<https://leetcode.cn/problems/search-insert-position/solution/te-bie-hao-yong-de-er-fen-cha-fa-fa-mo-ban-python-/>

<https://leetcode.cn/problems/median-of-two-sorted-arrays/solution/he-bing-yi-hou-zhao-gui-bing-guo-cheng-zhong-zhao-/> 评论区的问题非常好

left < right 这种写法最核心的部分是：把待搜索区间分成两个部分，把要查找的元素留在退出循环以后。（这一句非常重要，我几乎所有的二分查找的题解都采用了这样的思想。）当 left == right 的时候，表示 left 与 right 重合，left（或者 right） 所在的位置就是要找的目标元素的位置。

写对「二分查找」，在于我们能够根据题意：**得到某种单调性，和可以逐步缩小搜索规模的条件，进而准确地设计可以使得搜索区间缩小的条件。**

做二分查找问题的思路一般是：先分析单调性，或者是利用题目中已经给出的单调性解题，逐步缩小搜索的范围。要特别注意的是题目中一些很细致的描述，例如这个问题「至多」、「至少」，有些时候变量之间的关系是负相关的，思考起来就会很绕。所以一定要搞清楚题目中说的「大于」是「大于等于」还是「严格大于」。

一般来说，我的经验是去找题目要找的数的性质，然后对这个性质取反，就可以达到缩减搜索区间的目的。这也只是一个经验，是想告诉大家：去考虑看到的 nums[mid] 在什么情况下不是目标元素，这个角度在绝大多数情况下更容易一点，最终的目的也是为了确定下一轮搜索的区间是什么，进而考虑边界该怎么收缩。

所以weiwei的做法是只思考其中一个部分（好想的，不容易出错的部分），这一部分我非常确定的时候，另一部分就是第一部分的反面。这种写二分法的思路其实就是在一个区间里不断地“砍”去左边和右边的部分（即那些不是目标元素的部分），直到最后剩下一个数，这个数是不是我们要的就很容易判断。

从代码层面看，就是在 [left, right] 这样的区间里在循环中不断把 left 向右边移动，把 right 向左边移动，直到 left == right 成立的时候，退出循环，[left, left] 这个区间就只有一个元素，这个元素是不是我们要的就很容易判断啦。当 left 和 right 重合的时候，我们就找了目标元素（如果目标元素在数组中存在的话），所以返回 left 和 right 都是可以的。而middle 是在二分查找的过程中，猜测的某个位置的值，没有说结束循环时一定有middle=left=right，这件事情没有根据。

* **只有把区间分成两个部分的时候，退出循环以后 left 与 right 才会重合。如果我们不能确定目标元素一定在搜索区间里，还需要再单独判断一次。**

**再说一遍：只有在while循环中把区间分成「两个部分」，最后left和right才能重合，下面是划分成三个区间不重合的例子**<https://leetcode.cn/problems/find-first-and-last-position-of-element-in-sorted-array/solution/si-lu-hen-jian-dan-xi-jie-fei-mo-gui-de-er-fen-cha/945817>

1. 划分区间的逻辑是 left = mid + 1; 和 right = mid; 时，while(left < right) 退出循环以后 left == right 成立，此时 mid 中间数正常下取整就好；
2. 划分区间的逻辑是 left = mid; 和 right = mid - 1; 时，while(left < right) 退出循环以后 left == right 成立，此时为了避免死循环，mid 中间数需要改成上取整。

* **在写二分搜索的时候,不要有分支逻辑和中位数取法不匹配的做法:**

这就是指，在二分搜索缩小区间的时候,不要下面两种分配区间方式同时出现.

1. 分配区间方式1: left = mid + 1; 和 right = mid；
2. 分配区间方式2: left = mid 和 right = mid - 1。

第一种 [分配区间方式] 求取mid方式应该是 mid= (left + right) / 2

第二种 [分配区间方式] 求取mid方式应该是 mid = (left + right + 1) // 2

**两种分配区间方式同时出现, 将造成逻辑混乱**

例子就是 81 题中(best 版本) 进入 "nums[mid] < nums[left]" 分支时, 按理说 [mid, right] 区间是有序的, 下面的代码可以写成

文本

描述已自动生成

这样是可以成功运行的，但是出现了逻辑混乱的问题。上一个 if 分支分配区间的方式是

文本

描述已自动生成

也就是 "left = mid + 1; 和 right = mid"。但是 我们应该在同一段二分查找代码中保持分支一致，所以在这个 elif 分支中我们强行认为 [mid + 1, right] 之间是有序的, 代码变成

文本

描述已自动生成

这样写同一段二分查找代码其分支逻辑满足了 分配区间方式1，可以得到正确的解！

* **题目 1300 的两个解法中展示了计算 [第一个大于等于某个值] 和 [最后一个小于等于某个值] 的整数或者数组中某个数的计算方法**
* **Mid 为什么 在 left=mid的时候要加1？**

<https://leetcode.cn/problems/sqrtx/solution/er-fen-cha-zhao-niu-dun-fa-python-dai-ma-by-liweiw/>

应用程序

中度可信度描述已自动生成

* **如果题目要我们找的元素满足的性质是「条件 1 && 条件 2」如何写判定条件**

从不满足的性质去设计判别函数得到的条件会相对简单一点，这是因为 「条件 1 && 条件 2」的反面是 「条件 1取反」或者「条件 2 取反」。即：对其中一个条件取反，设计判别函数就可以逐步逼近，找到目标元素了。

「条件 1 || 条件 2」只对其中一个条件取反是不可以的，例如找「大于 3 或者小于 0」 的反面是 「小于等于 3 并且 大于等于 0」 。

若干个条件中间是「并且」连接的时候，找反面的时候，只需要对其中一个条件取反就可以；

若干个条件中间是「或者」连接的时候，找反面的时候，需要对每一个条件取反，并且这些条件之间需要用「并且」连接。

一些经验:

1. 明确 while(left <= right) 和 while(left < right) 这两种写法其实在思路上有本质差别， while(left <= right) 在循环体内部直接查找元素，而 while(left < right) 在循环体内部一直在排除元素，第 2 种思路在解决复杂问题的时候，可以使得问题变得简单；
2. 始终在思考下一轮搜索区间是什么，把它作为注释写到代码里面，就能帮助我们搞清楚边界是不是能取到，等于、+1 、-1 之类的细节；
3. 思考清楚每一行代码背后的语义是什么，保证语义上清晰，也是写对代码，减少 bug 的一个非常有效的策略。

* **宫水三叶关于二分查找的说明：**

「二分」不是单纯指从有序数组中快速找某个数，这只是「二分」的一个应用。「二分」的本质是两段性，并非单调性。只要一段满足某个性质，另外一段不满足某个性质，就可以用「二分」。 ***33题*** 中 经过旋转的数组，显然前半段满足 >= nums[0]，而后半段不满足 >= nums[0]。我们可以以此作为依据，通过「二分」找到旋转点。 而 ***81题*** 中 元素并不唯一。这意味着我们无法直接根据与 nums[0] 的大小关系，将数组划分为两段，即无法通过「二分」来找到旋转点。 因为「二分」的本质是二段性，并非单调性。只要一段满足某个性质，另外一段不满足某个性质，就可以用「二分」。 所以需要做一些处理，使其单调性恢复