## 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?

# Binary Search

Go to Discuss

Given a **sorted** (in ascending order) integer array `nums` of `n` elements and a `target` value, write a function to search `target` in `nums`. If `target` exists, then return its index, otherwise return `-1`.

**Example 1:**

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

**Example 2:**

```
Input: nums = [-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 `nums` are unique.
2. `n` will be in the range `[1, 10000]`.
3. The value of each element in `nums` will be in the range `[-9999, 9999]`.

Python

```python
class Solution(object):
    def search(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: int
        """

        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
            elif target == nums[mid]:
                return mid
        return -1
```

Custom Testcase ( Contribute )

Run Code Submit Solution

# 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.

# Binary Search Template I

**Template #1:**

C++ Java Python Copy

```python
    :rtype: int
    """
    if len(nums) == 0:
        return -1

    left, right = 0, len(nums) - 1
    while left <= right:
        mid = (left + right) // 2
        if nums[mid] == target:
            return mid
        elif nums[mid] < target:
            left = mid + 1
        else:
            right = mid - 1

    # End Condition: left > right
    return -1
```

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`

# Sqrt(x)

Go to Discuss

Implement `int sqrt(int x)`.

Compute and return the square root of *x*, where *x* is guaranteed to be a non-negative integer.

Since the return type is an integer, the decimal digits are truncated and only the integer part of the result is returned.

**Example 1:**

```
Input: 4
Output: 2
```

**Example 2:**

```
Input: 8
Output: 2
Explanation: The square root of 8 is 2.82842..., and since
             the decimal part is truncated, 2 is returned.
```

Python

```python
class Solution:
    def mySqrt(self, x):
        """
        :type x: int
        :rtype: int
        """
        if x==1: return 1 #deal with exception
        l, r = 0, x
        while l <= r:
            mid = (r+l)//2
            if mid * mid <= x < (mid+1)*(mid+1):
                return mid
            elif x < mid * mid:
                r = mid
            else:
                l = mid
```

Custom Testcase ( Contribute )

Run Code Submit Solution

# Guess Number Higher or Lower

Go to Discuss

We are playing the Guess Game. The game is as follows:

I pick a number from **1** to ***n***. You have to guess which number I picked.

Every time you guess wrong, I'll tell you whether the number is higher or lower.

You call a pre-defined API `guess(int num)` which returns 3 possible results (`-1`, `1`, or `0`):

```
-1 : My number is lower
 1 : My number is higher
 0 : Congrats! You got it!
```

**Example :**

```
Input: n = 10, pick = 6
Output: 6
```

Python

```python
# The guess API is already defined for you.
# @param num, your guess
# @return -1 if my number is lower, 1 if my number is higher, otherwise return 0
# def guess(num):

class Solution(object):
    def guessNumber(self, n):
        """
        :type n: int
        :rtype: int
        """
        low = 0
        high = n
        while low <= high:
            mid = int((high+low)/2)
            if guess(mid) == 1:
                low = mid+1
            elif guess(mid) == -1:
                high = mid-1
            else:
                break # found the right number.
        return mid
```

Custom Testcase ( Contribute )

Run Code

Submit Solution

# Search in Rotated Sorted Array

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]`).

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
```

Python

```python
class Solution(object):
    def search(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: int
        """
        i, j = 0, len(nums) - 1
        while i<=j:
            mid = (i+j)//2
            # print i, j
            if nums[mid] == target:
                return mid
            elif nums[i] <= nums[mid]:
                if nums[i] <= target < nums[mid]:
                    j = mid - 1
                else:
                    i = mid + 1
            else:
                if nums[mid] < target <= nums[j]:
                    i = mid + 1
                else:
                    j = mid - 1
        return -1
```

Custom Testcase ( Contribute )

Run Code Submit Solution

# Binary Search Template II

**Template #2:**

C++ Java Python Copy

```python
    if len(nums) == 0:
        return -1

    left, right = 0, len(nums)
    while left < right:
        mid = (left + right) // 2
        if nums[mid] == target:
            return mid
        elif nums[mid] < target:
            left = mid + 1
        else:
            right = mid

    # Post-processing:
    # End Condition: left == right
    if left != len(nums) and nums[left] == target:
        return left
    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

Go to Discuss

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.
```

