

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。 $O(\log N) + O(m)$ // m 为目标的出现个数
- 2.两次二分法寻找左右边界。 $O(\log N)$

我的方法二 中 $O(m)$ 其实趋近 $O(N)$ ，效率不佳。

```
1 // 我的方法：二分寻找到一个target，然后左右循环计算个数。
2 class Solution {
3 public:
4     int search(vector<int>& nums, int target) {
5         int rp = nums.size() - 1;
6         int lp = 0;
7         while(lp <= rp) {
8             int index = (rp + lp)/2;
9             if (nums[index] < target) {
10                 lp = index + 1;
11             }
12             else if (nums[index] > target) {
13                 rp = index - 1;
14             }
15             else {
16                 int times = 1;
17                 for (int i = index-1; i >= lp; i--) {
18                     if (nums[i] == target)
19                         times++;
20                     else
21                         break;
22                 }
23                 for (int i = index+1; i <= rp; i++) {
24                     if (nums[i] == target)
25                         times++;
26                     else
27                         break;
28                 }
29                 return times;
30             }
31         }
32         return 0;
33     }
34 };
```

```
1 // k神：用2次二分，分别查找right和left，最终返回right-left-1
2
3 // 我的初次尝试：
4 class Solution {
5 public:
6     int search(vector<int>& nums, int target) {
7         int rp = nums.size() - 1;
8         int lp = 0;
```

```

9      while(lp <= rp) {
10          int index = (rp + lp)/2;
11          if (nums[index] < target) {
12              lp = index + 1;
13          }
14          else if (nums[index] > target) {
15              rp = index - 1;
16          }
17          else {
18              rp = index - 1;
19          }
20      }
21      int begin = rp + 1;
22
23      rp = nums.size() - 1;
24      lp = 0;
25      while(lp <= rp) {
26          int index = (rp + lp)/2;
27          if (nums[index] < target) {
28              lp = index + 1;
29          }
30          else if (nums[index] > target) {
31              rp = index - 1;
32          }
33          else {
34              lp = index + 1;
35          }
36      }
37
38      int end = lp - 1;
39      return end - begin + 1;
40  }
41 };
42
43
44 // k神更优:
45 class Solution {
46     public int search(int[] nums, int target) {
47         // 搜索右边界 right
48         int i = 0, j = nums.length - 1;
49         while(i <= j) {
50             int m = (i + j) / 2;
51             if(nums[m] <= target) i = m + 1;
52             else j = m - 1;
53         }
54         int right = i;
55         // 若数组中无 target , 则提前返回
56         if(j >= 0 && nums[j] != target) return 0;
57         // 搜索左边界 right
58         i = 0; j = nums.length - 1;
59         while(i <= j) {
60             int m = (i + j) / 2;
61             if(nums[m] < target) i = m + 1;
62             else j = m - 1;
63         }
64         int left = j;

```

```
65         return right - left - 1;
66     }
67 }
```

```
68
```

```
69 // 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。
```