二分查找

模板

```
# 二分查找

left, right = 0, len(List) - 1
while left <= right:         # <=
    mid = (left + right)//2
    if List[mid] == target:
        # find the target
        break or return result
    elif List[mid] > target:
        right = mid -1
    else:
        left = mid + 1
```

适用条件

单调数组（递增或递减）【否则就只能遍历了】

存在上下界（bounded）【有边界，向中间收缩】

通过索引访问（index accessible ）

题目

704. 二分查找

难度简单134

给定一个 `n` 个元素有序的（升序）整型数组 `nums` 和一个目标值 `target` ，写一个函数搜索 `nums` 中的 `target`，如果目标值存在返回下标，否则返回 `-1`。

示例 1:

```
输入: nums = [-1,0,3,5,9,12], target = 9
输出: 4
解释: 9 出现在 nums 中并且下标为 4
```

示例 2:

```
输入: nums = [-1,0,3,5,9,12], target = 2
输出: -1
解释: 2 不存在 nums 中因此返回 -1
```

提示:

1. 你可以假设 `nums` 中的所有元素是不重复的。
2. `n` 将在 `[1, 10000]` 之间。
3. `nums` 的每个元素都将在 `[-9999, 9999]` 之间。
4. 时间复杂度：**O(log*N*)**。
5. 空间复杂度：**O(1)**。

```
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        """
        审题：
        升序，可以用二分查找
        return：idx；-1
        304ms, beats 75.69%
        """
        left, right = 0, len(nums) - 1       # error: -1 not +1

        while left <= right:
            # error: <= must have =
            mid = left + (right - left) // 2 # 目的：防止left+right太大导致溢出
            # mid = (left + right) // 2
            if nums[mid] == target:          # error: nums[mid]
                return mid
            elif nums[mid] > target:
                # target is on the left of [mid]
                right = mid -1
            else:
                # target is on the right of [mid]
                left = mid + 1
        return -1
```

1、为什么 while 循环的条件中是 <=，而不是 <?

答：因为初始化 right 的赋值是 nums.length - 1，即最后一个元素的索引，而不是 nums.length。

这二者可能出现在不同功能的二分查找中，区别是：前者相当于两端都闭区间 [left, right]，后者相当于左闭右开区间 [left, right)，因为索引大小为 nums.length 是越界的。

我们这个算法中使用的是前者 [left, right] 两端都闭的区间。这个区间其实就是每次进行搜索的区间。

什么时候应该停止搜索呢？当然，找到了目标值的时候可以终止：

```
if(nums[mid] == target)
    return mid;
```

但如果没找到，就需要 while 循环终止，然后返回 -1。那 while 循环什么时候应该终止？搜索区间为空的时候应该终止，意味着你没得找了，就等于没找到嘛。

while(left <= right) 的终止条件是 left == right + 1，写成区间的形式就是 [right + 1, right]，或者带个具体的数字进去 [3, 2]，可见这时候区间为空，因为没有数字既大于等于 3 又小于等于 2 的吧。所以这时候 while 循环终止是正确的，直接返回 -1 即可。

while(left < right) 的终止条件是 left == right，写成区间的形式就是 [left, right]，或者带个具体的数字进去 [2, 2]，这时候区间非空，还有一个数 2，但此时 while 循环终止了。也就是说这区间 [2, 2] 被漏掉了，索引 2 没有被搜索，如果这时候直接返回 -1 就是错误的。

作者：labuladong
链接：https://leetcode-cn.com/problems/binary-search/solution/er-fen-cha-zhao-xiang-jie-by-labuladong/
来源：力扣（LeetCode）