Python

```python
# The isBadVersion API is already defined for you.
# @param version, an integer
# @return a bool
# def isBadVersion(version):

class Solution(object):
    def firstBadVersion(self, n):
        """
        :type n: int
        :rtype: int
        """
        l,r = 1, n

        while l < r:
            mid = l + (r - l)/2
            if isBadVersion(mid) == False:
                l = mid + 1
            else:
                r = mid


        if isBadVersion(l):
            return l

        return l + 1
```

Custom Testcase ( Contribute )

Run Code Submit Solution

# 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] = -∞`.

**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:**

```
Input: nums = [1,2,1,3,5,6,4]
Output: 1 or 5
Explanation: Your function can return either index number 1 where the peak element is 2,
             or index number 5 where the peak element is 6.
```

**Note:**

Your solution should be in logarithmic complexity.

Python

```python
class Solution(object):
    def findPeakElement(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        ret=[]
        if(len(nums)==1):
            return 0
        if(len(nums)==2):
            return 0 if(nums[1]<=nums[0]) else 1
        for i in range(1,len(nums)-1):
            if(nums[i]>nums[i-1] and nums[i]>nums[i+1]):
                ret.append(i)
                break
        if(nums[0]>nums[1]):
            ret.append(0)
        if(nums[-1]>nums[-2]):
            ret.append(len(nums)-1)
        return ret[0]
```

Custom Testcase ( Contribute ) Run Code Submit Solution

## Find Minimum in Rotated Sorted Array

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.

You may assume no duplicate exists in the array.

**Example 1:**

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

**Example 2:**

```
Input: [4,5,6,7,0,1,2]
Output: 0
```

```python
class Solution(object):
    def findMin(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        l, r = 0, len(nums) - 1
        while l < r:
            if nums[l] < nums[r]:
                return nums[l]

            m = l + (r - l) // 2
            if nums[m] >= nums[l]:
                l = m + 1
            else:
                r = m

        return nums[l]

class Solution(object):
    def findMin(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        for i in range(1,len(nums)):
            if nums[i-1] > nums[i]:
                return nums[i]
        return(nums[0])
```

Custom Testcase ( Contribute ) Run Code Submit Solution

# Binary Search Template III

**Template #3:**

C++ | Java | Python

Copy

```python
    :rtype: int
    """
    if len(nums) == 0:
        return -1

    left, right = 0, len(nums) - 1
    while left + 1 < right:
        mid = (left + right) // 2
        if nums[mid] == target:
            return mid
        elif nums[mid] < target:
            left = mid
        else:
            right = mid

    # Post-processing:
    # End Condition: left + 1 == right
    if nums[left] == target: return left
    if nums[right] == target: return right
    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`

# Search for a Range

Go to Discuss

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]
```

Python

```python
class Solution(object):
        ##First, bisect is faster than hand-written binary search.
        ##Also, if we didn't find the element in the first binary search, then there is no need to do the second binary search.

    def searchRange(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: List[int]
        """
        import bisect
        lidx = bisect.bisect_left(nums, target)
        lidx = -1 if lidx == len(nums) or nums[lidx] != target else lidx
        if lidx != -1:
            ridx = bisect.bisect_right(nums, target, lo=lidx)
            ridx = -1 if ridx == 0 or nums[ridx - 1] != target else ridx - 1
        else:
            ridx = -1
        return [lidx, ridx]
```

Custom Testcase ( Contribute )

Run Code  Submit Solution

# Find K Closest Elements

Go to Discuss

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 $10^4$
3. Absolute value of elements in the array and x will not exceed $10^4$

---

**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



5

Created at: August 12, 2017 11:48 PM

yangshun 985

**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: O(n.lgn + k.lgk)
        # Space: O(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: O(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) <= abs(x - upper):
                res.extend([lower] * min(k - len(res), c[lower]))
                lower -= 1
            else:
                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]
```

## 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] = -∞`.

**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:**

```
Input: nums = [1,2,1,3,5,6,4]
Output: 1 or 5
Explanation: Your function can return either index number 1 where the peak element is 2,
             or index number 5 where the peak element is 6.
```

**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] ≠ 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] = -∞`.

**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:**

