A How does it work?

In its simplest form, Binary Search operates on a contiguous sequence with a specified left and right index. This is called the Search Space. Binary Search maintains the left, right, and middle indicies of the search space and compares the search target or applies the search condition to the middle value of the collection; if the condition is unsatisfied or values unequal, the half in which the target cannot lie is eliminated and the search continues on the remaining half until it is successful. If the search ends with an empty half, the condition cannot be fulfilled and target is not found.

In the following chapters, we will review how to identify Binary Search problems, reasons why we use Binary Search, and the 3 different Binary Search templates that you might be previously unaware of. Since Binary Search is a common interview topic, we will also categorize practice problems to different templates so you can practice using each.

Note:

Binary Search can take many alternate forms and might not always be as straight forward as searching for a specific value. Sometimes you will have to apply a specific condition or rule to determine which side (left or right) to search next.

We will provide examples in the coming chapters. First, could you try write a binary search algorithm yourself?



```
∄ 2 >_ ♦
 Python
  1 v class Solution(object):
  2 v def search(self, nums, target):
               :type nums: List[int]
               :type target: int
              :rtype: int
  6
         i = 0
j = len(nums) - 1
while i <= j:
    mid = (i + j)/2
    if target < nums[mid]:
        j = mid - 1
    elif target > nums[mid]:
        i = mid + 1
  9
 10
 11 *
 12
 13 v
 14
 15 *
 16
                     elif target == nums[mid]:
 17 v
 18
                         return mid
 19 return -1

    Run Code

                                                                                                                                       △ Submit Solution
☐ Custom Testcase ( Contribute ● )
```

A Identification and Template Introduction

How do we identify Binary Search?

As mentioned in earlier, Binary Search is an algorithm that divides the search space in 2 after every comparison. Binary Search should be considered every time you need to search for an index or element in a collection. If the collection is unordered, we can always sort it first before applying Binary Search.

3 Parts of a Successful Binary Search

Binary Search is generally composed of 3 main sections:

- 1. Pre-processing Sort if collection is unsorted.
- 2. Binary Search Using a loop or recursion to divide search space in half after each comparison.
- 3. Post-processing Determine viable candidates in the remaining space.

3 Templates for Binary Search

When we first learned Binary Search, we might struggle. We might study hundreds of Binary Search problems online and each time we looked at a developer's code, it seemed to be implemented slightly differently. Although each implementation divided the problem space in 1/2 at each step, one had numerous questions:

- · Why was it implemented slightly differently?
- · What was the developer thinking?
- · Which way was easier?
- · Which way is better?

After many failed attempts and lots of hair-pulling, we found 3 main templates for Binary Search. To prevent hair-pulling and to make it easier for new developers to learn and understand, we have provided them in the next chapter.

A Binary Search Template I

Template #1:

```
Copy Copy
       Java Python
       :rcype: Inc
      if len(nums) == 0:
          return -1
       left, right = 0, len(nums) - 1
10
11
      while left <= right:
          mid = (left + right) // 2
12
13
          if nums[mid] == target:
14
              return mid
15
         elif nums[mid] < target:
16
              left = mid + 1
17
18
              right = mid - 1
19
20
       # End Condition: left > right
```

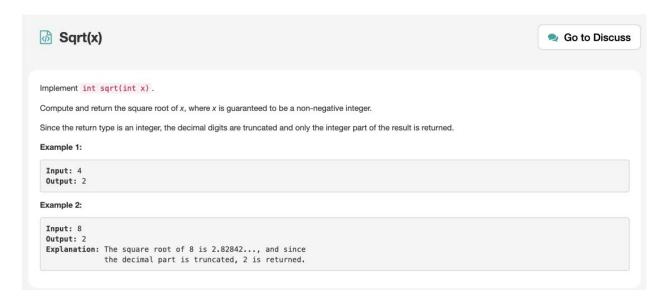
Template #1 is the most basic and elementary form of Binary Search. It is the standard Binary Search Template that most high schools or universities use when they first teach students computer science. Template #1 is used to search for an element or condition which can be determined by accessing a single index in the array.

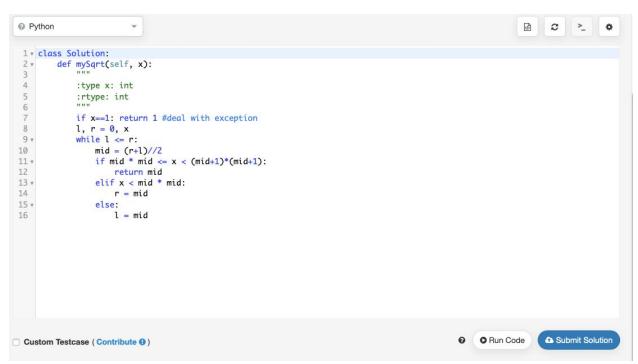
Key Attributes:

