```
/*
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。
你的算法时间复杂度必须是 O(log n) 级别。
如果数组中不存在目标值，返回 [-1, -1]。
示例 1:
输入: nums = [5,7,7,8,8,10], target = 8
输出: [3,4]
示例 2:
输入: nums = [5,7,7,8,8,10], target = 6
输出: [-1,-1]
来源：力扣（LeetCode）
链接：https://leetcode-cn.com/problems/find-first-and-last-position-of-element-in-sorted-array
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。
*/
```

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

```
class Solution
{
    public:
```

```
        vector<int> searchRange(vector<int>& nums, int target)
        {
            vector<int> vi(2,-1);

            if(nums.size() < 1)
            {
                return vi;
            }

            if(nums[0]== target)
            {
                vi[0] = 0;
            }

            if(nums[nums.size()-1]== target)
            {
                vi[1] = nums.size()-1;
            }

            for(int i = 1 ; i < nums.size();i++)
            {
                if(nums[i]==target && nums[i-1]!=target )
                {
                    vi[0] = i;
                }

                if(nums[i]!=target && nums[i-1]==target )
                {
                    vi[1] = i-1;
                    break;
                }

            }
            return vi;
        }
};

/*
https://leetcode-cn.com/submissions/detail/38742435/
执行结果:
通过
显示详情
执行用时 :20 ms, 在所有 cpp 提交中击败了10.54% 的用户
内存消耗 :10.1 MB, 在所有 cpp 提交中击败了93.37%的用户
*/
```

方法二:C++_二分查找法+待优化寻找边界

```
class Solution
{
    public:
        vector<int> searchRange(vector<int>& nums, int target)
        {
            vector<int> vi(2,-1);
            int left    =   0               ;
            int right   =   nums.size()-1   ;
```

```
            int mid     =   0               ;
            bool flag   =   false           ;

            while(right >= left)
            {
                mid = left + (right - left)/2;
                if(nums[mid] ==  target)
                {
                    flag = true;
                    break;
                }
                else if(nums[mid] > target)
                {
                    right = mid - 1;
                }
                else
                {
                    left = mid + 1;
                }
            }

            if(flag)
            {
                left = mid;
                right = mid;
                while(left >=0 && nums[left]==target)
                {
                    left--;
                }


                while(right < nums.size() && nums[right]==target)
                {
                    right++;
                }

                vi[0] = left + 1;
                vi[1] = right - 1;


            }
            return vi;
        }
};

/*

https://leetcode-cn.com/submissions/detail/38737198/
执行结果：
通过
显示详情
执行用时 :12 ms, 在所有 cpp 提交中击败了59.35% 的用户
内存消耗 :10.2 MB, 在所有 cpp 提交中击败了90.66%的用户
*/
```

方法三:C++_二分查找法+递归寻找边界

```cpp
class Solution
{

    private:

        int SearchLeft( vector<int>&    nums    ,
                        int             target  ,
                        int             left    ,
                        int             right
        )
        {
            bool flag = false;
            int  mid  = 0;
            while(left <= right)
            {
                mid = left + (right - left) / 2;
                if(nums[mid]==target)
                {
                    flag = true;
                    break;
                }
                else if(nums[mid] < target)
                {
                    left = mid + 1;
                }
                else
                {
                    right = mid - 1;
                }
            }

            if(flag)
            {
                return SearchLeft(nums,target,0,mid-1);

            }
            else
            {
                return right+1;
            }

        }

        int SearchRight(    vector<int>&    nums    ,
                            int             target  ,
                            int             left    ,
                            int             right
        )
        {
            bool flag = false;
            int  mid  = 0;
            int  rright = right;
            while(left <= right)
            {
                mid = left + (right - left) / 2;
                if(nums[mid]==target)
                {
```

```
                    flag = true;
                    break;
                }
                else if(nums[mid] < target)
                {
                    left = mid + 1;
                }
                else
                {
                    right = mid - 1;
                }
            }

            if(flag)
            {
                return SearchRight(nums,target,mid+1,rright);

            }
            else
            {
                return left-1;
            }
        }

    public:
        vector<int> searchRange(vector<int>& nums, int target)
        {
            vector<int> vi(2,-1);
            int  left    =   0               ;
            int  right   =   nums.size()-1   ;
            int  mid     =   0               ;
            bool flag    =   false           ;

            /*看是否有*/
            while(right >= left)
            {
                mid = left + (right - left)/2;
                if(nums[mid] == target)
                {
                    flag = true;
                    break;
                }
                else if(nums[mid] < target)
                {
                    left  = mid + 1;
                }
                else
                {
                    right = mid - 1;
                }
            }

            if(flag)
            {
                vi[0] = SearchLeft(nums,target,0,mid-1);
                vi[1] = SearchRight(nums,target,mid+1,nums.size()-1);
            }

```

```
            return vi;
        }
};

/*
执行结果：
通过
显示详情
执行用时 :8 ms, 在所有 cpp 提交中击败了90.42% 的用户
内存消耗 :10.2 MB, 在所有 cpp 提交中击败了89.08%的用户
*/

```

方法四:C++_二分查找法+偏mid寻找边界

```
class Solution
{
    public:

        vector<int> searchRange(    vector<int>&    nums    ,
                                    int             target
                                )
        {
            vector<int> vi(2,-1);
            int length = nums.size();
            if(length<1)
            {
                return vi;
            }

            int left    =   0           ;
            int right   =   0           ;
            int mid     =   0           ;

            /*找最左边界*/
            left    =   0           ;
            right   =   length - 1  ;
            while(left<right)
            {
                mid =   left + ( right - left ) / 2; /*求均值时要求向下取整*/
                if(nums[mid]>=target)
                {
                    right = mid;
                }
                else
                {
                    left = mid + 1;
                }
            }
            /*若不存在*/
            if(nums[left]!=target)
            {
                return vi;
            }

```

```
            vi[0] = left;
            /*找最右边界*/
            right = length-1;
            while(left<right)
            {
                mid = left + (right-left)/2 + 1;    /*求均值时要求向上取整*/
                if(nums[mid]<=target)
                {
                    left = mid;
                }
                else
                {
                    right = mid-1;
                }
            }
            vi[1]=left;
            return vi;
        }
};


/*
执行结果：
通过
显示详情
执行用时 :8 ms, 在所有 cpp 提交中击败了90.42% 的用户
内存消耗 :10.2 MB, 在所有 cpp 提交中击败了92.16%的用户
*/
```

AlimyBreak
2019.12.19