```
Input: nums = [1,2,1,3,5,6,4]
Output: 1 or 5
Explanation: Your function can return either index number 1 where the peak element is 2,
             or index number 5 where the peak element is 6.
```

**Note:**

Your solution should be in logarithmic complexity.

Python

```python
class Solution(object):
    def findPeakElement(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
    lo, hi = 0, len(nums) - 1
    while lo < hi:
        mid = lo + hi >> 1
        if nums[mid] > nums[mid+1]: hi = mid
        else: lo = mid + 1
    return lo


"""Assume there is a peak, then peak's left will be monotonic increasing while its right will be monotonic decreasing.

So during binary search,
if nums[mid] > nums[mid+1], we know mid is on the right side of peak, and we should left shift our hi,
otherwise we should right shift our lo.
When lo == hi, we find our peak index.

Since we are comparing nums[mid] and nums[mid+1],
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)

We can safely use mid+1 as long as we initialized hi as len(nums)-1.
As the maximum value of mid is hi-1, otherwise lo <= hi and loop breaks, mid+1
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 #1: | Template #2: | Template #3: |
| --- | --- | --- |
| // 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.

# Closest Binary Search Tree Value

Go to Discuss

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
```

Python

```python
# 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):
                r = root.val
            root = root.left if target < root.val else root.right
        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 `Arr`
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, `Array`

**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:**

# Search in a Sorted Array of Unknown Size

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

```python
class Solution(object):
    def search(self, reader, target):
        """
        :type reader: ArrayReader
        :type target: int
        :rtype: int
        """
        l = 1

        while reader.get(l-1) < target:
            l *= 2

        if reader.get(l-1) == target:
            return l-1

        r = l - 1
        l = l // 2 - 1

        while l < r:
            mid = (l + r) // 2
            if reader.get(mid) > target:
                r = mid
            elif reader.get(mid) < target:
                l = mid + 1
            else:
                return mid

        return -1 if reader.get(l) != target else l
```

Custom Testcase ( Contribute )

Run Code

Submit Solution

# Pow(x, n)

Go to Discuss

Implement pow(*x*, *n*), which calculates *x* raised to the power *n* ($x^n$).

**Example 1:**

```
Input: 2.00000, 10
Output: 1024.00000
```

**Example 2:**

```
Input: 2.10000, 3
Output: 9.26100
```

**Example 3:**

```
Input: 2.00000, -2
Output: 0.25000
Explanation: 2^-2 = 1/2^2 = 1/4 = 0.25
```

**Note:**

- $-100.0 < x < 100.0$
- $n$ is a 32-bit signed integer, within the range $[-2^{31}, 2^{31} - 1]$

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)
```

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
```

## Valid 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 `sqrt`.

**Example 1:**

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

**Example 2:**

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

Python

```python
class Solution(object):
    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
```

## 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 the given target.

Letters also wrap around. For example, if the target is `target = 'z'` and `letters = ['a', 'b']`, the answer is `'a'`.

**Examples:**

```
Input:
letters = ["c", "f", "j"]
target = "a"
Output: "c"

Input:
letters = ["c", "f", "j"]
target = "c"
Output: "f"

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"
```

Python

```python
class Solution(object):
    def nextGreatestLetter(self, letters, target):
        """
        :type letters: List[str]
        :type target: str
        :rtype: str
        """

        n = len(letters)
        if n == 0:
            return None

        low = 0
        high = n - 1
        # If it can not be found, must be the first element (wrap around)
        result = 0

        while low <= high:
            mid = low + (high-low) // 2
            if letters[mid] > target:
                result = mid
                high = mid - 1
            else:
                low = mid + 1

        return letters[result]
```

Custom Testcase ( Contribute ) Run Code Submit Solution

## Find Minimum in Rotated Sorted Array

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.

You may assume no duplicate exists in the array.

**Example 1:**

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

**Example 2:**

```
Input: [4,5,6,7,0,1,2]
Output: 0
```

## Find Minimum in Rotated Sorted Array II

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?

