##binarySearch

278. First Bad version
279. Guess Number Higher or Lower
280. Heaters
281. H-Index2
282. Find Minimum in Rotated Sorted Array
283. Search in Rotated Sorted Array
284. Search a 2D Matrix
285. Find the Duplicate Number
286. Search in Rotated Sorted Array
287. Search in Rotated Sorted Array II（33 follow up-with duplicates）
288. Search a 2D Matrix
289. Search a 2D Matrix II

####notice! binarySearch 常用 start/end，sort问题常用 left/right，two pointers问题常用 fast/slow，linked list + two pointers常用 walker/runner

####278. First Bad version/

注意：和first position of target（两个题二分考点一毛一样），区别在于返回值，version或者说bad version是连续存在的，有3必有2，但target可能不存在，所以version直接返回start，target要考虑不存在的情况下返回-1，其他情况返回start

看题应该会立刻想到二分查找，另外这个思想和和438中sliding window的思路有有一点类似. 这个题需要搞清楚两点：

1. 如果mid不是，那mid前都不是，查找mid后面，如果mid是，查找mid前
2. 返回谁，我是举了个例子试了一下，不过从if这句可以看出，返回start
3. 注意start=1，从1开始

```
while(start<end){
    int mid = start+(end-start)/2;
    //all the versions after a bad version are also bad, 所
```

```
以如果mid没有，mid前肯定都没有，查找后一半
    if(!isBadVersion(mid)) start = mid+1;
    else end = mid;
}
```

####374. Guess Number Higher or Lower 跟上题一样，binary search 代码也几乎一样，==1说明更大，向后找，所以start=mid+1；==-1说明更小，向前找，返回start。可以背个模板了。

####475. Heaters 还是tag：binary search, 看题并没有很好的思路，discuss

```
public int findRadius(int[] houses, int[] heaters) {
    //用binary search前需要排序，不然答案不确定
    Arrays.sort(heaters);
    int result = Integer.MIN_VALUE;
    for (int house : houses) {
        //Arrays.bianrySearch(object[] a, object key)如果它包含在数组中，则返回搜索键的索引；否则返回 (-(插入点) - 1)。插入点被定义为将键插入数组的那一点：即第一个大于此键的元素索引。
        int index = Arrays.binarySearch(heaters, house);
        if (index < 0) {
            index = -(index + 1);
        }
        //如果<0，说明house在最左边heater的左边，dist1=max无意义，实际算的距离是dist2，此时=heaters[index] - house
        int dist1 = index - 1 >= 0 ? house - heaters[index - 1] : Integer.MAX_VALUE;
        int dist2 = index < heaters.length ? heaters[index] - house : Integer.MAX_VALUE;
        result = Math.max(result, Math.min(dist1, dist2));
    }
    return result;
}
```

####275. H-Index2 given: sorted array, require O(lgn)，搞清楚要求的话很容易想到binary search。注意有一种特殊情况：input: [0,1,2,4,5,6]，output: 3，整个while循环都不能输出正确答案，所以需要return输出的帮助

```
while(start<=end){
    int mid = (start+end)/2;
    if(citations[mid]==(citations.length-mid)) return
citations.length-mid;
    else if(citations[mid]<(citations.length-mid)) start =
mid+1;
    else end = mid-1;
}
return citations.length-start;
```

####274. H-Index new一个新array实现类似哈希表的思想，新array的下标对应input的element，新array长度比input长1，输入的数组中，比较大的数字都记在最后一位：

比如input：[0,3，1，6，5] new array:[1，1，0，1，0，2] idx: 0 1 2 3 4 5

之后从后往前计算new array element sum 当 sum>=idx 此时的idx就是我们要找的h-index

```
public int hIndex(int[] citations) {
    int len = citations.length;
    if(len==0) return 0;
    int[] re = new int[len+1];
    for(int i = 0; i<len; i++){
        if(citations[i]>len) re[len]++;
        else re[citations[i]]++;
    }
    int sum = 0;
    for(int i = len; i>0; i--){
        sum+=re[i];
```

```
        if(sum>=i) return i;
    }
    return 0;
}
```

####153. Find Minimum in Rotated Sorted Array binary search写了这么多，这个还是不会写，感觉没抓住要点，有模板也不行.

这个题的要点，应该在if判断那里，discuss里的解析写的挺好的

1. The minimum element must satisfy one of two conditions: 1) If rotate, A[min] < A[min - 1]; 2) If not, A[0].
2. check the middle element, if it is less than previous one, then it is minimum.
3. If not, there are 2 conditions as well: If it is greater than both left and right element, then minimum element should be on its right, otherwise on its left.

```
public int findMin(int[] nums) {
    if(nums==null||nums.length==0) return 0;
    if(nums.length==1) return nums[0];
    int start = 0;
    int end = nums.length-1;
    while(start<end){
        int mid = start+(end-start)/2;
        if(nums[mid]<nums[mid-1]) return nums[mid];
        else
if(nums[mid]>nums[end]&&nums[mid]>nums[start]) start =
mid+1;
        else  end = mid-1;
    }
    return nums[start];
}
```

####287. Find the Duplicate Number(bb) 这个题超智商了，后面两种方法均看不懂，我只能想到这一种，然而不满足题目要求，改变了数组，，gg，不如挑一个背吧┐(￣Д ￣)┌。。。

```
public int findDuplicate(int[] nums) {
        Arrays.sort(nums);
        int missing = nums.length;
        for(int i = 0;i<nums.length-1;i++){
            if ((nums[i+1]-nums[i])==0){
                return nums[i];
            }
        }
        return missing;
    }
```

这题要求是要求是：

You must not modify the array (assume the array is read only).