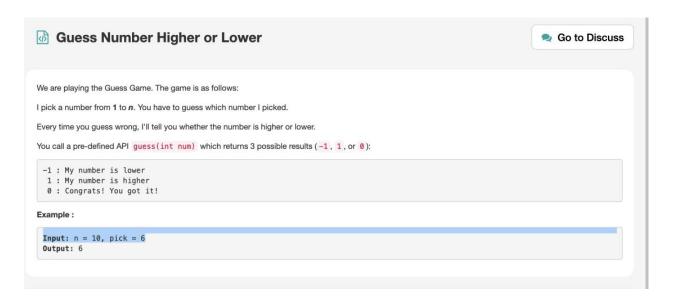
- · Most basic and elementary form of Binary Search
- · Search Condition can be determined without comparing to the element's neighbors (or use specific elements around it)
- · No post-processing required because at each step, you are checking to see if the element has been found. If you reach the end, then you know the element is not found

Distinguishing Syntax:

```
• Initial Condition: left = 0, right = length-1
• Termination: left > right
• Searching Left: right = mid-1
• Searching Right: left = mid+1
```









```
Suppose an array sorted in ascending order is rotated at some pivot unknown to you beforehand.

(i.e., [0,1,2,4,5,6,7] might become [4,5,6,7,0,1,2]).

You are given a target value to search. If found in the array return its index, otherwise return -1.

You may assume no duplicate exists in the array.

Your algorithm's runtime complexity must be in the order of O(log n).

Example 1:

Input: nums = [4,5,6,7,0,1,2], target = 0
Output: 4

Example 2:

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





A Binary Search Template II

Template #2:

```
Сору
            Python
      if len(nums) == 0:
8
          return -1
      left, right = 0, len(nums)
10
11
      while left < right:
          mid = (left + right) // 2
12
13
         if nums[mid] == target:
          elif nums[mid] < target:</pre>
15
             left = mid + 1
16
17
         else:
              right = mid
18
19
      # Post-processing:
20
21
       # End Condition: left == right
      if left != len(nums) and nums[left] == target:
23
           return left
24
      return -1
```

Template #2 is an advanced form of Binary Search. It is used to search for an element or condition which requires accessing the current index and its immediate right neighbor's index in the array.

Key Attributes:

- · An advanced way to implement Binary Search.
- · Search Condition needs to access element's immediate right neighbor
- · Use element's right neighbor to determine if condition is met and decide whether to go left or right
- Gurantees Search Space is at least 2 in size at each step
- Post-processing required. Loop/Recursion ends when you have 1 element left. Need to assess if the remaining element meets the condition.

Distinguishing Syntax:

```
• Initial Condition: left = 0, right = length
```

```
Termination: left == right
Searching Left: right = mid
Searching Right: left = mid+1
```

First Bad Version



You are a product manager and currently leading a team to develop a new product. Unfortunately, the latest version of your product fails the quality check. Since each version is developed based on the previous version, all the versions after a bad version are also bad.

Suppose you have n versions [1, 2, ..., n] and you want to find out the first bad one, which causes all the following ones to be bad.

You are given an API bool isBadVersion(version) which will return whether version is bad. Implement a function to find the first bad version. You should minimize the number of calls to the API.

Example:

```
Given n = 5, and version = 4 is the first bad version.

call isBadVersion(3) -> false
call isBadVersion(5) -> true
call isBadVersion(4) -> true

Then 4 is the first bad version.
```

```
₫ C >_ •
@ Python
 1 # The isBadVersion API is already defined for you.
 2 # @param version, an integer
 3 # @return a bool
 4 # def isBadVersion(version):
 6 v class Solution(object):
       def firstBadVersion(self, n):
 8
 9
           :type n: int
 10
           :rtype: int
 11
 12
           l,r=1,n
 13
           while l < r:
 14 *
              mid = 1 + (r - 1)/2
 15
             if isBadVersion(mid) == False:
 16 +
 17
                 l = mid + 1
            else:
 18 v
 19
                 r = mid
 20
 21
 22 *
         if isBadVersion(l):
 23
              return l
 24
 25
           return 1 + 1
                                                                                               Submit Solution

    Run Code
```

Find Peak Element

Go to Discuss

A peak element is an element that is greater than its neighbors.

Given an input array nums, where nums[i] ≠ nums[i+1], find a peak element and return its index.

The array may contain multiple peaks, in that case return the index to any one of the peaks is fine.

You may imagine that $nums[-1] = nums[n] = -\infty$.

Example 1:

```
Input: nums = [1,2,3,1]
Output: 2
Explanation: 3 is a peak element and your function should return the index number 2.
```

Example 2:

Note:

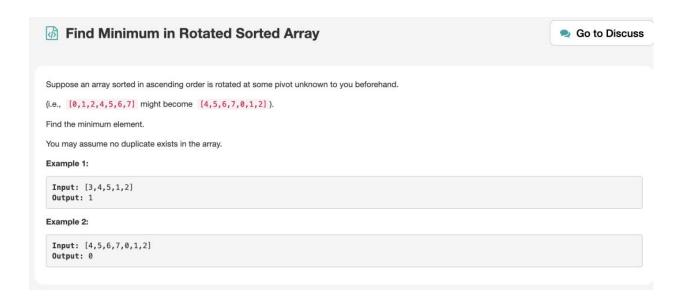
Your solution should be in logarithmic complexity.