## Intersection of Two Arrays

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 ) Run Code Submit Solution

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
```

# Intersection of Two Arrays II

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: [2,2]
```

**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.

```
class Solution(object):
    def intersection(self, nums1, nums2):
        """
        :type nums1: List[int]
        :type nums2: List[int]
        :rtype: List[int]
        """
        nums1_set = set(nums1)
        result_set = set([])
        for x in nums2:
            if x in nums1_set and x not in result_set:
                result_set.add(x)
        return [x for x in result_set]
```

Python

```
class Solution(object):
    def twoSum(self, n, target):
        """
        :type numbers: List[int]
        :type target: int
        :rtype: List[int]
        """
        p1 = 0
        p2 = len(n)-1

        while p1 < p2:
            if n[p1] + n[p2] == target:
                return [p1+1, p2+1]
            elif (target - n[p1]) > n[p2]:
                p1 += 1
            else:
                p2 -= 1
```

**Custom Testcase** ( Contribute ) Run Code Submit Solution

# Find the Duplicate Number

Go to Discuss

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.

```python
class Solution(object):
    def findDuplicate(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        seen = set()
        for num in nums:
            if num in seen:
                return num
            seen.add(num)
        return -1
```

Custom Testcase ( Contribute )

Run Code

Submit Solution

## Median of Two Sorted Arrays

Go to Discuss

There are two sorted arrays **nums1** and **nums2** of size m and n respectively.

Find the median of the two sorted arrays. The overall run time complexity should be O(log (m+n)).

You may assume **nums1** and **nums2** cannot be both empty.

**Example 1:**

```
nums1 = [1, 3]
nums2 = [2]

The median is 2.0
```

**Example 2:**

```
nums1 = [1, 2]
nums2 = [3, 4]

The median is (2 + 3)/2 = 2.5
```

### 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
```

# Median of Two Sorted Arrays

Go to Discuss

There are two sorted arrays **nums1** and **nums2** of size m and n respectively.

Find the median of the two sorted arrays. The overall run time complexity should be O(log (m+n)).

You may assume **nums1** and **nums2** cannot be both empty.

**Example 1:**

```
nums1 = [1, 3]
nums2 = [2]

The median is 2.0
```

**Example 2:**

```
nums1 = [1, 2]
nums2 = [3, 4]

The median is (2 + 3)/2 = 2.5
```

Python

```
class Solution(object):
    def findMedianSortedArrays(self, nums1, nums2):
        """
        :type nums1: List[int]
        :type nums2: List[int]
        :rtype: float
        """
        combList = sorted(nums1 + nums2)
        lenList = len(combList)
        if lenList % 2 == 0:
            indexFinder = lenList/2
            med = float((combList[indexFinder] + combList[indexFinder - 1]))
            return med/2

        medFinder = (lenList - 1) / 2
        return combList[medFinder]
```

Custom Testcase ( Contribute )

Run Code

Submit Solution

# 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) -> 2
(1,1) -> 0
(3,1) -> 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 <= k <= len(nums) * (len(nums) - 1) / 2`.

Python

```python
class Solution(object):
    def smallestDistancePair(self, nums, k):
        def possible(guess):
            #Is there k or more pairs with distance <= guess?
            count = left = 0
            for right, x in enumerate(nums):
                while x - nums[left] > guess:
                    left += 1
                count += right - left
            return count >= k

        nums.sort()
        lo = 0
        hi = nums[-1] - nums[0]
        while lo < hi:
            mi = (lo + hi) / 2
            if possible(mi):
                hi = mi
            else:
                lo = mi + 1

        return lo
```

**Custom Testcase** ( **Contribute** )

Run Code

Submit Solution

# Split Array Largest Sum

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.
```

Python

```python
class Solution(object):
    def splitArray(self, nums, m):
        """
        :type nums: List[int]
        :type m: int
        :rtype: int
        """
        def valid(mid):
            cnt = 0
            current = 0
            for n in nums:
                current += n
                if current>mid:
                    cnt += 1
                    if cnt>=m:
                        return False
                    current = n
            return True

        l = max(nums)
        h = sum(nums)

        while l<h:
            mid = l+(h-l)/2
            if valid(mid):
                h = mid
            else:
                l = mid+1
        return l
```

Custom Testcase ( Contribute )

Run Code Submit Solution