69. x 的平方根

https://leetcode-cn.com/problems/sqrtx/

难度简单433

实现 `int sqrt(int x)` 函数。

计算并返回 *x* 的平方根，其中 *x* 是非负整数。

由于返回类型是整数，结果只保留整数的部分，小数部分将被舍去。

示例 1:

```
输入: 4
输出: 2
```

示例 2:

```
输入: 8
输出: 2
说明: 8 的平方根是 2.82842...,
     由于返回类型是整数，小数部分将被舍去。
```

一：二分查找

- 时间复杂度：O(*log*x*)，即为二分查找需要的次数。
- 空间复杂度：*O*(1)。

  Python

```python
class Solution:
def mySqrt(self, x: int) -> int:
    """
    审题：保留整数部分，所以不是=，而是一个范围，所以不用-1
    二分查找：
    """
    # special condition
    if x == 1: return

    left, right = 0, x

    while left <= right:
        mid = left + (right - left) // 2
        if mid * mid <= x < (mid + 1) * (mid + 1):
            return mid
        elif mid * mid >= x:
            # target is on the left of [[mid]
            right = mid
        else:
```

```
                left = mid
```

```python
# Binary search
class Solution:
    def mySqrt(self, x: int) -> int:
        """
        审题：保留整数部分，所以不是=，而是一个范围，所以不用-1
        二分查找：
        time：O(logx)
        space: O(1)
        time: 44ms, beats 83.81%
        """
        # deal with exception
        if x == 1: return 1

        left, right = 0, x          # error: x

        while left <= right:        # error: must have =
            mid = left + (right - left) // 2
            if mid * mid <= x < (mid + 1) * (mid + 1):      # 右边是<，没有=
                return mid
            elif x < mid * mid:
                # target is on the left of [[mid]
                # above: mid * mid == x, include mid, so no need to -1 ???
                right = mid
            else:
                left = mid + 1
```

java

测试用例值变大，int不行了，因此用了long，最后返回结果时，(int)转一下

return left and right 试一下即可

二：牛顿迭代法

https://leetcode-cn.com/problems/sqrtx/solution/x-de-ping-fang-gen-by-leetcode-solution/

用的更多

todo： 看前两个题解，了解数学思路

367. 有效的完全平方数

难度简单140

给定一个正整数 *num*，编写一个函数，如果 *num* 是一个完全平方数，则返回 True，否则返回 False。

说明：不要使用任何内置的库函数，如 `sqrt`。

示例 1：

```
输入：16
输出：True
```

示例 2：

```
输入：14
输出：False
```

一：二分查找

自己写出来的哈~

```python
class Solution:
    def isPerfectSquare(self, num: int) -> bool:
        # time：44ms, beats 52.56%
        # 时间复杂度：O(logN)。
        # 空间复杂度：O(1)。
        # special case !!!
        if num == 1: return True
```

```
        left, right = 0, num

        while left <= right:                # error: must have "="
            mid = left + (right - left) // 2
            guess_squared = mid * mid       # error: is mid, not x
            if guess_squared == num:        # error：双=, compare with num
                return True                 # error: return True or False
            elif guess_squared > num:
                # target is on the left on [mid]
                right = mid - 1
            else:
                left = mid + 1
        return False
```

```
from international website:

class Solution(object):
    def isPerfectSquare(self, num):
        """
        :type num: int
        :rtype: bool
        """
        if num<0:
            return False
        x,i=0,1
        while x<num:
            x+=i
            i+=2
        return x==num
```

二：牛顿迭代法 Newton's method

```
class Solution(object):
    def isPerfectSquare(self, num):
        """
        :type num: int
        :rtype: bool
        """
        if num < 0: return False
        if num <= 1: return True
        n = num/2  # start guessing using n = num/2
        while n*n!= num:
            inc = (num-n*n)/(2*n)
            n += inc
            if -1 <= inc <= 1: break
        if n*n < num: n+=1
        if n*n > num: n-=1
        return n*n == num
```

f(x) = x^2 (find x that f(x) = num)