```
₫ C >_ •
 Python
 4
            :type nums: List[int]
  5
            :rtype: int
  6
            ret=[]
  8 *
            if(len(nums)==1):
 9
                return 0
           if(len(nums)==2):
    return 0 if(nums[1]<=nums[0]) else 1</pre>
 10 v
 11
         for i in range(1,len(nums)-1):
    if(nums[i]>nums[i-1] and nums[i]>nums[i+1]):
 12 *
 13 *
 14
                    ret.append(i)
 15
                    break
         if(nums[0]>nums[1]):
 16 *
 17
                ret.append(0)
            if(nums[-1]>nums[-2]):
 18 v
                ret.append(len(nums)-1)
 19
            return ret[0]
 20

    Run Code

    Submit Solution

☐ Custom Testcase ( Contribute ● )
```



```
:type nums: List[int]
          l, r = 0, len(nums) - 1
while l < r:</pre>
 8 *
            if nums[l] < nums[r]:
    return nums[l]</pre>
 9 +
10
11
12
13 *
                m = l + (r - l) // 2
if nums[m] >= nums[l]:
 14
                   l = m + 1
 15 *
 16
                    r = m
 17
 18
            return nums[1]
 19
 20 v class Solution(object):
21 v
22
23
24
25
        def findMin(self, nums):
            :type nums: List[int]
            :rtype: int
          for i in range(1,len(nums)):
 26 *
 27 *
           if nums[i-1] > nums[i]:
 28
                   return nums[i]
 29
            return(nums[0])
30
                                                                                                            ♠ Submit Solution

    Run Code
```

A Binary Search Template III

Template #3:

```
Сору
C++
       Java Python
       :rcybe: Tur
 6
       if len(nums) == 0:
           return -1
10
     left, right = 0, len(nums) - 1
11
     while left + 1 < right:
           mid = (left + right) // 2
12
13
          if nums[mid] == target:
14
              return mid
15
         elif nums[mid] < target:
16
              left = mid
17
          else:
18
              right = mid
19
      # Post-processing:
20
21
       # End Condition: left + 1 == right
22
      if nums[left] == target: return left
23
       if nums[right] == target: return right
24
       return -1
```

Template #3 is another unique form of Binary Search. It is used to search for an element or condition which requires accessing the current index and its immediate left and right neighbor's index in the array.

Key Attributes:

- · An alternative way to implement Binary Search
- · Search Condition needs to access element's immediate left and right neighbors
- · Use element's neighbors to determine if condition is met and decide whether to go left or right
- · Gurantees Search Space is at least 3 in size at each step
- · Post-processing required. Loop/Recursion ends when you have 2 elements left. Need to assess if the remaining elements meet the condition.

Distinguishing Syntax:

```
• Initial Condition: left = 0, right = length-1
• Termination: left + 1 == right

    Searching Left: right = mid

    Searching Right: left = mid
```

```
Given an array of integers nums sorted in ascending order, find the starting and ending position of a given target value.

Your algorithm's runtime complexity must be in the order of O(\log n).

If the target is not found in the array, return [-1, -1].

Example 1:

Input: nums = [5,7,7,8,8,10], target = 8
Output: [3,4]

Example 2:

Input: nums = [5,7,7,8,8,10], target = 6
Output: [-1,-1]
```





Given a sorted array, two integers k and x, find the k closest elements to x in the array. The result should also be sorted in ascending order. If there is a tie, the smaller elements are always preferred.

Example 1:

```
Input: [1,2,3,4,5], k=4, x=3
Output: [1,2,3,4]
```

Example 2:

```
Input: [1,2,3,4,5], k=4, x=-1
Output: [1,2,3,4]
```

Note:

- 1. The value k is positive and will always be smaller than the length of the sorted array.
- 2. Length of the given array is positive and will not exceed 104
- 3. Absolute value of elements in the array and x will not exceed 10⁴

UPDATE (2017/9/19):

The arr parameter had been changed to an array of integers (instead of a list of integers). Please reload the code definition to get the latest changes.

3 different approaches in Python, with explanation

648 VIEWS



Created at: August 12, 2017 11:48 PM



5

First approach - Sorting by absolute difference and taking the first k elements. This approach does not scale well and only works because the input is small.

