

## 二分查找法的第一种——闭区间

In [15]:

```
1  ### 查找不重复元素中的target
2
3  def non_search1(nums, target):
4      left, right = 0, len(nums)-1
5      while left <= right:
6          mid = left + (right-left)//2
7          if nums[mid] == target:
8              return mid
9          elif nums[mid] < target:
10             left = mid + 1 # 永远不变
11          elif nums[mid] > target:
12             right = mid - 1
13     return -1
14
15
16
17 # 找右缩左
18 def left_search1(nums, target):
19     left, right = 0, len(nums)-1
20     while left <= right:
21         mid = left + (right-left)//2
22         if nums[mid] == target:
23             right = mid - 1 # 同 nums[mid] > target这种情况, 不断压缩右边
24         elif nums[mid] < target:
25             left = mid + 1 # 永远不变
26         elif nums[mid] > target:
27             right = mid - 1 # 压缩右边界
28
29     # 越界条件
30     if left == len(nums) or nums[left] != target:
31         return -1
32     return left
33
34 # 找左缩右
35 def right_search1(nums, target):
36     left, right = 0, len(nums)-1
37     while left <= right:
38         mid = left + (right-left)//2
39         if nums[mid] == target:
40             left = mid + 1 # 同 nums[mid] < target这种情况, 不断压缩左边
41         elif nums[mid] < target:
42             left = mid + 1 # 永远不变
43         elif nums[mid] > target:
44             right = mid - 1
45
46     # 越界条件
47     if right < 0 or nums[right] != target:
48         return -1
49     return right
```

## 二分查找法的第二种——开区间

In [8]:

```
1  ### 查找不重复元素中的target
2
3  def non_search2(nums, target):
4      left, right = 0, len(nums)
5      while left < right:      # right 变就变
6          mid = left + (right-left)//2
7          if nums[mid] == target:
8              return mid
9          elif nums[mid] < target:
10             left = mid + 1  # 永远不变
11          elif nums[mid] > target:
12             right = mid    # right 变就变
13      return -1
14
15  def left_search2(nums, target):
16      left, right = 0, len(nums)
17      while left < right:
18          mid = left + (right-left)//2
19          if nums[mid] == target:
20             right = mid    # 同 nums[mid] > target这种情况，不断压缩右边
21          elif nums[mid] < target:
22             left = mid + 1  # 永远不变
23          elif nums[mid] > target:
24             right = mid
25      return left  # 返回左边界
26
27  def right_search2(nums, target):
28      left, right = 0, len(nums)
29      while left < right:
30          mid = left + (right-left)//2
31          if nums[mid] == target:
32             left = mid + 1  # 同 nums[mid] < target这种情况，不断压缩左边
33          elif nums[mid] < target:
34             left = mid + 1  # 永远不变
35          elif nums[mid] > target:
36             right = mid
37      return right-1  # 记得-1，因为是开区间
```

In [ ]:

```
1 # 875 二分吃香蕉
2 def minEatingSpeed(self, piles: List[int], h: int) -> int:
3     # 二分法, 速度有序
4     def canFindish(piles, speed):
5         # H 小时内能吃完
6         time = 0
7         for n in piles:
8             time += (n//speed)+(1 if n%speed > 0 else 0) # 向上取整
9         return time <= h
10    left = 1 # 速度从1开始
11    right = max(piles)
12    while left <= right:
13        mid = left + (right-left)//2
14        if canFindish(piles, mid):
15            right = mid - 1
16        else:
17            left = mid + 1
18    return left
19
20 # 1011 货运船
21 def shipWithinDays(self, weights: List[int], days: int) -> int:
22     def canFindish(weights, D, cap):
23         # H 小时内能吃完
24         days = 1
25         current = 0
26         for weight in weights:
27             current += weight
28             if current > cap:
29                 days += 1
30                 current = weight
31         return days <= D
32    left = max(weights) # 速度从1开始
33    right = sum(weights)
34    while left <= right:
35        mid = left + (right-left)//2
36        if canFindish(weights, days, mid):
37            right = mid - 1
38        else:
39            left = mid + 1
40    return left
```