You must use only constant, O(1) extra space.

Your runtime complexity should be less than O(n2).

There is only one duplicate number in the array, but it could be repeated more than once.

虽然有binary search的tag但我肯定想不到这种binary search的方法。。。而且这个方法复杂度也不够好：O(1) space complexity, O(nlgn) time complexity，跟排序，然后for循环找duplicate的复杂度一样。不过还是放上code,cnt是计数的，通过cnt和mid比较判断重复元素在哪边。不好想也不优，算了。。推荐下面的two pointers的解法

```
public int findDuplicate(int[] nums) {
        int low = 1, high = nums.length - 1;
    while (low <= high) {

```

```
        int mid = (int) (low + (high - low) * 0.5);
        int cnt = 0;
        for (int a : nums) {
            if (a <= mid) ++cnt;
        }
        if (cnt <= mid) low = mid + 1;
        else high = mid - 1;
    }
    return low;
}
```

discuss区一个解法：O(n) time and O(1) space without modifying the array.two pointer

```
public int findDuplicate(int[] nums) {
    if (nums.length > 1)
    {
        int slow = nums[0];
        int fast = nums[nums[0]];
        while (slow != fast)
        {
            slow = nums[slow];
            fast = nums[nums[fast]];
        }
        System.out.println(slow+"-"+fast);
        fast = 0;
        while (fast != slow)
        {
            fast = nums[fast];
            slow = nums[slow];
        }
        return slow;
    }
    return -1;
```

```
}
```

####33. Search in Rotated Sorted Array 这个是有点复杂的二分，不太好想。思路是分两种情况，一种是start-mid是不rotated的，那rotated部分肯定在mid-end，另一种相反。假设sratr-mid是不rotated，我们做正常二分，反之，我们在mid-end部分做二分

```
while(start+1<end){
    int mid = start+(end-start)/2;
    if(nums[mid]==target) return mid;
    if(nums[start]<nums[mid]){
        if(nums[start]<=target&&target<=nums[mid]) end =
mid;
        else start = mid;
    }else{
        if(nums[end]>=target&&target>=nums[mid]) start =
mid;
        else end = mid;
    }
}
if(nums[start]==target) return start;
if(nums[end]==target) return end;
return -1;
```

####81. Search in Rotated Sorted Array II（33 follow up-with duplicates）九章给的这个解释很有道理。。

```
public class Solution {
    // 这个问题在面试中不会让实现完整程序
    // 只需要举出能够最坏情况的数据是 [1,1,1,1... 1] 里有一个0即
可。
    // 在这种情况下是无法使用二分法的，复杂度是O(n)
    // 因此写个for循环最坏也是O(n)，那就写个for循环就好了
    //  如果你觉得，不是每个情况都是最坏情况，你想用二分法解决不是最
```

```
坏情况的情况，那你就写一个二分吧。
    //  反正面试考的不是你在这个题上会不会用二分法。这个题的考点是你想不想得到最坏情况。
    public boolean search(int[] A, int target) {
        for (int i = 0; i < A.length; i ++) {
            if (A[i] == target) {
                return true;
            }
        }
        return false;
    }
}
```

####74. Search a 2D Matrix binary search, convers the nth number to matrix[n/col][n%col]. Notice: we should check if(matrix[0]==null||matrix[0].length==0) return false; too.

```
public boolean searchMatrix(int[][] matrix, int target) {
    if(matrix==null||matrix.length==0) return false;
    if(matrix[0]==null||matrix[0].length==0) return false;
    int row = matrix.length;
    int col = matrix[0].length;
    int start = 0, end = row*col-1;
    while(start+1<end){
        int mid = start+(end-start)/2;
        if(matrix[mid/col][mid%col]==target) return true;
        else if(matrix[mid/col][mid%col]>target) end =
mid;
        else start = mid;
    }
    if(matrix[start/col][start%col]==target) return true;
    else if(matrix[end/col][end%col]==target) return true;
    return false;
}
```

####240. Search a 2D Matrix II 这个题的考点在思路上，想做到时间复杂度小于暴力解，就考虑每次不是一个一个排除，而是根据sort后的元素关系一列或一排的排除，此时选取最大值或最小值不合适，因为他们所在列还是行都肯定比最大最小，小/大，所以选择左下点和右上点，以左下点为例（代码是根据左下实现的），如果左下点<target 左下点所在的列都可以不看，因为左下点是此列最大值，此时左数第二列的最下面成为我们新的左下点，还是这样比较，如果==target，当前列和排都不看，如果>target，当前排不看。

以上想清楚了，代码写起来很简单，注意（xy要搞清楚）

```
public boolean searchMatrix(int[][] matrix, int target) {
    if(matrix==null||matrix.length==0) return false;
    if(matrix[0]==null||matrix[0].length==0) return false;
    int row = matrix.length;
    int col = matrix[0].length;

    int x = row-1, y =0;
    while(y<col&&x>=0){
        if(matrix[x][y]==target){
            return true;
        }else if(matrix[x][y]>target){
            x--;
        }else if(matrix[x][y]<target){
            y++;
        }
    }
    return false;
}
```