```
class Solution(object):
    def findClosestElements(self, arr, k, x):
        # Time: 0(n.lgn + k.lgk)
        # Space: 0(n + k)
        return sorted(sorted(arr, key=lambda val: abs(val - x))[:k])
```

Second approach - Two pointers approach but without any binary search performed on the original array. This approach does not make use of the sorted property of the input array. We use a **Counter** to track how many of each value in the array. The two pointers here essentially are the lower and upper values in the final result. We decrement the lower pointer and increment the upper pointer and add the count of the values into the result until the total length of the result exceeds k. This approach works for arrays where elements have a relatively small absolute difference.

```
class Solution(object):
   def findClosestElements(self, arr, k, x):
       # Time: O(n + k.lgk)
       # Space: 0(n + k)
       from collections import Counter
       c = Counter(arr)
       res = []
       lower, upper = x - 1, x
       min_val, max_val = min(arr), max(arr)
       if x in c:
           res.extend([x] * c[x])
           upper += 1
       while len(res) < k:
           while lower >= min_val and lower not in c:
                lower -= 1
           while upper <= max_val and upper not in c:
               upper += 1
            if abs(x - lower) \le abs(x - upper):
                res.extend([lower] * min(k - len(res), c[lower]))
                lower -= 1
                res.extend([upper] * min(k - len(res), c[upper]))
                upper += 1
        return sorted(res)[:k]
```

Third approach - Use binary search to find the value's position in the array, and expand outwards using two pointers. This approach scales the best.

```
class Solution(object):
    def findClosestElements(self, arr, k, x):
        # Time: O(lgn + k)
        # Space: O(k)
        import bisect
        index = min(bisect.bisect_left(arr, x), len(arr) - 1)
        left, right = index - 1, index + 1 if arr[index] == x else index
        while right - left < k + 1:
            if left >= 0 and (right >= len(arr) or abs(arr[left] - x) <= abs(arr[right] - x)):
            left -= 1
        else:
            right += 1
        return arr[left + 1:right]</pre>
```

Find Peak Element

Go to Discuss

A peak element is an element that is greater than its neighbors.

Given an input array nums, where nums [i] ≠ nums [i+1], find a peak element and return its index.

The array may contain multiple peaks, in that case return the index to any one of the peaks is fine.

You may imagine that $nums[-1] = nums[n] = -\infty$.

Example 1:

```
Input: nums = [1,2,3,1]
Output: 2
Explanation: 3 is a peak element and your function should return the index number 2.
```

Example 2:

Note:

Your solution should be in logarithmic complexity.

Find Peak Element

Go to Discuss

A peak element is an element that is greater than its neighbors.

Given an input array nums, where $nums[i] \neq nums[i+1]$, find a peak element and return its index.

The array may contain multiple peaks, in that case return the index to any one of the peaks is fine.

You may imagine that $nums[-1] = nums[n] = -\infty$.

Example 1:

```
Input: nums = [1,2,3,1]
Output: 2
Explanation: 3 is a peak element and your function should return the index number 2.
```

Example 2:

Note

Your solution should be in logarithmic complexity.

```
⊕ C >_ •
Python
 1 v class Solution(object):
        def findPeakElement(self, nums):
 3
             :type nums: List[int]
 4
 5
             :rtype: int
 6
        lo, hi = 0, len(nums) - 1
while lo < hi:</pre>
 8 +
 9
            mid = lo + hi >> 1
            if nums[mid] > nums[mid+1]: hi = mid
else: lo = mid + 1
10
11
12
       return lo
13
14
15 """Assume there is a peak, then peak's left will be monotonic increasing while its right will be monotonic decreasing.
16
17 So during binary search,
18 if nums[mid] > nums[mid+1], we know mid is on the right side of peak, and we should left shift our hi,
19 otherwise we should right shift our lo.
20 When lo == hi, we find our peak index.
21
22 Since we are comparing nums[mid] and nums[mid+1],
23 we should update as lo = mid + 1 and hi = mid.
Because if nums[mid+1] is the peak, we right shift lo to mid+1; Otherwise nums[mid] is the peak, we left shift hi to mid)
28 We can safely use mid+1 as long as we initialized hi as len(nums)-1.
29 As the maximum value of mid is hi-1, otherwise lo <= hi and loop breaks, mid+1
30 will not exceed len(nums)-1.
```

Template Explanation:

99% of binary search problems that you see online will fall into 1 of these 3 templates. Some problems can be implemented using multiple templates, but as you practice more, you will notice that some templates are more suited for certain problems than others.

Note: The templates and their differences have been colored coded below.

