【第十三课】 二分查找

彩蛋题目：开课吧OJ-262

475. 供暖器

思路：1.给了房子的坐标和供暖期的坐标，要求找出覆盖所有房屋的最小加热半径。
2.首先对数组中的供暖器排序，接着扫描每一个房间，接下来问题就变成了需要在供暖器的数组中找到一个离房子最近的供暖器，在所有的房间中找到了最近的供暖器之后，在所有房子和供暖器之间距离关系中，找到最大的半径值就是题中要求。

```
 /**
 * @param {number[]} houses
 * @param {number[]} heaters
 * @return {number}
 */
// 思考，如何去查找最小路径
// 实际上就是去，查找第一个大于等于X的位置，
// 假设是房子X，供暖器是a1,a2,a3,a4,a5;a4是大于等于x的，那么前面的a3是小于x的
// x是落在a3 和 a4中间的，相当于x落在了查找到位置和前一位的中间值
// 那么我们只要判断，这两个值哪一个是距离x最近的就行了
var findRadius = function(houses,  heaters ) {
    heaters.sort((a, b) => a - b)
     let ans = 0;
    for(const x of houses){//遍历每一个房子的位置
        let j = binarySearch(heaters,x);//j是第一个大于等于房子位置的，这个二分模型0,
1，前面是小于x的，后面的大于等于x的，找到大于等于x的
        let a = Math.abs(heaters[j] - x);
        let b = Math.abs(j ? x - heaters[j - 1] : a + 1);//j 位置前面有多少元素
        ans = Math.max(ans,Math.min(a,b));//当前x的房子的供暖器的半径最近就是a,b之间的
较小值
    }
    return ans;


};
var binarySearch = function(nums,x){
    let head = 0, tail = nums.length - 1,mid;
    while(head < tail){//当头尾坐标不重合
        mid = (head + tail) >> 1;
        if(nums[mid] >= x) tail = mid;//
        else head = mid + 1;
    }
    return head;
}
// 理解这个程序，有可能返回的是大于等于x的，有可能返回的是数组的最后一个
// 那么这个写法呢，就会在合法范围内，尽量返回大于等于X的位置，如果不能返回，就返回小于X的位置
```

81. 搜索旋转排序数组 II

思路： 1.一个有序的数组在旋转之后，变成较大值在前面，较小值在后面，最后查找一个元素是否存在.
2.旋转之后的数组，这个时候分两种情况：
(1)两头的值等于X，那么我们就从后往前找小于x的位置，从前往后找大于x的位置，如果两头的值等于x,证明数组中存在待查找值x,直接返回true;
(2)如果两头的值不等于x，分成两层关系；如果 mid <= tail，证明当前中间的位置mid在后半段递增关系中,所以当前确定关系区间是mid - tail；如果 mid > tail,证明当前中间位置mid在前半段递增关系中,所以当前确定关系区间是head - mid。

```
 /**
 * @param {number[]} nums
 * @param {number} target
 * @return {boolean}
 */
var search = function(nums, target) {
    // 首先确定两头的值等不等于target,存在返回true
    if(nums[0] == target || nums[nums.length - 1] == target) return true;
    // 如果不等于的话，就把两头相等的值给他去掉,设置左右区间
    let mid, l = 0,r = nums.length - 1,head,tail;
    // 因为去掉两头相等的值，保证待查找区间的头的值大于尾部的值，二分到中间值的时候 方便做判断
    while(l < nums.length && nums[l] == nums[0]) ++l;
    while( r >= 0 && nums[r] == nums[0]) --r;
    head = l, tail = r;
    while(l <= r){//证明当前待查找区间还有数
        mid = (l + r) >> 1;
        if(nums[mid] == target) return true;//首先第一层判断，中间值在前半段有序区间还是在后半段的有序区间中
        if(nums[mid] <= nums[tail]){//在后半段的有序区间中
            if(target > nums[mid] && target <= nums[tail]) l = mid + 1;
            else r = mid - 1;
        }else{//在前半段的有序区间中
            if(target < nums[mid] && target >= nums[head]) r = mid - 1;
            else l = mid + 1;
        }
    }
    return false;
};
```

