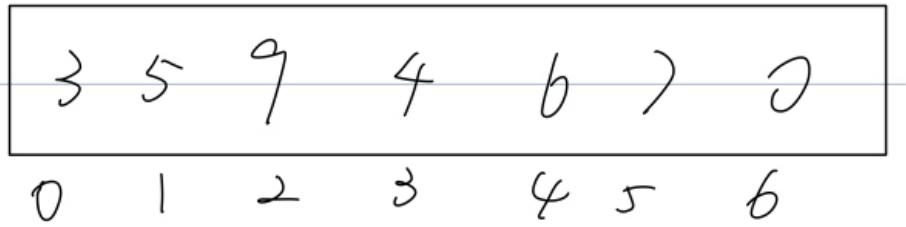
- 我们可以**对这个下标的元素**调用一次 `heapify`, 调用一次 heap insert
- 这样如果这个数改小了，那么就会按照 `heapify` 到合适自己的位置上去，heap insert 不会起效果
- 这样如果这个数改大了，那么就会按照 heap insert 到合适自己的位置上去，`heapify` 不会起效果

完全二叉树的高度 `O(logN)`,时间复杂度 `O(NlogN)`,空间复杂度 `O(1)`

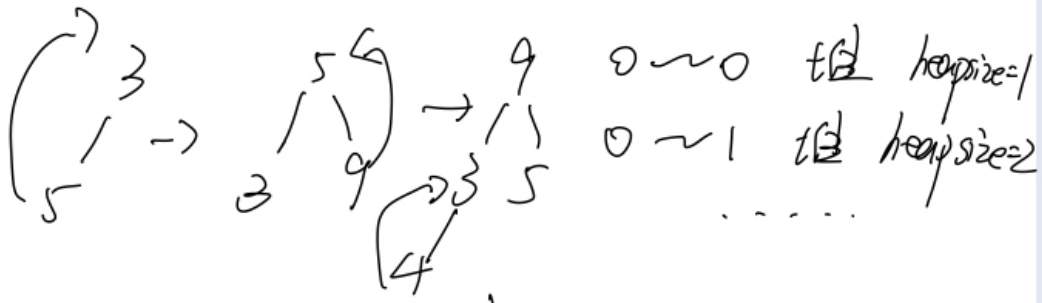
`heapInsert` 和 `heapify` 级别的调整代价

堆排序 `heapSort`

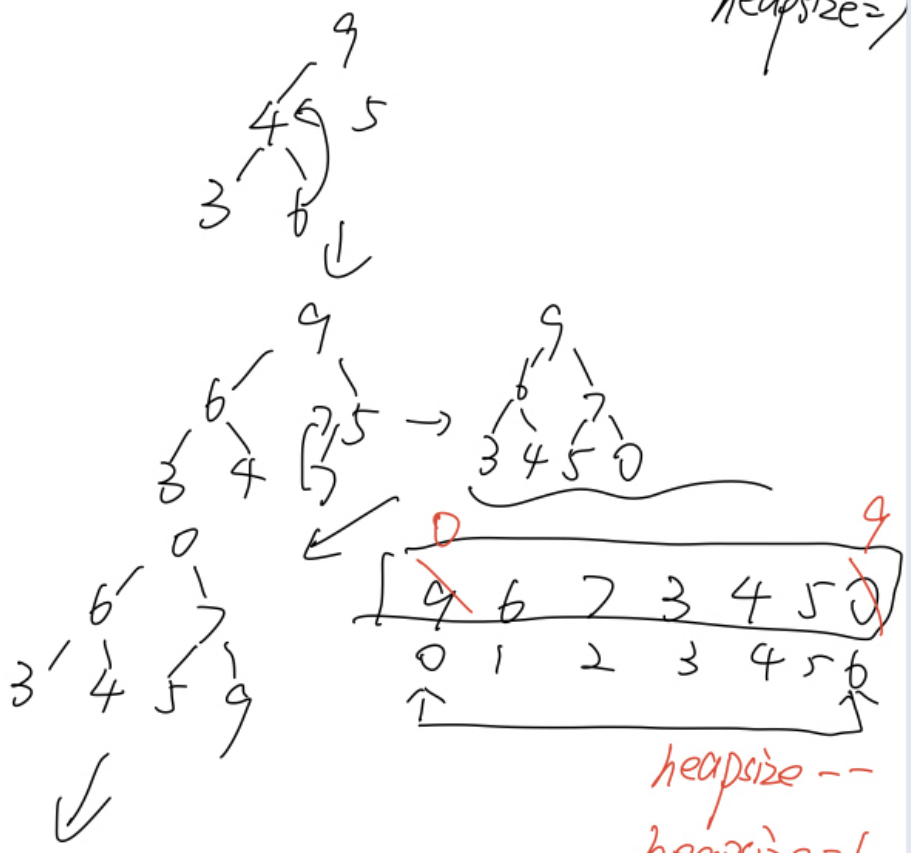
堆排序

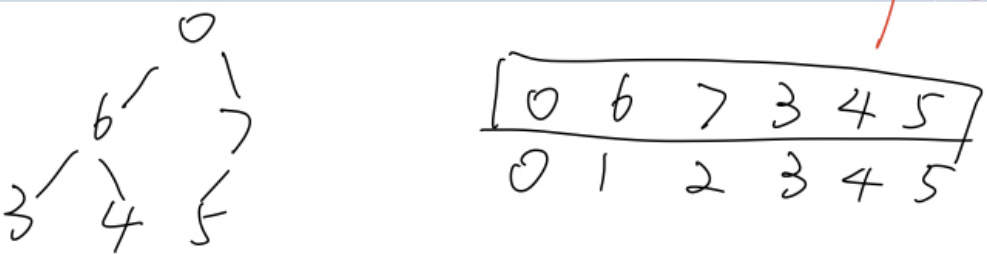


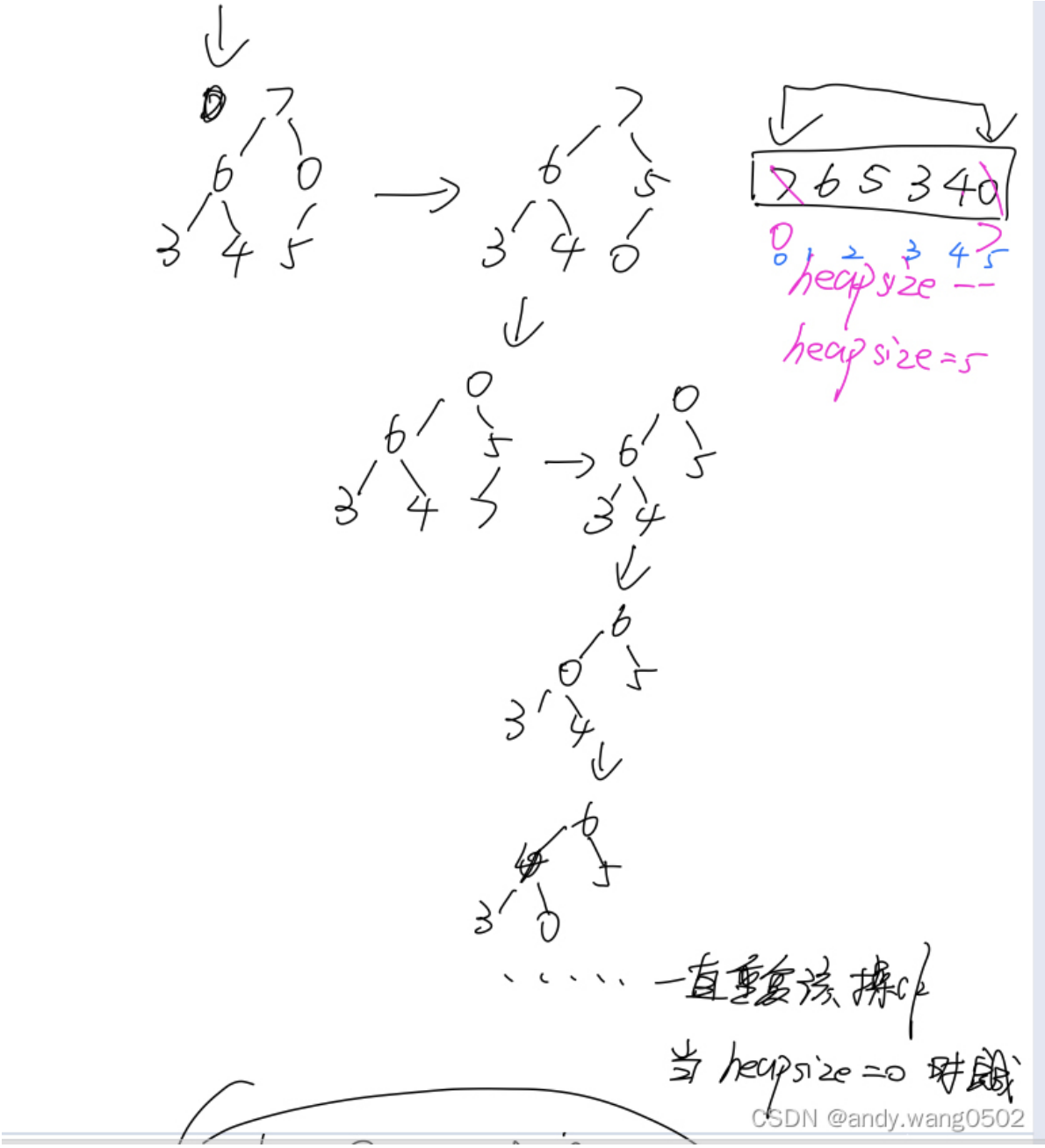