```
Template #2:
                                                               Template #3:
Template #1:
                                                               // Pre-processing
// Pre-processing
                               // Pre-processing
left = 0; right = length - 1;
                               left = 0; right = length;
                                                               left = 0; right = length - 1;
while (left <= right) {
                                while (left < right) {
                                                               while (left + 1 < right) {
mid = left + (right - left) / 2;
                                mid = left + (right - left) / 2;
                                                                mid = left + (right - left) / 2;
if (nums[mid] == target) {
                                if(nums[mid] < target) {
                                                                if (num[mid] < target) {
 return mid;
                                  left = mid + 1;
                                                                 left = mid;
} else if(nums[mid] < target) { } else {
                                                               } else {
  left = mid + 1;
                                  right = mid;
                                                                 right = mid;
} else
 right = mid - 1;
}
                               // left == right
                                                               // left + 1 == right
// right + 1 == left
                               // 1 more candidate
                                                               // 2 more candidates
// No more candidate
                                // Post-Processing
                                                               // Post-Processing
```

These 3 templates differ by their:

- · left, mid, right index assignments
- · loop or recursive termination condition
- · necessity of post-processing

Template 1 and 3 are the most commonly used and almost all binary search problems can be easily implemented in one of them. Template 2 is a bit more advanced and used for certain types of problems.

Each of these 3 provided templates provide a specific use case:

Template #1 (left <= right):

- · Most basic and elementary form of Binary Search
- Search Condition can be determined without comparing to the element's neighbors (or use specific elements around it)
- No post-processing required because at each step, you are checking to see if the element has been found. If you reach the end, then you know the element is not found

Template #2 (left < right):

- An advanced way to implement Binary Search.
- · Search Condition needs to access element's immediate right neighbor
- · Use element's right neighbor to determine if condition is met and decide whether to go left or right
- · Gurantees Search Space is at least 2 in size at each step
- Post-processing required. Loop/Recursion ends when you have 1 element left. Need to assess if the remaining element meets the condition.

Template #3 (left + 1 < right):

- · An alternative way to implement Binary Search
- · Search Condition needs to access element's immediate left and right neighbors
- · Use element's neighbors to determine if condition is met and decide whether to go left or right
- · Gurantees Search Space is at least 3 in size at each step
- Post-processing required. Loop/Recursion ends when you have 2 elements left. Need to assess if the remaining elements meet the condition.

Given a non-empty binary search tree and a target value, find the value in the BST that is closest to the target. Note: Given target value is a floating point. You are guaranteed to have only one unique value in the BST that is closest to the target. Example: Input: root = [4,2,5,1,3], target = 3.714286 4 / \ 2 5 / \ 1 3 Output: 4

```
# Definition for a binary tree node.

# class TreeNode(object):

# def __init__(self, x):

# self.val = x

# self.left = None

# self.right = None

# class Solution(object):

# def closestValue(self, root, target):

# r = root.val

while root:

# if abs(root.val - target) < abs(r - target):

# return r
```



Search in a Sorted Array of Unknown Size

Given an integer array sorted in ascending order, write a function to search target in nums. If target exists, the array size is unknown to you. You may only access the array using an ArrayReader interface, where Arra index k (0-indexed).

You may assume all integers in the array are less than 10000, and if you access the array out of bounds, Arrayl

Example 1:

```
Input: array = [-1,0,3,5,9,12], target = 9
Output: 4
Explanation: 9 exists in nums and its index is 4
```

Example 2:

```
Input: array = [-1,0,3,5,9,12], target = 2
Output: -1
Explanation: 2 does not exist in nums so return -1
```

Note:

Go to Discuss

Given an integer array sorted in ascending order, write a function to search target in nums. If target exists, then return its index, otherwise return -1. However, the array size is unknown to you. You may only access the array using an ArrayReader interface, where ArrayReader.get(k) returns the element of the array at index k (0-indexed).

You may assume all integers in the array are less than 10000, and if you access the array out of bounds, ArrayReader.get will return 2147483647.

Example 1:

```
Input: array = [-1,0,3,5,9,12], target = 9
Output: 4
Explanation: 9 exists in nums and its index is 4
```

Example 2:

```
Input: array = [-1,0,3,5,9,12], target = 2
Output: -1
Explanation: 2 does not exist in nums so return -1
```

Note:

- 1. You may assume that all elements in the array are unique.
- 2. The value of each element in the array will be in the range [-9999, 9999] .

