

```

1  /*
2  给定一个按照升序排列的整数数组 nums，和一个目标值 target。找出给定目标值在数组中的开始
   位置和结束位置。
3  你的算法时间复杂度必须是  $O(\log n)$  级别。
4  如果数组中不存在目标值，返回 [-1, -1]。
5  示例 1：
6  输入：nums = [5,7,7,8,8,10]，target = 8
7  输出：[3,4]
8  示例 2：
9  输入：nums = [5,7,7,8,8,10]，target = 6
10 输出：[-1,-1]
11 来源：力扣（LeetCode）
12 链接：https://leetcode-cn.com/problems/find-first-and-last-position-of-
    element-in-sorted-array
13 著作权归领扣网络所有。商业转载请联系官方授权，非商业转载请注明出处。
14 */

```

分析:

- 方法一: 线性扫描法,时间复杂度显然是 $O(n)$,不符合题目要求
- 看到 $O(\log n)$ 的算法时间复杂度要求,首先想到的就是二分法,难点在二分法之后如何通过左右两个边界还符合 $O(\log n)$ 的要求.
- 首先利用二分搜索找到一个 mid 对应位置的值为 $target$,时间复杂度 $O(\log n)$
 - 方法二:从 mid 开始分别向左和向右遍历寻找边界,这样的时间度还是 $O(n)$;
 - 方法三:找到 mid 后分别用递归二分分别找左边界和右边界,这样的时间复杂度为 $O((\log n)^2)$
 - 方法四:分别寻找左边界和右边界,调整逻辑使得 mid 总能求出左边界或右边界,这样的时间复杂度 $O(\log(n))$
 - 在寻找左边界的过程中,只要 $left < right$
 - 若 $nums[mid] \geq target$, $right$ 就调整为 mid
 - 若 $nums[mid] < target$, $left$ 就调整为 $mid + 1$
 - 最后一步 $left + 1 = right$, $mid = (left + right) / 2 = left$,
 - 若 $nums[mid] < target$,则 $left = mid + 1 = right$,跳出后 $nums[left] \geq target$
 - 若 $nums[mid] \geq target$,则 $right = mid = left$,跳出后 $nums[left] \geq target$
 - 所以跳出以后只要判断 $nums[left]$ 是否等于 $target$
 - 寻找右边界的思想与上面类同.
 - 寻找左边界和寻找右边界需要注意 mid 的取整方向.

方法一:C++_线性扫描

```

1  class Solution
2  {
3      public:

```

```

4      vector<int> searchRange(vector<int>& nums, int target)
5      {
6          vector<int> vi(2,-1);
7
8          if(nums.size() < 1)
9          {
10             return vi;
11          }
12
13          if(nums[0]== target)
14          {
15              vi[0] = 0;
16          }
17
18          if(nums[nums.size()-1]== target)
19          {
20              vi[1] = nums.size()-1;
21          }
22
23          for(int i = 1 ; i < nums.size();i++)
24          {
25              if(nums[i]==target && nums[i-1]!=target )
26              {
27                  vi[0] = i;
28              }
29
30              if(nums[i]!=target && nums[i-1]==target )
31              {
32                  vi[1] = i-1;
33                  break;
34              }
35
36          }
37          return vi;
38      }
39  };
40
41  /*
42  https://leetcode-cn.com/submissions/detail/38742435/
43  执行结果:
44  通过
45  显示详情
46  执行用时 :20 ms, 在所有 cpp 提交中击败了10.54% 的用户
47  内存消耗 :10.1 MB, 在所有 cpp 提交中击败了93.37%的用户
48  */

```

方法二:C++_二分查找法+待优化寻找边界

```

1  class Solution
2  {
3      public:
4          vector<int> searchRange(vector<int>& nums, int target)
5          {
6              vector<int> vi(2,-1);
7              int left    =    0          ;
8              int right   =    nums.size()-1  ;

```

```

9      int mid      = 0          ;
10     bool flag    = false      ;
11
12     while(right >= left)
13     {
14         mid = left + (right - left)/2;
15         if(nums[mid] == target)
16         {
17             flag = true;
18             break;
19         }
20         else if(nums[mid] > target)
21         {
22             right = mid - 1;
23         }
24         else
25         {
26             left = mid + 1;
27         }
28     }
29
30     if(flag)
31     {
32         left = mid;
33         right = mid;
34         while(left >=0 && nums[left]==target)
35         {
36             left--;
37         }
38
39
40         while(right < nums.size() && nums[right]==target)
41         {
42             right++;
43         }
44
45         vi[0] = left + 1;
46         vi[1] = right - 1;
47
48
49     }
50     return vi;
51 }
52 };
53
54 /*
55
56 https://leetcode-cn.com/submissions/detail/38737198/
57 执行结果:
58 通过
59 显示详情
60 执行用时 :12 ms, 在所有 cpp 提交中击败了59.35% 的用户
61 内存消耗 :10.2 MB, 在所有 cpp 提交中击败了90.66%的用户
62 */