heapsize=0



heapsize=7





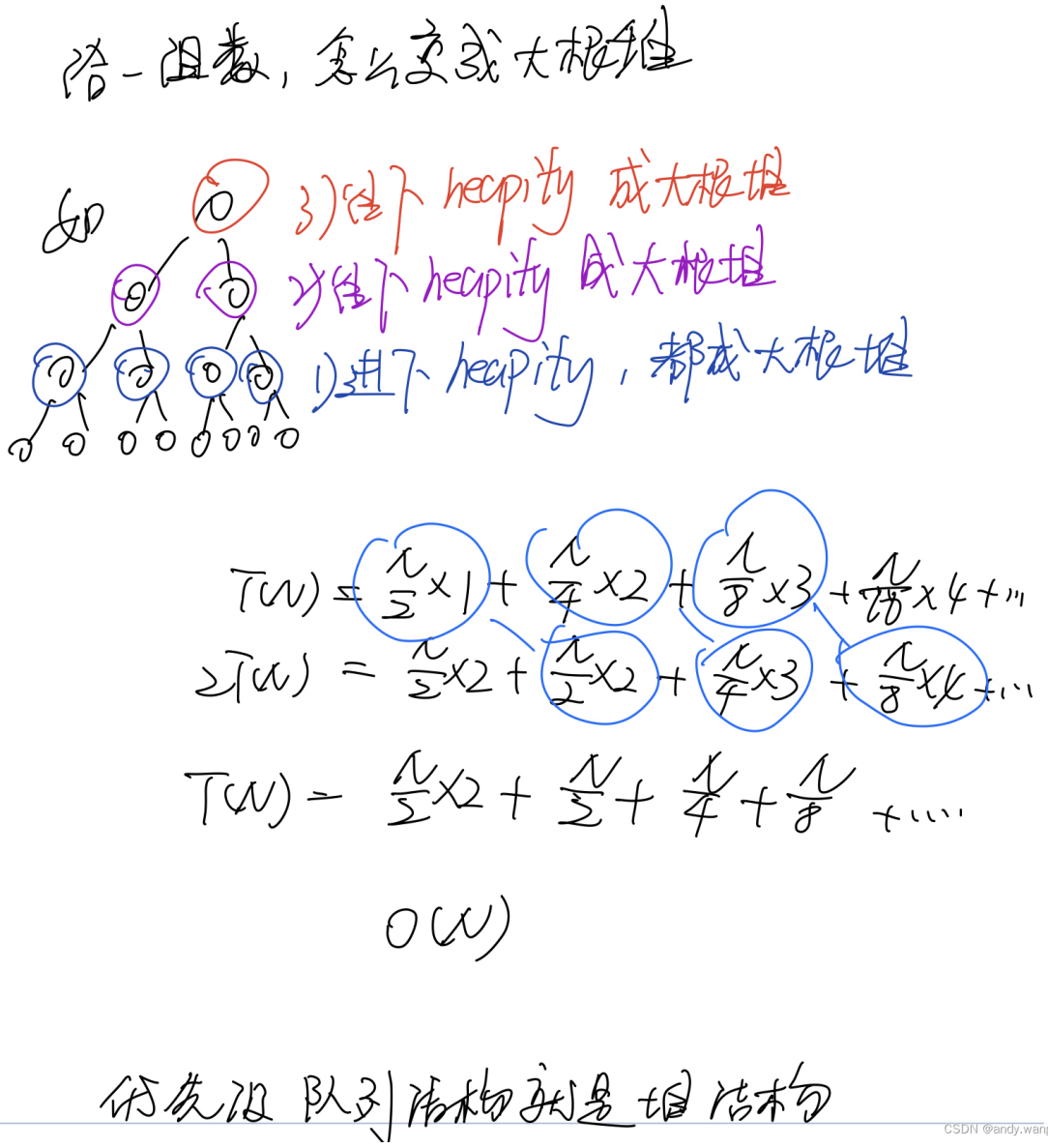


代码：

```
public static void heapSort(int[] arr){
    if(arr == null || arr.length < 2){
        return ;
    }
    for (int i = 0; i< arr.length; i++){
        heapInsert(arr,i);
    }
    int heapSize = arr.length;
    swap(arr, 0, --heapSize);
    while(heapSize > 0){
        heapify(arr, 0 ,heapSize);
        swap(arr, 0 ,--heapSize);
    }//把最大值放在刚失效的位置
}
```

堆排序的时间复杂度 空间复杂度O(1)