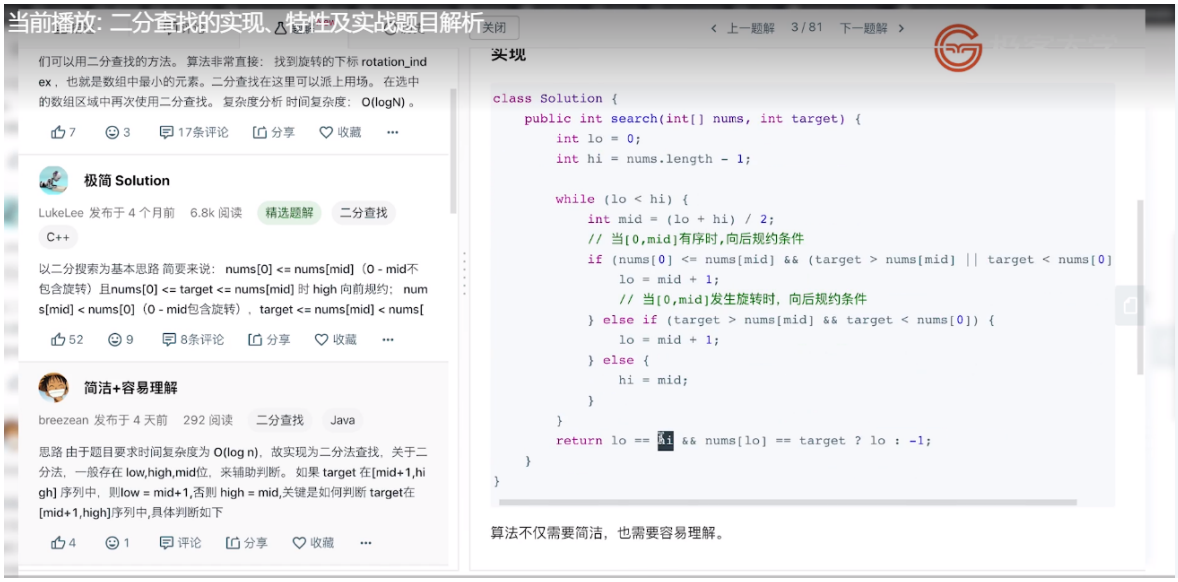
f'(x) = 2*x

start process with x = n (any positive number)

if f(x) != num, update x = x + (num - f(x))/f'(x) = x + (num - n^2)/(2n)

https://leetcode-cn.com/problems/valid-perfect-square/solution/you-xiao-de-wan-quan-ping-fang-shu-by-leetcode/

复杂二分查找 （作业1）



153. 寻找旋转排序数组中的最小值

Category	Difficulty	Likes	Dislikes
algorithms	Medium (50.94%)	197	-

▶ **Tags**

▶ **Companies**

假设按照升序排序的数组在预先未知的某个点上进行了旋转。

(例如，数组 `[0,1,2,4,5,6,7]` 可能变为 `[4,5,6,7,0,1,2]`)。

请找出其中最小的元素。

你可以假设数组中不存在重复元素。

示例 1:

```
输入: [3,4,5,1,2]
输出: 1
```

示例 2:

```
输入: [4,5,6,7,0,1,2]
输出: 0
```

Discussion | Solution

二分查找

代码

Python

```
class Solution:
    def findMin(self, nums: List[int]) -> int:
        """
        1 sort
        time: O(NlogN)
        space: O(1) ?

        2 min_num = nums[0], traverse and compare to find the min num
        time: O(N)
        space: O(1)  but the time is bigger than 1, why?

        3 二分查找 binary search -- from national website
        time:
        space:
        """
        # time: 32ms, beats 96.35%

        if nums[-1] > nums[0]: return nums[0]
        # define the left and right index
        left, right = 0, len(nums) - 1

        # terminator
        while left < right:      # error: < ! ! !
            # define mid
            mid = left + (right - left) // 2
            # compare between [mid] and [right] to narrow the range
            if nums[mid] < nums[right]:              # error: nums[mid] 别忘了
nums
                # eg: [4,5,1,2,3] and [6,7,1,2,3,4]
                # the min num is mid or on the left of mid
                right = mid
            else:
                # eg: [3,4,5,1,2]
                # the min num is on the right of mid
                # no duplicate num, so no = condition
                left = mid + 1

        # at last, left = right, return num, not index -- error!!!
        return nums[left]

================================================================================
=========
low解法
================================================================================
=========
    def findMin1(self, nums: List[int]) -> int:
        return sorted(nums)[0]


    def findMin2(self, nums: List[int]) -> int:
        # 2 time:44ms, beats 45.49%
        # define min_num
        min_num = nums[0]
        for num in nums:
```

```
            # traverse each num in nums, if smaller than min_num, exchange
            if num < min_num:
                num, min_num = min_num, num

        return min_num
```