```

```

1  class Solution
2  {
3
4      private:
5
6          int SearchLeft( vector<int>&    nums    ,
7                          int            target ,
8                          int            left   ,
9                          int            right
10                     )
11     {
12         bool flag = false;
13         int mid = 0;
14         while(left <= right)
15         {
16             mid = left + (right - left) / 2;
17             if(nums[mid]==target)
18             {
19                 flag = true;
20                 break;
21             }
22             else if(nums[mid] < target)
23             {
24                 left = mid + 1;
25             }
26             else
27             {
28                 right = mid - 1;
29             }
30         }
31
32         if(flag)
33         {
34             return SearchLeft(nums,target,0,mid-1);
35         }
36         else
37         {
38             return right+1;
39         }
40     }
41
42 }
43
44 int SearchRight(    vector<int>&    nums    ,
45                   int            target ,
46                   int            left   ,
47                   int            right
48               )
49 {
50     bool flag = false;
51     int mid = 0;
52     int rright = right;
53     while(left <= right)
54     {
55         mid = left + (right - left) / 2;
56         if(nums[mid]==target)
57         {

```

```

58         flag = true;
59         break;
60     }
61     else if(nums[mid] < target)
62     {
63         left = mid + 1;
64     }
65     else
66     {
67         right = mid - 1;
68     }
69 }
70
71 if(flag)
72 {
73     return SearchRight(nums,target,mid+1,right);
74 }
75
76 else
77 {
78     return left-1;
79 }
80 }
81
82 public:
83     vector<int> searchRange(vector<int>& nums, int target)
84     {
85         vector<int> vi(2,-1);
86         int left = 0 ;
87         int right = nums.size()-1 ;
88         int mid = 0 ;
89         bool flag = false ;
90
91         /*看是否有*/
92         while(right >= left)
93         {
94             mid = left + (right - left)/2;
95             if(nums[mid] == target)
96             {
97                 flag = true;
98                 break;
99             }
100             else if(nums[mid] < target)
101             {
102                 left = mid + 1;
103             }
104             else
105             {
106                 right = mid - 1;
107             }
108         }
109
110         if(flag)
111         {
112             vi[0] = SearchLeft(nums,target,0,mid-1);
113             vi[1] = SearchRight(nums,target,mid+1,nums.size()-1);
114         }
115

```

```

116
117         return vi;
118     }
119 };
120
121 /*
122 执行结果:
123 通过
124 显示详情
125 执行用时 :8 ms, 在所有 cpp 提交中击败了90.42% 的用户
126 内存消耗 :10.2 MB, 在所有 cpp 提交中击败了89.08%的用户
127 */
128

```

方法四:C++_二分查找法+偏mid寻找边界

```

1  class Solution
2  {
3      public:
4
5          vector<int> searchRange(    vector<int>&    nums    ,
6                                     int    target
7                                     )
8      {
9          vector<int> vi(2,-1);
10         int length = nums.size();
11         if(length<1)
12         {
13             return vi;
14         }
15
16         int left    =    0            ;
17         int right   =    0            ;
18         int mid     =    0            ;
19
20         /*找最左边界*/
21         left    =    0            ;
22         right   =    length - 1    ;
23         while(left<right)
24         {
25             mid =    left + ( right - left ) / 2; /*求均值时要求向下取整*/
26             if(nums[mid]>=target)
27             {
28                 right = mid;
29             }
30             else
31             {
32                 left = mid + 1;
33             }
34         }
35         /*若不存在*/
36         if(nums[left]!=target)
37         {
38             return vi;
39         }
40

```

```
41     vi[0] = left;
42     /*找最右边界*/
43     right = length-1;
44     while(left<right)
45     {
46         mid = left + (right-left)/2 + 1;    /*求均值时要求向上取整*/
47         if(nums[mid]<=target)
48         {
49             left = mid;
50         }
51         else
52         {
53             right = mid-1;
54         }
55     }
56     vi[1]=left;
57     return vi;
58 }
59 };
60
61
62 /*
63 执行结果:
64 通过
65 显示详情
66 执行用时 :8 ms, 在所有 cpp 提交中击败了90.42% 的用户
67 内存消耗 :10.2 MB, 在所有 cpp 提交中击败了92.16%的用户
68 */
```