问题：给一组数，怎么变成大根堆

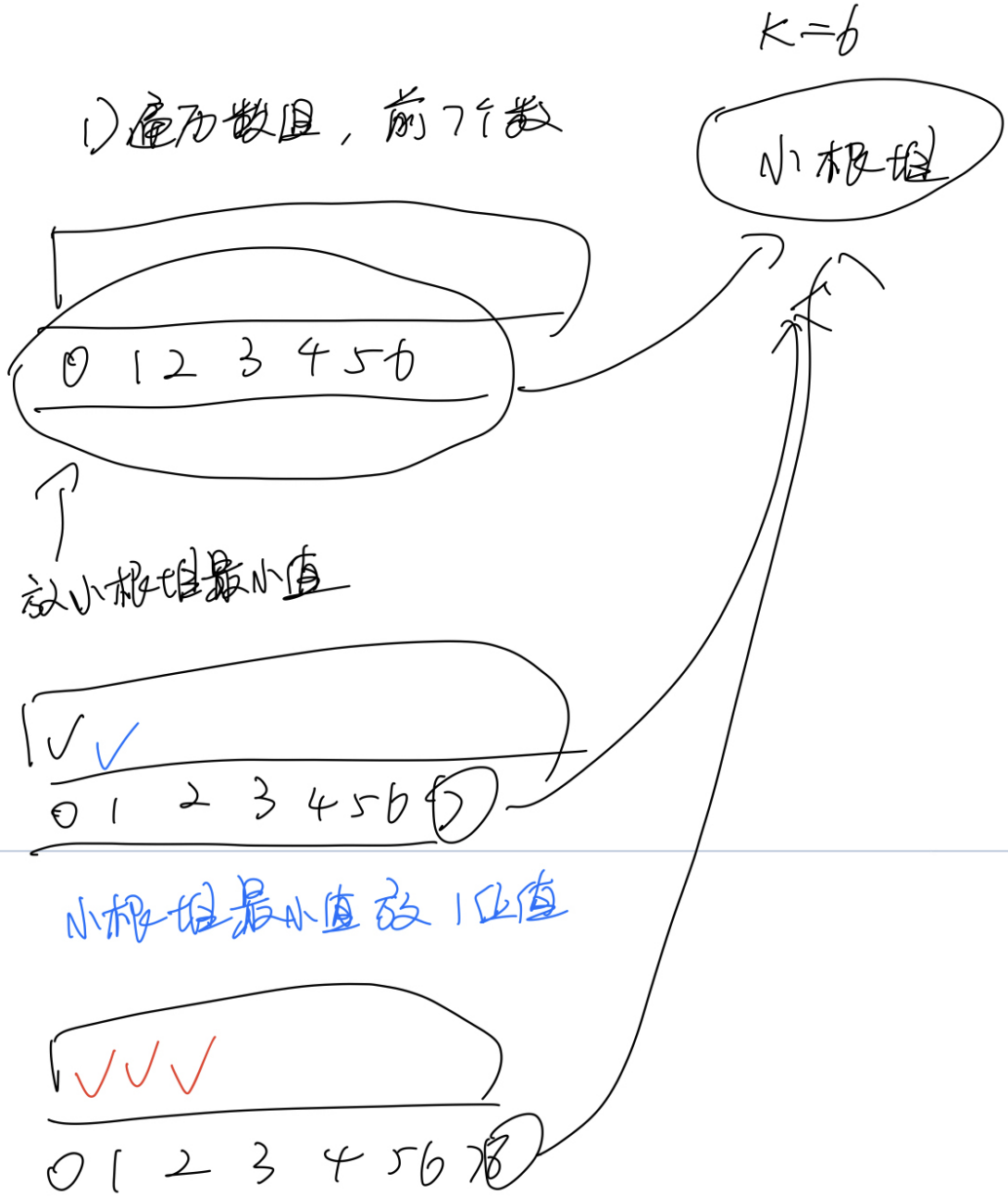



```java
public static void heapSort(int[] arr){
    if(arr == null || arr.length < 2){
        return ;
    }

    for(int i = arr.length - 1; i>= 0; i--){
        heapify(arr, i ,arr.length);
    }

    int heapSize = arr.length;
    swap(arr, 0 --heapSize);
    while(heapSize > 0){
        heapify(arr, 0 ,heapSize);
        swap(arr, 0 ,--heapSize);
    }//把最大值放在刚失效的位置

}
```

堆排序扩展题目

相扩序扩展题目

几乎有序的数组，几乎有序是指，如果把数组排好顺序的话，每个元素移动的距离可以不超过k，并且k相对于数组来说比较小，请选择一个合适的排序算法对这个数据进行排序



```
    public void sortedArrDistanceLessK(int[] arr, int k) {
        PriorityQueue<Integer> heap = new PriorityQueue<>();
        int index = 0;
        for (; index < Math.min(arr.length, k); index++) {
            heap.add(arr[index]);
        }
        int i = 0;
        for (; index < arr.length; i++, index++) {
            heap.add(arr[index]);
            arr[i] = heap.poll();
        }
        while (!heap.isEmpty()) {
            arr[i++] = heap.poll();
        }
    }
```