from national website:

```
class Solution:
    def findMin(self, nums: List[int]) -> int:
        # 32ms, 96.33%
        # set left and right bounds
        left, right = 0, len(nums) - 1

        # left and right both converge to the minimum index;
        # DO NOT use left <= right because that would loop forever
        while left < right:
            # find the middle value between the left and right bounds (their
average);
            # can equivalently do: mid = left + (right - left) // 2,
            # if we are concerned left + right would cause overflow (which would
occur
            # if we are searching a massive array using a language like Java or
C that
            # has fixed size integer types)
            mid = (left + right) // 2
            if nums[mid] > nums[right]:
                # we KNOW the pivot must be to the right of the middle:
                # if nums[mid] > nums[right], we KNOW that the
                # pivot/minimum value must have occurred somewhere to the right
                # of mid, which is why the values wrapped around and became
smaller.

                # example:  [3,4,5,6,7,8,9,1,2]
                # in the first iteration, when we start with mid idx = 4, right
idx = 9.
                # if nums[mid] > nums[right], at some point to the right of mid,
                # the pivot must have occurred, which is why the values wrapped
around
                # so that nums[right] is less then nums[mid]

                # we know that the number at mid is greater than at least
                # one number to the right, so we can use mid + 1 and
                # never consider mid again; we know there is at least
                # one value smaller than it on the right
                left = mid + 1
            else:
                # here, nums[mid] <= nums[right]:
                # we KNOW the pivot must be at mid or to the left of mid:
                # if nums[mid] <= nums[right], we KNOW that the pivot was not
encountered
                # to the right of middle, because that means the values would
wrap around
                # and become smaller (which is caught in the above if
statement).
                # this leaves the possible pivot point to be at index <= mid.
```

```
                # example: [8,9,1,2,3,4,5,6,7]
                # in the first iteration, when we start with mid idx = 4, right
idx = 9.
                # if nums[mid] <= nums[right], we know the numbers continued
increasing
                # to the right of mid, so they never reached the pivot and
wrapped around
                # around. therefore, we know the pivot must be at index <= mid.

                # we know that nums[mid] <= nums[right].
                # therefore, we know it is possible for the mid index to store a
smaller
                # value than at least one other index in the list (at right), so
we do
                # not discard it by doing right = mid - 1. it still might have
the minimum value.
                right = mid  # not mid - 1, eg: [4,5,1,2,3]

        # at this point, left and right converge to a single index (for minimum
value) since
        # our if/else forces the bounds of left/right to shrink each iteration:

        # when left bound increases, it does not disqualify a value
        # that could be smaller than something else (we know nums[mid] >
nums[right],
        # so nums[right] wins and we ignore mid and everything to the left of
mid).

        # when right bound decreases, it also does not disqualify a
        # value that could be smaller than something else (we know nums[mid] <=
nums[right],
        # so nums[mid] wins and we keep it for now).

        # so we shrink the left/right bounds to one value,
        # without ever disqualifying a possible minimum
        return nums[left]
```

