【第十三课】 二分查找

69. x 的平方根

思路：咱们将前面小于等于x的记为1，大于x的记为0；所以就把平方求解的问题抽象成，前面一堆1后面一堆0的二分查找，最后求最后一个1的位置 。

```
 /**
 * @param {number} x
 * @return {number}
 */
var mySqrt = function(x) {
    // 因为要找最后一个1的位置，假设head指向代表待查找的区间的第一个值，
    // 求 y = 根号x;y最小是0，最大为x
    let head = 0,tail = x,mid;
    tail += 1;
    for(let i = 0; i < 100;i++){
        // 避免计算超界
        mid  = head + ((tail -head) /2.0);
        if(mid * mid <= x) head = mid;
        else tail = mid;
    }
    return Math.floor(head);
};
// 骚操作的处理：
// 调整固定次数，为什么调整100次就能找到；
// 二分调整一次，就把带查找的区间的缩少一半
// 二分调整100次，待调整区间就是2的100次方分之一；
// 此时，head和tail就已经挨得很近很近了，所以head和tail之间的误差是可能仅有2的100次方分之一
// 可以认为两个是一个值，最后保留head的整数部分
```

35. 搜索插入位置

思路：搜索第一个大于或等于的X的元素位置，这个二分就是前面一堆0，后面一堆1，查找第一个1的这种二分模型。

```
 /**
 * @param {number[]} nums
 * @param {number} target
 * @return {number}
 */
var searchInsert = function(nums, target) {
    let head = 0,tail = nums.length,mid;
    while(head < tail){
        mid = (head + tail) >> 1;
        if(nums[mid] < target) head = mid + 1;
        else tail = mid;
    }
    return head;
};
```

1. 两数之和

思路我们遍历到数字 a 时，用 target 减去 a，就会得到 b，若 b 存在于哈希表中，我们就可以直接返回结果了。若 b 不存在，那么我们需要将 a 存入哈希表，好让后续遍历的数字使用。

```
/**
 * @param {number[]} nums
 * @param {number} target
 * @return {number[]}
 */
var twoSum = function(nums, target) {
    let map = new Map();
    for(let i =0;i<nums.length;i++){
        if(map.has(target - nums[i])){
            return [map.get(target - nums[i]),i];
        }else{
            map.set(nums[i],i);
        }
    }
    return [];
};
```

34. 在排序数组中查找元素的第一个和最后一个位置

思路：// 思路：查找两个位置，都是前面都是0，后面都是1的。找到第一个>=x和第一个>x的位置的前一位。第一个：>=x 0000 1111111 第二个：>x 0000 11111111 。

```
/**
 * @param {number[]} nums
 * @param {number} target
 * @return {number[]}
 */
var binarySearch = function(nums,x) {
    let head = 0,tail = nums.length-1,mid;
    while(tail - head > 3){
        mid = (head + tail) >> 1;
        if(nums[mid] >= x) tail = mid;
        else head = mid + 1;
    }
    // [1,2,3,4,5] x = 9
    for(let i = head;i <= tail;i++){
        if(nums[i] >= x) return i;
    }
    return nums.length;
}
var searchRange = function(nums, target) {
    let ret = new Array(2);
    ret[0] = binarySearch(nums,target);
    if(ret[0] == nums.length || nums[ret[0]] != target){
        ret[0] = ret[1] = -1;
        return ret;
```

```
    }
    ret[1] = binarySearch(nums,target + 1) - 1;//最后一个位置
    return ret;
};
```

1658. 将 x 减到 0 的最小操作数

思路： 1、要求减去左边的一些值，减去右边的一些值，然后我们最少经过多少次操作，才能将X 减到0。

2、用到了前缀和；前缀和就是这个数组i元素前面的所有的元素相加之和。

3、前缀和就是用来解决，区间和，就是从左边数前缀和，从右边数前缀和。又因为数组中没有负数，所以前缀和一定是个有序数组，所以处理出两个前缀和、后缀和数组，sumLeft 、sumRight这两个数组肯定都是单调的，一定是个有序数据，递增的。

4、遍历是左边的前缀和里面的每个值，再到右边前缀和里面的值去查找，查找剩余部分是否存在

```
/**
 * @param {number[]} nums
 * @param {number} x
 * @return {number}
 */
var minOperations = function(nums, x) {
    let sumL = new Array(nums.length + 1);
    let sumR = new Array(nums.length + 1);
    sumL[0] = sumR[0] = 0;
    for(let i = 0;i < nums.length;i++){
        sumL[i+1] = sumL[i] + nums[i];
    }
    for(let i = nums.length-1;i >= 0;--i){
        sumR[nums.length - i] = sumR[nums.length - i-1]+ nums[i];
    }
    let ans = -1;
    for(let i =0; i < sumL.length;i++){
        let j = binarySearch(sumR,x-sumL[i]);//查找剩余数量
        if(j == -1) continue;
        if(i+j>nums.length) continue;
        if(ans == -1 || ans > i+j) ans = i+j;
    }
    return ans;
};
var binarySearch = function(nums,x){
    let head = 0,tail = nums.length -1,mid;
    while(head <= tail){
        mid = (head +tail) >>1;
        if(nums[mid] == x) return mid;
        if(nums[mid] < x) head = mid +1;
        else tail = mid -1;
    }
    return -1;
}
```

kaikeba
开课吧