Agenda

1. 本期算法学习小组筹备组成员介绍
2. 讲解Binary Search
3. 本期学习小组学习计划

Binary Search

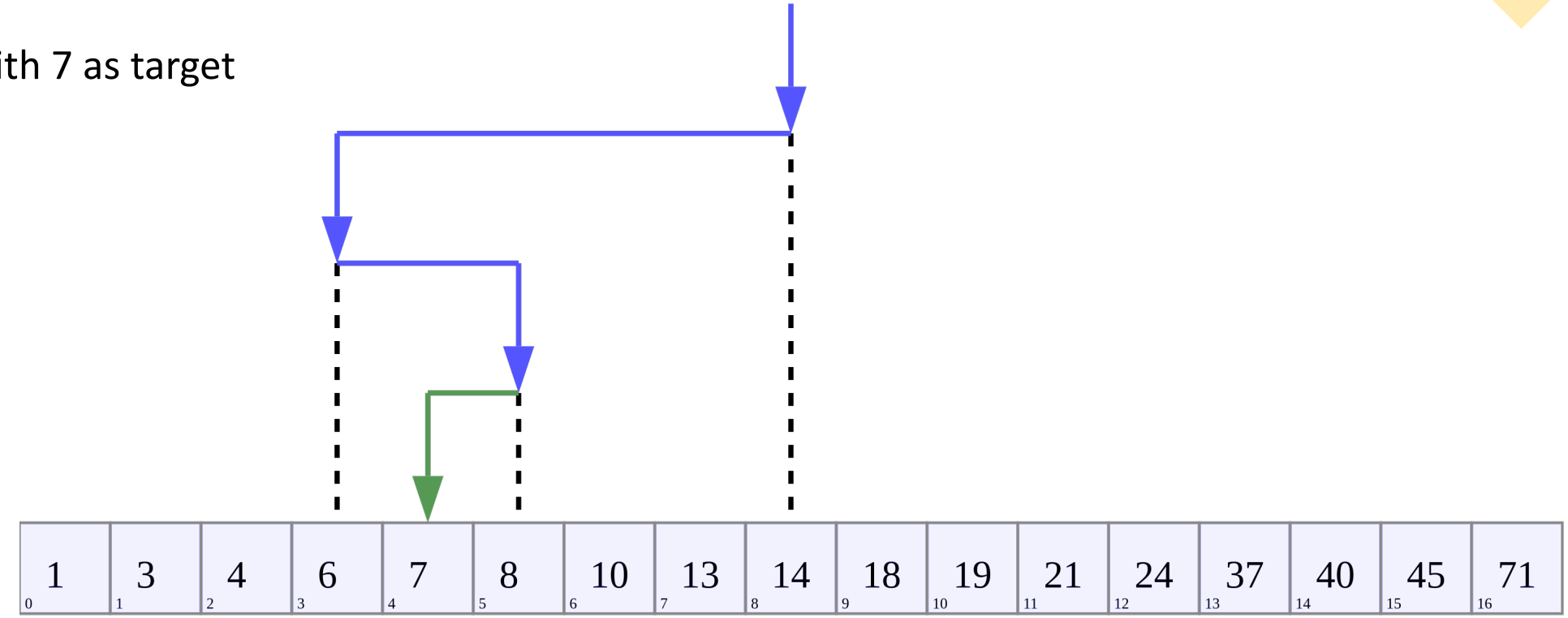
Wenting Yang

What is binary search

- Wikipedia: In computer science, binary search is a search algorithm that finds the position of a target value within **a sorted array**.

Binary search reduce the search space by half at each step

- An example with 7 as target



Time Complexity and Space Complexity

- Assuming the operation of evaluate array mid element vs. target takes O(1) time – which is usually the case
- T(n) = T(n/2) + O(1)
- Worst case time complexity – O(log n)
- Best case time complexity – O(1)
- Average time complexity – O(log n)

- Space complexity – O(1)

- **very FAST**

Range	Binary	Linear
100	7	100
10,000	14	10000
1,000,000	20	1,000,000
1,000,000,000	30	TLE