Java

```
class Solution {
    public int findMin(int[] nums) {
        if (nums.length == 1) return nums[0];

        int left = 0, right = nums.length - 1;
        if (nums[right] > nums[left]) return nums[left];

        while (left < right) {
            int mid = (left + right) / 2;
            if (nums[mid] > nums[mid + 1] ) return nums[mid + 1];
            if (nums[mid - 1] > nums[mid]) return nums[mid];
            if (nums[mid] > nums[0]) {
                left = mid + 1;
            } else {
```

```
                right = mid - 1;
            }
        }
        return -1;
    }
}

// 0s, beats 100%
```

```
from national web
The minimum element must satisfy one of two conditions: 1) If rotate, A[min] < A[min - 1]; 2) If not, A[0]. Therefore, we can use binary search: check the middle element, if it is less than previous one, then it is minimum. If not, there are 2 conditions as well: If it is greater than both left and right element, then minimum element should be on its right, otherwise on its left.

public class Solution {
    public int findMin(int[] num) {
        if (num == null || num.length == 0) {
            return 0;
        }
        if (num.length == 1) {
            return num[0];
        }
        int start = 0, end = num.length - 1;
        while (start < end) {
            int mid = (start + end) / 2;
            if (mid > 0 && num[mid] < num[mid - 1]) {
                return num[mid];
            }
            if (num[start] <= num[mid] && num[mid] > num[end]) {
                start = mid + 1;
            } else {
                end = mid - 1;
            }
        }
        return num[start];
    }
}
```

74. 搜索二维矩阵 (报错！！！)

难度中等198

编写一个高效的算法来判断 *m* x *n* 矩阵中，是否存在一个目标值。该矩阵具有如下特性：

- 每行中的整数从左到右按升序排列。
- 每行的第一个整数大于前一行的最后一个整数。

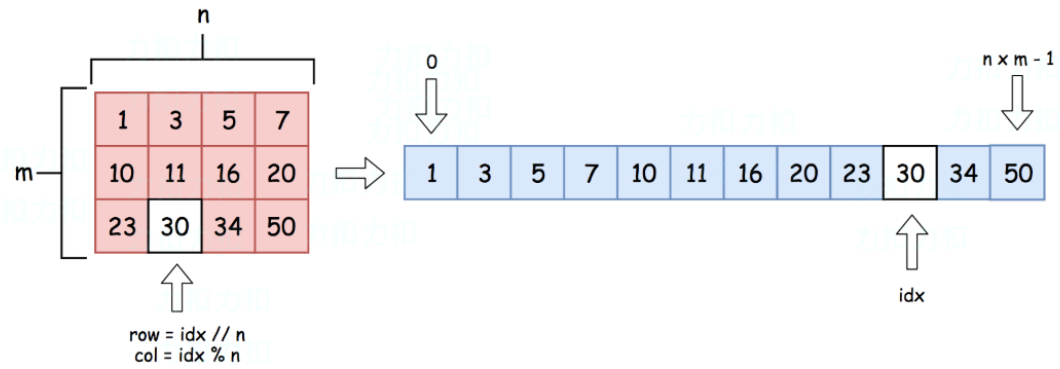
示例 1:

输入:

```
matrix = [
  [1,   3,  5,  7],
  [10, 11, 16, 20],
  [23, 30, 34, 50]
]
target = 3
输出: true
```