```
Python
                                                                                                            ₫ C >_ •
  1 v class Solution(object):
         def search(self, reader, target):
 4
             :type reader: ArrayReader
  5
             :type target: int
             :rtype: int
  6
  8
             1 = 1
 10 v
             while reader.get(l-1) < target:
 11
 13 +
            if reader.get(l-1) == target:
 14
                 return 1-1
 15
             r = 1 - 1
 16
 17
             1 = 1 // 2 - 1
 18
             while l < r:
 19 +
                 mid = (l + r) // 2
 20
 21 *
                 if reader.get(mid) > target:
 22
                     r = mid
 23 v
                 elif reader.get(mid) < target:</pre>
 24
                     l = mid + 1
 25 v
                 else:
 26
                     return mid
 27
             return -1 if reader.get(l) != target else l

    Run Code

                                                                                                                 ♠ Submit Solution
□ Custom Testcase ( Contribute ● )
```



That's even shorter than the other more obvious "cheat":

```
class Solution:
   def myPow(self, x, n):
     return x ** n
```

And to calm down the haters, here's me "doing it myself":

Recursive:

```
class Solution:
    def myPow(self, x, n):
        if not n:
            return 1
        if n < 0:
            return 1 / self.myPow(x, -n)
        if n % 2:
            return x * self.myPow(x, n-1)
        return self.myPow(x*x, n/2)</pre>
```

Iterative:

```
class Solution:
    def myPow(self, x, n):
        if n < 0:
            x = 1 / x
            n = -n
        pow = 1
        while n:
            if n & 1:
                pow *= x
            x *= x
            n >>= 1
        return pow
```

Walid Perfect Square

Go to Discuss

Given a positive integer *num*, write a function which returns True if *num* is a perfect square else False.

Note: Do not use any built-in library function such as $\ensuremath{\mathsf{sqrt}}$.

Example 1:

```
Input: 16
Output: true
```

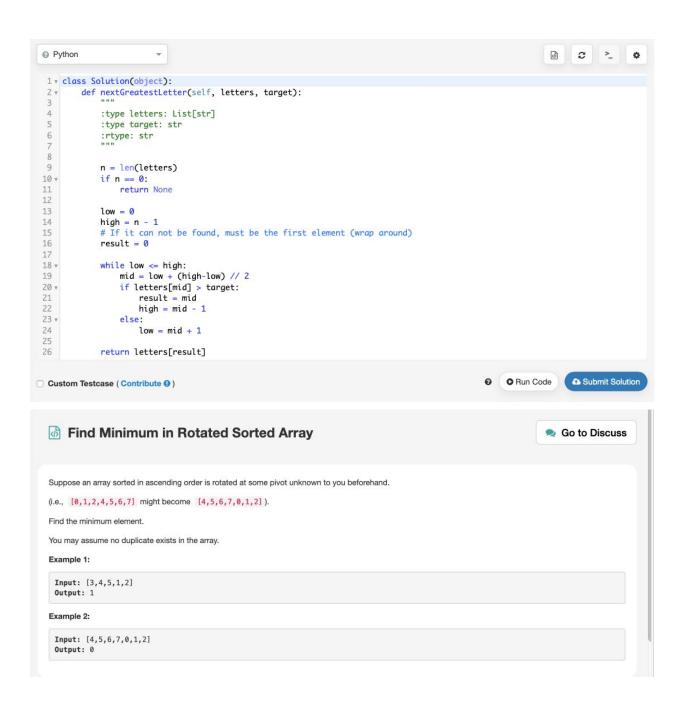
Example 2:

```
Input: 14
Output: false
```

```
⊕ 2 >_ •
 1 v class Solution(object):
 def isPerfectSquare(self, num):
    b, e = 1. (num >> 1) + 1
           def isPerfectSquare(self, num,
    b, e = 1, (num >> 1) + 1
    while b <= e:
        mid = (b + e) >> 1
        sq = mid * mid
        if sq == num:
            return True
        if sq > num:
            e = mid - 1
        else:
            b = mid + 1
        return False
 4 +
 5
 7 *
 9 *
11 v
12
13
Find Smallest Letter Greater Than Target
                                                                                                                                                                  Go to Discuss
Given a list of sorted characters letters containing only lowercase letters, and given a target letter target, find the smallest element in the list that is larger than
Letters also wrap around. For example, if the target is target = 'z' and letters = ['a', 'b'], the answer is 'a'.
Examples:
  letters = ["c", "f", "j"]
  target = "a"
Output: "c"
 letters = ["c", "f", "j"]
target = "c"
Output: "f"
  Input:
  Input:
  letters = ["c", "f", "j"]
 target = "d"
Output: "f"
  Input:
  letters = ["c", "f", "j"]
  target = "g"
  Output: "j"
  Input:
```

letters = ["c", "f", "j"]

target = "j"
Output: "c"



Go to Discuss

Suppose an array sorted in ascending order is rotated at some pivot unknown to you beforehand.

(i.e., [0,1,2,4,5,6,7] might become [4,5,6,7,0,1,2]).

Find the minimum element.

The array may contain duplicates.

Example 1:

```
Input: [1,3,5]
Output: 1
```

Example 2:

```
Input: [2,2,2,0,1]
Output: 0
```

Note:

- This is a follow up problem to Find Minimum in Rotated Sorted Array.
- Would allow duplicates affect the run-time complexity? How and why?

Go to Discuss

Given two arrays, write a function to compute their intersection.

Example 1:

```
Input: nums1 = [1,2,2,1], nums2 = [2,2]
Output: [2]
```

Example 2:

```
Input: nums1 = [4,9,5], nums2 = [9,4,9,8,4]
Output: [9,4]
```

Note:

- · Each element in the result must be unique.
- The result can be in any order.