堆结构的扩容也是`logN`级别的，不会影响性能。

Java中现存的堆排序是一个黑盒，不支持已形成的堆结构修改其中一个数，再以小代价形成对结构，所以有时候要自己手写堆结构。

比较器的使用

返回负数的时候，第一个参数放前面；正数的时候，第二个在前面；0无所谓

1）实质上是重载比较运算符

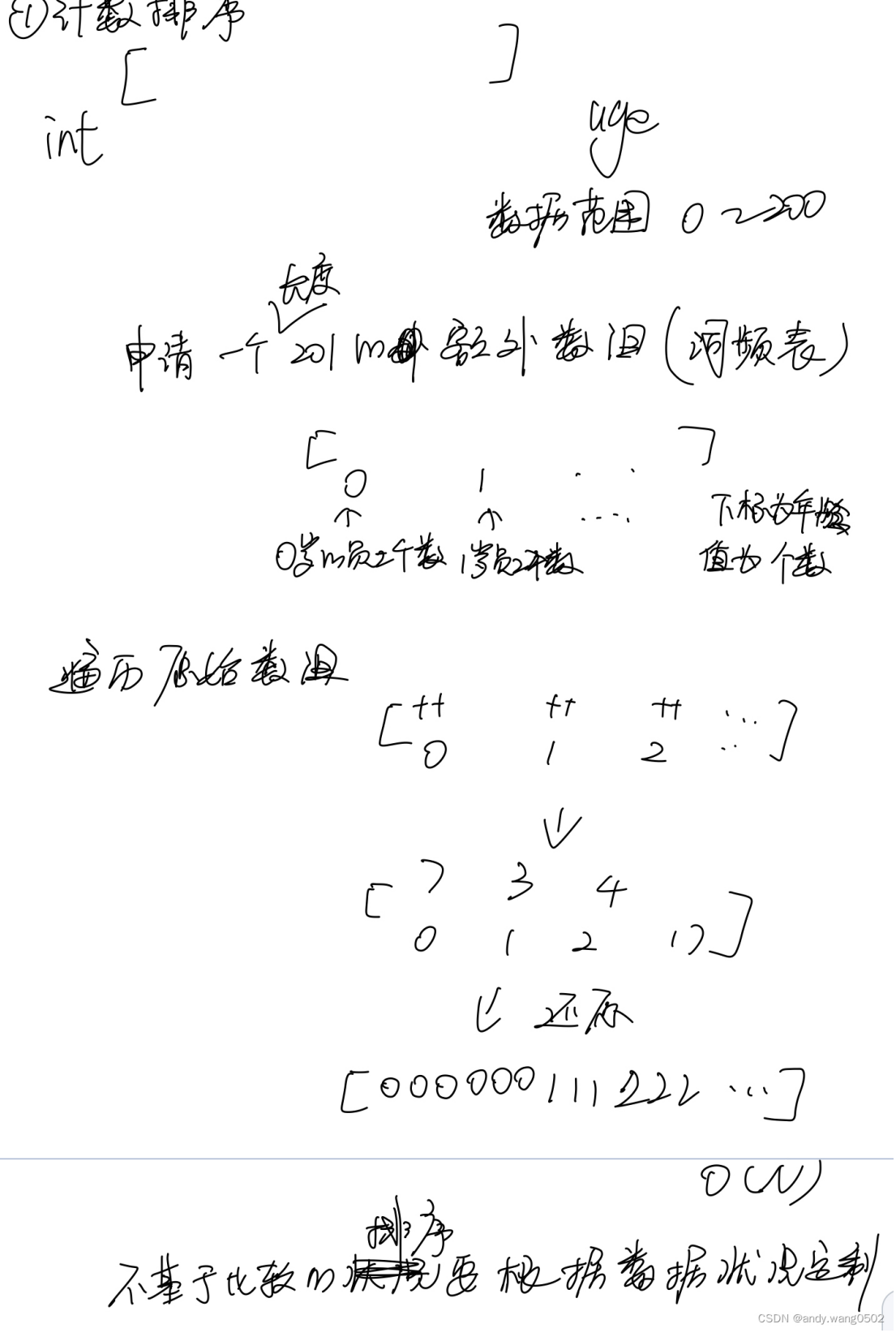
2）比较器可以很好的应用在特殊标准的排序上

3）可以很好的应用在根据特殊标准排序的结构上

比较器可以使用在堆结构里，实现复杂数据的排序。

不基于比较的排序

计数排序



基数排序

② 基数排序

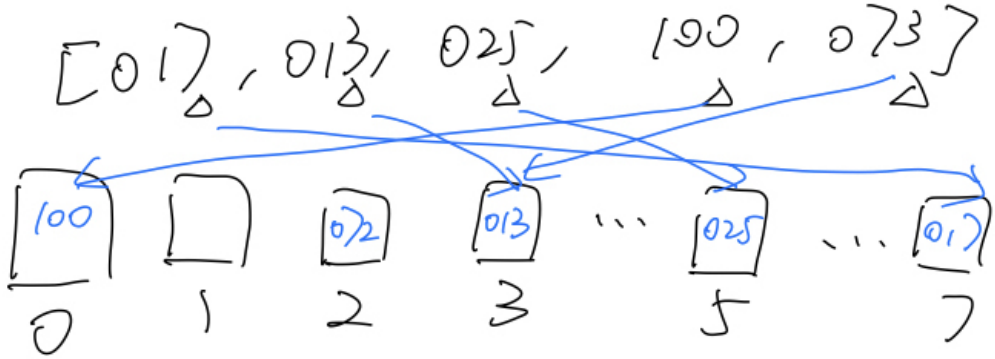
[17, 13, 25, 100, 73]

↓ 先看最大数有几位，补齐0

[017, 013, 025, 100, 073]