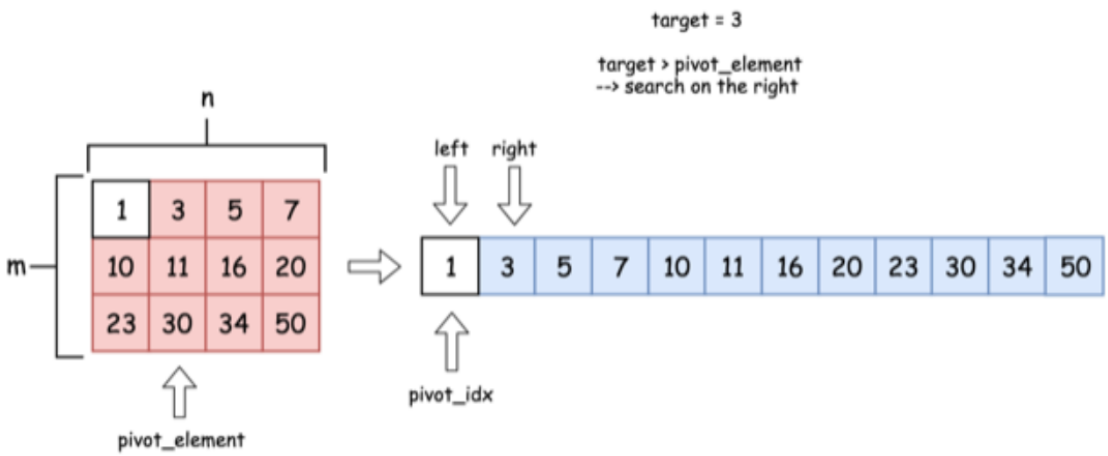
二分查找

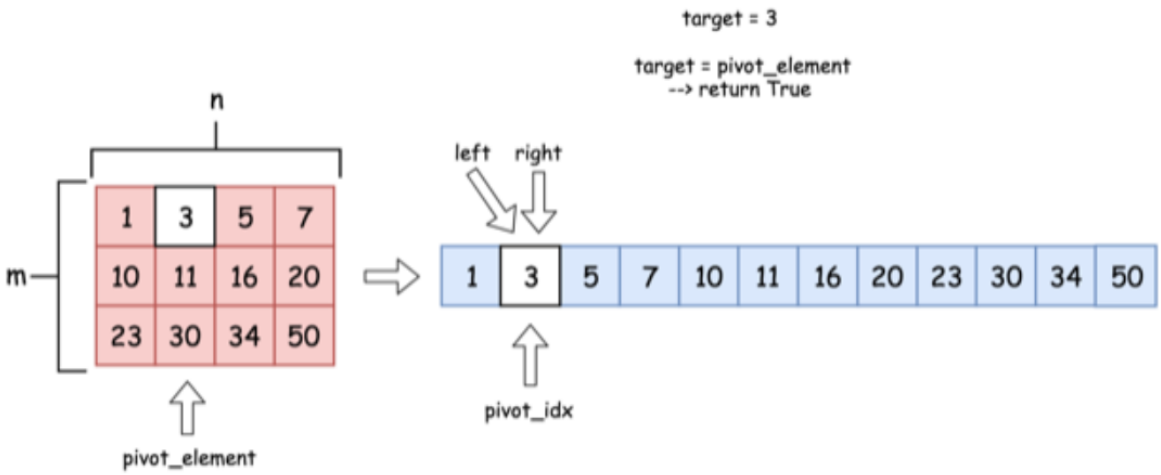
注意到输入的 `m x n` 矩阵可以视为长度为 `m x n` 的有序数组。



由于该 *虚* 数组的序号可以由下式方便地转化为原矩阵中的行和列 (*我们当然不会真的创建一个新数组*)，该有序数组非常适合二分查找。

> `row = idx // n` ， `col = idx % n` 。





代码

Python

```
class Solution:
    def searchMatrix(self, matrix: List[List[int]], target: int) -> bool:
        # time: O(log(mn))
        # 36ms, beats 90.91%
        if not matrix or target is None:            # error: target, not int, 拼写!
        # error: target is None 和 target = 0 不一样!  注意matrix拼写!
            return False                            # error: Flase 大写

        rows, cols = len(matrix), len(matrix[0])    # error: rows 和 cols 反了
        left, right = 0, rows * cols - 1            # error: idx, not num
        while left <= right:
            mid = left + (right - left) // 2
            num = matrix[mid // cols][mid % cols]   # error: mid?

            # compare mid or mid_expression with target !!!
            if num == target:       # error, target, not mid
                return True
            elif num < target:      # error, target, not mid
                # target position compare with mid or mid_expression???
                # target is on the right of [mid]
                left = mid + 1      # error: mid, not target
            else:
                right = mid - 1     # error: mid, not target

        return False

left, right 和 mid 有关 +- 1
```