```
class Solution(object):
    def intersection(self, nums1, nums2):
        """
            :type nums1: List[int]
            :type nums2: List[int]
            :rtype: List[int]
            """
            nums1=set(nums1)
            nums2=set(nums2)
            return list(nums1&nums2)

stom Testcase (Contribute ①)
```

Solution 1:

use set operation in python, one-line solution.

```
class Solution(object):
def intersection(self, nums1, nums2):
    """
    :type nums1: List[int]
    :type nums2: List[int]
    :rtype: List[int]
    """
    return list(set(nums1) & set(nums2))
```

Solution 2:

brute-force searching, search each element of the first list in the second list. (to be more efficient, you can sort the second list and use binary search to accelerate)

```
class Solution(object):
    def intersection(self, nums1, nums2):
        """
        :type nums1: List[int]
        :type nums2: List[int]
        :rtype: List[int]
        """
        res = []
        for i in nums1:
            if i not in res and i in nums2:
                 res.append(i)
        return res
```



Given two arrays, write a function to compute their intersection.

Example 1:

```
Input: nums1 = [1,2,2,1], nums2 = [2,2]
Output: [2,2]
```

Example 2:

```
Input: nums1 = [4,9,5], nums2 = [9,4,9,8,4]
Output: [4,9]
```

Note:

- Each element in the result should appear as many times as it shows in both arrays.
- · The result can be in any order.

Follow up:

- What if the given array is already sorted? How would you optimize your algorithm?
- What if nums1's size is small compared to nums2's size? Which algorithm is better?
- What if elements of nums2 are stored on disk, and the memory is limited such that you cannot load all elements into the memory at once?

Algorithm

- · Use a set to store all numbers from nu1.
- . Then test membership of num2 in this set. Keep results in another set so that they are automatically deduped.

```
₫ C >_ •
 Python
  1 v class Solution(object):
        def twoSum(self, n, target):
            :type numbers: List[int]
:type target: int
  5
            :rtype: List[int]
  6
           p1 = 0
p2 = len(n)-1
  8
 9
 10
           while p1 < p2:
 11 v
                if n[p1] + n[p2] == target:
 12 v
 13
                   return [p1+1, p2+1]
                elif (target - n[p1]) > n[p2]:
 14 *
 15
                   p1 += 1
                else:
 16 *
                   p2 -= 1
 18

    Run Code

☐ Custom Testcase ( Contribute ● )
```



Given an array nums containing n+1 integers where each integer is between 1 and n (inclusive), prove that at least one duplicate number must exist. Assume that there is only one duplicate number, find the duplicate one.

Example 1:

```
Input: [1,3,4,2,2]
Output: 2
```

Example 2:

```
Input: [3,1,3,4,2]
Output: 3
```

Note:

- 1. You must not modify the array (assume the array is read only).
- 2. You must use only constant, O(1) extra space.
- 3. Your runtime complexity should be less than $O(n^2)$.
- 4. There is only one duplicate number in the array, but it could be repeated more than once.

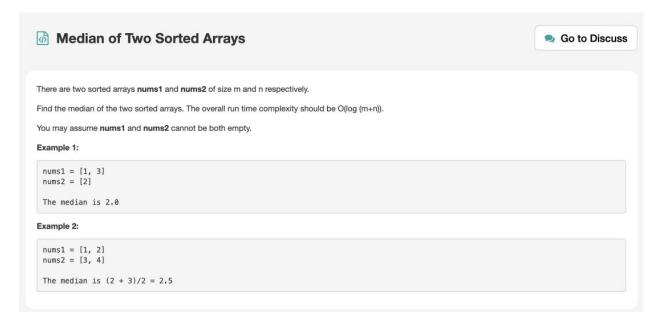
```
1 v class Solution(object):
        def findDuplicate(self, nums):
 3
            :type nums: List[int]
 4
  5
           :rtype: int
 6
         seen = set()
for num in nums:
  8 v
              if num in seen:
 10
                   return num
               seen.add(num)
 11
 12
         return -1

