剑指53 在排序数组中查找数字1

1.简单二分法后循环查询。O(logN) + O(m) // m为目标的出现个数
2.两次二分法寻找左右边界。 O(logN)

我的方法二 中O(m)其实趋近O(N)，效率不佳。

```cpp
// 我的方法：二分寻找到一个target，然后左右循环计算个数。
class Solution {
public:
    int search(vector<int>& nums, int target) {
        int rp = nums.size() - 1;
        int lp = 0;
        while(lp <= rp) {
            int index = (rp + lp)/2;
            if (nums[index] < target) {
                lp = index + 1;
            }
            else if (nums[index] > target) {
                rp = index - 1;
            }
            else {
                int times = 1;
                for (int i = index-1; i >= lp; i--) {
                    if (nums[i] == target)
                        times++;
                    else
                        break;
                }
                for (int i = index+1; i <= rp; i++) {
                    if (nums[i] == target)
                        times++;
                    else
                        break;
                }
                return times;
            }
        }
        return 0;
    }
};
```

```cpp
// k神：用2次二分，分别查找right和left，最终返回right-left-1

// 我的初次尝试：
class Solution {
public:
    int search(vector<int>& nums, int target) {
        int rp = nums.size() - 1;
        int lp = 0;
        while(lp <= rp) {
            int index = (rp + lp)/2;
            if (nums[index] < target) {
                lp = index + 1;
            }
            else if (nums[index] > target) {
                rp = index - 1;
            }
            else {
                rp = index - 1;
            }
        }
        int begin = rp + 1;

        rp = nums.size() - 1;
        lp = 0;
        while(lp <= rp) {
            int index = (rp + lp)/2;
            if (nums[index] < target) {
                lp = index + 1;
            }
            else if (nums[index] > target) {
                rp = index - 1;
            }
            else {
                lp = index + 1;
            }
        }

        int end = lp - 1;
        return end - begin + 1;
    }
};


// k神更优:
class Solution {
    public int search(int[] nums, int target) {
        // 搜索右边界 right
        int i = 0, j = nums.length - 1;
        while(i <= j) {
            int m = (i + j) / 2;
            if(nums[m] <= target) i = m + 1;
            else j = m - 1;
        }
        int right = i;
        // 若数组中无 target ，则提前返回
        if(j >= 0 && nums[j] != target) return 0;
        // 搜索左边界 right
        i = 0; j = nums.length - 1;
        while(i <= j) {
            int m = (i + j) / 2;
            if(nums[m] < target) i = m + 1;
            else j = m - 1;
        }
        int left = j;
        return right - left - 1;
    }
}

// 应该更深入理解二分法，不局限于找到某元素的应用场景(小，等于，大)3种情况。也可以是找边界的应用场景(大于等于，小于)2种情况，这种情况必能找到边界，冷静模拟，来判断最后求到的边界是哪一情况。
```