Java

```
no BS from inter-web
The basic idea is from right corner, if the current number greater than target
col - 1 in same row, else if the current number less than target, row + 1 in
same column, finally if they are same, we find it, and return return.

  public boolean searchMatrix(int[][] matrix, int target) {
```

```
            int i = 0, j = matrix[0].length - 1;
            while (i < matrix.length && j >= 0) {
                    if (matrix[i][j] == target) {
                        return true;
                    } else if (matrix[i][j] > target) {
                        j--;
                    } else {
                        i++;
                    }
                }

            return false;
        }
```

```
public boolean searchMatrix(int[][] matrix, int target) {
    if (matrix == null || matrix.length == 0 || matrix[0].length == 0) {
        return false;
    }
    int row = 0;
    int col = matrix[0].length - 1;
    while (row < matrix.length && col >= 0) {
        if (matrix[row][col] == target) {
            return true;
        } else if (matrix[row][col] < target) {
            row++;
        } else {
            col--;
        }
    }
    return false;
}

// 1ms, beats 110.88%
```

33. 搜索旋转排序数组

难度中等799

假设按照升序排序的数组在预先未知的某个点上进行了旋转。

(例如，数组 `[0,1,2,4,5,6,7]` 可能变为 `[4,5,6,7,0,1,2]`)。

搜索一个给定的目标值，如果数组中存在这个目标值，则返回它的索引，否则返回 `-1` 。

你可以假设数组中不存在重复的元素。

你的算法时间复杂度必须是 *O*(log *n*) 级别。

示例 1:

```
输入: nums = [4,5,6,7,0,1,2], target = 0
输出: 4
```

示例 2:

```
输入: nums = [4,5,6,7,0,1,2], target = 3
输出: -1
```

二分查找

代码

Python

```python
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        # 题意：无重复元素，可能中间有断
        # 1 base: traverse all nums, if exists return idx; or return -1
        # time: O(N) not right

        # 2 binary-search
        # time: O(logN)

        # define left and right index
        left, right = 0, len(nums) - 1

        while left <= right:
            mid = (left + right) // 2
            if nums[mid] == target:
                return mid

            # first, 根据转折点位置分类讨论, compare wiht [left] ! ! !
            if nums[mid] >= nums[left]:
                # no duplicate, no =
                # [5,6,7,1,2]
                # 转折点 is on the right of [mid]，则[mid]左侧是单调递增
                if nums[left] <= target <= nums[mid]:
                    right = mid - 1
                else:
                    left = mid + 1
            else:
                # 转折点 is on the left of [mid]，则[mid]右侧是单调递增
                # tips：寻找单调递增的区域进行比较！！！
                # [6,7,1,2,3,4,5]
                if nums[mid] <= target <= nums[right]:
                    left = mid + 1
                else:
                    right = mid - 1

        # does not exist return -1
        return -1
```

StefanPochmann49906

Last Edit: October 25, 2018 3:51 AM

88.2K VIEWS

This very nice idea is from rantos22's solution who sadly only commented *"You are not expected to understand that :)"*, which I guess is the reason it's now it's hidden among the most downvoted solutions. I present an explanation and a more usual implementation.

Explanation

Let's say `nums` looks like this: [12, 13, 14, 15, 16, 17, 18, 19, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

Because it's not fully sorted, we can't do normal binary search. But here comes the trick:

- If target is let's say 14, then we adjust `nums` to this, where "inf" means infinity:
  [12, 13, 14, 15, 16, 17, 18, 19, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf]
- If target is let's say 7, then we adjust `nums` to this:
  [-inf, -inf, -inf, -inf, -inf, -inf, -inf, -inf, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

And then we can simply do ordinary binary search.

Of course we don't actually adjust the whole array but instead adjust only on the fly only the elements we look at. And the adjustment is done by comparing both the target and the actual element against nums[0].

Code

If `nums[mid]` and `target` are *"on the same side"* of `nums[0]`, we just take `nums[mid]`. Otherwise we use -infinity or +infinity as needed.

```
int search(vector<int>& nums, int target) {
    int lo = 0, hi = nums.size();
    while (lo < hi) {
        int mid = (lo + hi) / 2;

        double num = (nums[mid] < nums[0]) == (target < nums[0])
                   ? nums[mid]
                   : target < nums[0] ? -INFINITY : INFINITY;

        if (num < target)
            lo = mid + 1;
        else if (num > target)
            hi = mid;
        else
            return mid;
    }
    return -1;
}
```