4. 寻找两个正序数组的中位数

思路：1.两个有序数组合并之后，求中位数。2.在nums1数组取二分之k，在nums2取二分之k,如果前者小于后者，证明中位数在nums1的后半部分和nums2的前半部分。

```
/**
 *
 *
 * @param {number[]} nums1
 * @param {number[]} nums2
 * @return {number}
 */
var findMedianSortedArrays = function(nums1, nums2) {
    // nums1长度比nums2小
    if (nums1.length > nums2.length) {
        [nums1, nums2] = [nums2, nums1];
    }
```

```
    let m = nums1.length;
    let n = nums2.length;
    // 在0～m中查找
    let left = 0;
    let right = m;

    // median1：前一部分的最大值
    // median2：后一部分的最小值
    let median1 = 0;
    let median2 = 0;

    while(left <= right) {
        // 前一部分包含 nums1[0 .. i-1] 和 nums2[0 .. j-1]
        // 后一部分包含 nums1[i .. m-1] 和 nums2[j .. n-1]
        const i = left + Math.floor((right - left) / 2);
        const j = Math.floor((m + n + 1) / 2) - i;

        const maxLeft1 = i === 0 ? -Infinity : nums1[i - 1];
        const minRight1 = i === m ? Infinity : nums1[i];

        const maxLeft2 = j === 0 ? -Infinity : nums2[j - 1];
        const minRight2 = j === n ? Infinity : nums2[j];

        if (maxLeft1 <= minRight2) {
            median1 = Math.max(maxLeft1, maxLeft2);
            median2 = Math.min(minRight1, minRight2);
            left = i + 1;
        } else{
            right = i - 1;
        }
    }
    return (m + n) % 2 == 0 ? (median1 + median2) / 2 : median1;
};
```

300. 最长递增子序列

思路：1.我们维护一个数组 array[i]，表示长度为 i的最长上升子序列的末尾元素的最小值，用 len 记录目前最长上升子序列的长度，起始时 len为 1，array[1] =nums[0].

2.我们依次遍历数组 nums 中的每个元素，并更新数组 d 和 len 的值。如果 nums[i] > array[len] 则更新 len = len + 1，否则在 array[1...len]中找满足 array[i - 1] < nums[j] < array[i] 的下标 i，并更新 array[i] = nums[j]。

```
/**
 * @param {number[]} nums
 * @return {number}
 */
var lengthOfLIS = function(nums) {
    let head = 1, tail = nums.length
    if (tail === 0) return 0
    let d = []
    d[head] = nums[0]
    for (let i = 1; i < tail; ++i) {
```

```
        if (nums[i] > d[head]) {
            d[++head] = nums[i];
        } else {
            let l = 1, r = head, flag = 0; // 如果找不到说明所有的数都比 nums[i] 大，此时要更新 d[1]，所以这里将 flag 设为 0
            while (l <= r) {
                let mid = (l + r) >> 1;
                if (d[mid] < nums[i]) {
                    flag = mid;
                    l = mid + 1;
                } else {
                    r = mid - 1;
                }
            }
            d[flag + 1] = nums[i];
        }
    }
    return head;
};
```

1011. 在 D 天内送达包裹的能力

思路：

1.我们要确定二分的范围，由于不存在包裹拆分的情况，考虑如下两种边界情况：

2.理论最低运力：只确保所有包裹能够被运送，自然也包括重量最大的包裹，此时理论最低运力为 max，max 为数组 weights 中的最大值。

3.理论最高运力：使得所有包裹在最短时间（一天）内运送完成，此时理论最高运力为 sum，sum 为数组 weights 的总和。

```
/**
 * @param {number[]} weights
 * @param {number} days
 * @return {number}
 */
var shipWithinDays = function(weights, days) {
    let l = 0,r = 0, mid;
    for(const x of weights) r += x,l = Math.max(l,x);
    while(l < r){
        mid = (l+r) >> 1;
        if(check(weights,mid) <= days) r = mid;
        else l = mid + 1;
    }
    return l;
};
var check = function(nums,x){
    let cnt = 1,sum = 0;
    for(const y of nums){
        if(sum + y > x){
            cnt += 1;
            sum = y;
        }else{
            sum += y;
```

```
            }
        }
        return cnt;
    }
```