    Run Code
    Submit Solution

Custom Testcase (Contribute (1)
```







Solution 1

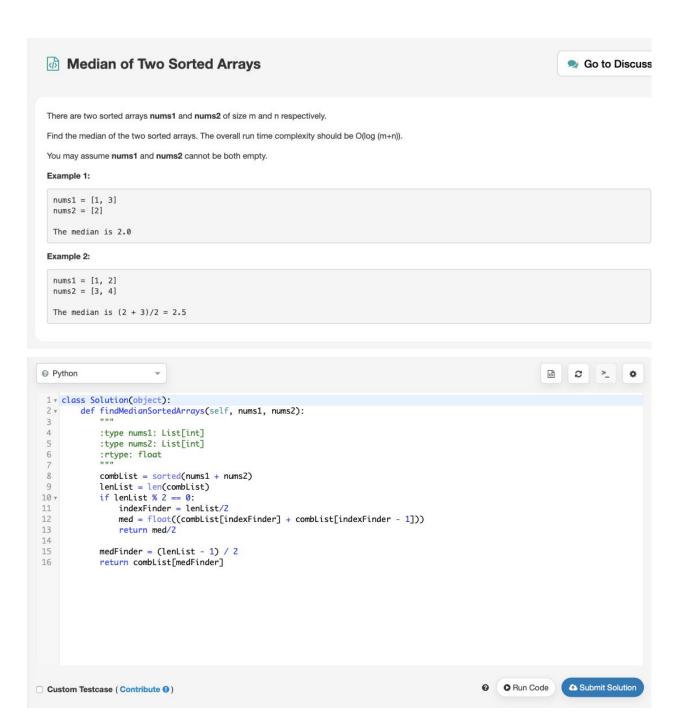
This one is a bit of a hack, as **bisect_left** is intended for lists, but apparently it's happy enough with something that behaves enough like a list (supports **__getitem__**):

```
def findMedianSortedArrays(self, nums1, nums2):
    a, b = sorted((nums1, nums2), key=len)
    m, n = len(a), len(b)
    after = (m + n - 1) / 2
    class Range:
        def __getitem__(self, i):
            return after-i-1 < 0 or a[i] >= b[after-i-1]
    i = bisect.bisect_left(Range(), True, 0, m)
    nextfew = sorted(a[i:i+2] + b[after-i:after-i+2])
    return (nextfew[0] + nextfew[1 - (m+n)%2]) / 2.0
```

Solution 2

Same, just with a self-made binary search:

```
def findMedianSortedArrays(self, nums1, nums2):
    a, b = sorted((nums1, nums2), key=len)
    m, n = len(a), len(b)
    after = (m + n - 1) / 2
    lo, hi = 0, m
    while lo < hi:
        i = (lo + hi) / 2
        if after-i-1 < 0 or a[i] >= b[after-i-1]:
            hi = i
        else:
            lo = i + 1
    i = lo
    nextfew = sorted(a[i:i+2] + b[after-i:after-i+2])
    return (nextfew[0] + nextfew[1 - (m+n)%2]) / 2.0
```



```
Find K-th Smallest Pair Distance
                                                                                                                        Go to Discuss
Given an integer array, return the k-th smallest distance among all the pairs. The distance of a pair (A, B) is defined as the absolute difference between A and B.
Example 1:
 Input:
 nums = [1,3,1]
 k = 1
 Output: 0
 Explanation:
 Here are all the pairs:
 (1,3) \rightarrow 2
 (1,1) -> 0
 (3,1) \rightarrow 2
 Then the 1st smallest distance pair is (1,1), and its distance is 0.
Note:
 1. 2 <= len(nums) <= 10000.
 2. 0 <= nums[i] < 1000000.
  3. 1 \le k \le len(nums) * (len(nums) - 1) / 2.
```





Go to Discuss

Given an array which consists of non-negative integers and an integer m, you can split the array into m non-empty continuous subarrays. Write an algorithm to minimize the largest sum among these m subarrays.

Note:

If n is the length of array, assume the following constraints are satisfied:

- $1 \le n \le 1000$
- $1 \le m \le \min(50, n)$

Examples:

```
Input:
nums = [7,2,5,10,8]
m = 2

Output:
18

Explanation:
There are four ways to split nums into two subarrays.
The best way is to split it into [7,2,5] and [10,8],
where the largest sum among the two subarrays is only 18.
```

```
B 2 ≥ •
 Python
 1 v class Solution(object):
        def splitArray(self, nums, m):
           :type nums: List[int]
 5
           :type m: int
          :rtype: int
 6
 8 v
         def valid(mid):
          cnt = 0
               current = 0
 10
              for n in nums:
 11 v
 12
                  current += n
 13 v
                   if current>mid:
 14
                      cnt += 1
                      if cnt>=m:
 15 v
 16
                          return False
 17
                      current = n
               return True
 19
           l = max(nums)
 20
 21
           h = sum(nums)
 22
 23 v
           while l<h:
               mid = 1+(h-1)/2
 24
 25 ₹
               if valid(mid):
 26
                   h = mid
 28
                  l = mid+1
 29
            return l
☐ Custom Testcase (Contribute ④)

    Run Code
```