↓ 准备桶 10个

从左往右 入桶，先从个位数开始



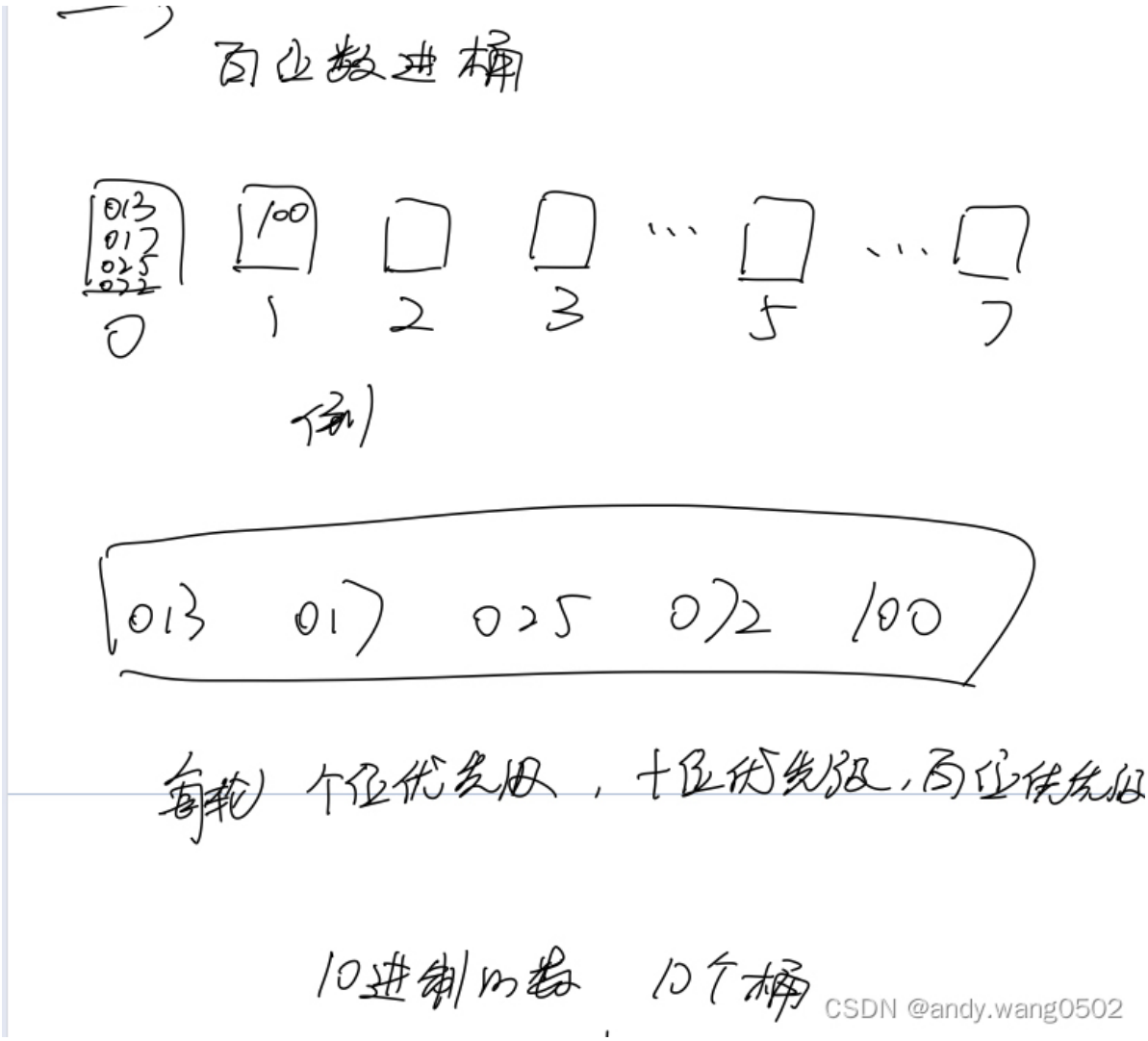
把桶从左往右倒出来 （先进，先出）

按照个位数进桶

[100, 072, 013, 025, 017]

倒出来

[100, 013, 017, 025, 072]




```
//only for no-negative value
public static void radixSort(int[] arr){
    if (arr == null || arr.length < 2){
        return;
    }
    radixSort(arr, 0, arr.length - 1, maxbits(arr));
}

public static int maxbits(int[] arr){
        int max = Integer.MIN_VALUE;
        for(int i = 0; i < arr.length; i++){
            max = Math.max(max, arr[i]);
    }
        int res = 0;
        while(max != 0){
            res++;
            max /= 10;
    }
    return res;
}

//arr[begin..end]排序
public static void radixSort(int[] arr, int L, int R, int digit){//digit最大值有几
个十进制位
        final int radix = 10;
        int i = 0, j = 0;
        //有多少个数准备多少个辅助空间
```

```
        int[] bucket = new int[R - L + 1];
        for(int d = 1; d <= digit; d++){//有多少位就进出几次
            //10个空间
            //count[0]当前位（d位）是0的数字有多少个
            //count[1]当前位（d位）是0和1的数字有多少个
            //count[2]当前位（d位）是0、1和2的数字有多少个
            //count[i]当前位（d位）是（0~i）的数字有多少个
            int[] count = new int[radix]; // count[0..9]
            for (i = L; i <= R; i++){
                j=getDigit(arr[i], d);
                count[j]++;
            }
            for(i = 1; i < radix; i++){
                count[i] = count[i] + count[i-1];
            }
            for(i = R; i >= L; i--){        //倒着看数字放
                j = getDigit(arr[i],d);
                bucket[count[j]-1]=arr[i];
                count[j]--;
            }
            for(i = L, j = 0; i <= R; i++, j--){
                arr[i] = bucket[j];
            }
        }
}

public static int getDigit(int x, int d){
    return (( x / (( int) Math.pow(10, d-1))) % 10);
}
```

————————————————————