Common Issues

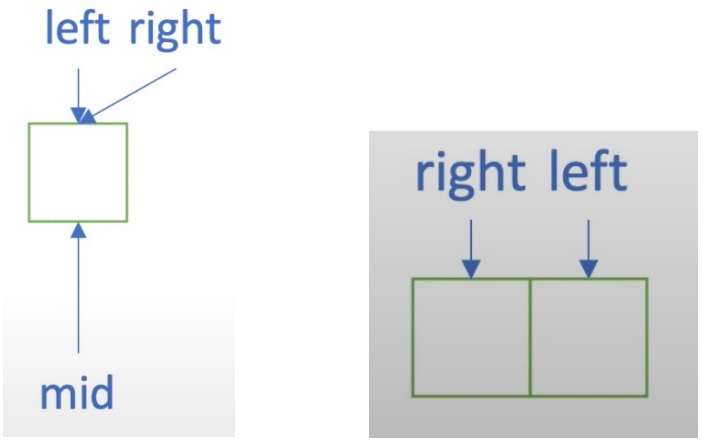
- When to stop
- Next feasible region (left or right; the boundary)

- Infinite loop **X**

Template I

Feasible region is always [left, right]

```python
def binary_search(self, nums, target):
    left, right = 0, len(nums) - 1

    while left <= right:
        mid = (left + right) // 2

        if nums[mid] < target:
            left = mid + 1

        elif nums[mid] == target:
            return mid

        else:
            right = mid - 1


    # terminate condition: left > right
    return -1
```



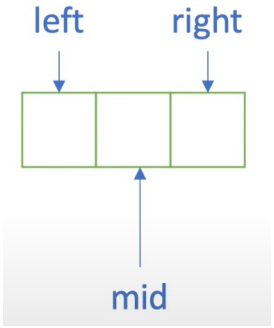
Example: find exact match of 3 (8)
[1, 3, 7, 20, 30]

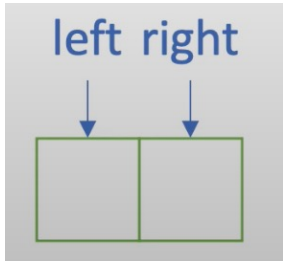
Hard to apply to questions to find first occurrence or last occurrence of a number in a sorted array
e.g. Find the first occurrence of 2
[1, 1, 2, 2, 2, 6, 7]

Template II – suits most scenarios

Feasible region is always (left, right), need to check left and right at the very end

```
1   def binary_search(self, nums, target):
2
3       left, right = 0, len(nums) - 1
4
5       while left + 1 < right:
6           mid = (left + right) // 2
7
8           if nums[mid] < target:
9               left = mid
10
11          elif nums[mid] == target:
12              # action depends on the question
13              # finding exact match, can return mid directly
14              # ...
15
16          else:
17              right = mid
18
19      # as the terminate condition is left+1 < right
20      # we need to check left and right
21      # depends on the question, the check can be > or <
22      # checking left before right or vice versa also depends
23      if nums[left] == target:
24          return left
25      if nums[right] == target:
26          return right
27
28      # didn't find any matches
29      return -1
```





Example 1: find exact match of 3 (8)

Line 12: return mid

[1, 3, 7, 20, 30]

Example 2: find the first occurrence of 2

Line 12: right = mid

[1, 1, 2, 2, 2, 6, 7]

Example 3: find the largest index with elements smaller than 2

1) Line 11 - 12: nums[mid]==target: right = mid

2) After exiting the loop, check right before left

[1, 1, 1, 2, 2, 6, 7]

Leetcode 278 – First Bad Version (Easy)

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 returns whether `version` is bad. Implement a function to find the first bad version. You should minimize the number of calls to the API.

```
Input: n = 5, bad = 4
Output: 4
```

Leetcode 278 – 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 returns whether `version` is bad. Implement a function to find the first bad version. You should minimize the number of calls to the API.

```
Input: n = 5, bad = 4
Output: 4
```

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

class Solution:
    def firstBadVersion(self, n):
        """
        :type n: int
        :rtype: int
        """
        start, end = 1, n
        while start + 1 < end:
            mid = (start + end) // 2
            if isBadVersion(mid):
                end = mid
            else:
                start = mid

        if isBadVersion(start):
            return start
        else:
            return end
```

TC: O(log n) ; SC: O(1)

Leetcode 33 Search in Rotated Sorted Array (Medium)

There is an integer array `nums` sorted in ascending order (with **distinct** values).

Prior to being passed to your function, `nums` is **possibly rotated** at an unknown pivot index `k` (`1 <= k < nums.length`) such that the resulting array is `[nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]]` (**0-indexed**). For example, `[0,1,2,4,5,6,7]` might be rotated at pivot index `3` and become `[4,5,6,7,0,1,2]`.

Given the array `nums` **after** the possible rotation and an integer `target`, return *the index of* `target` *if it is in* `nums`, *or* `-1` *if it is not in* `nums`.

You must write an algorithm with `O(log n)` runtime complexity.

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

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

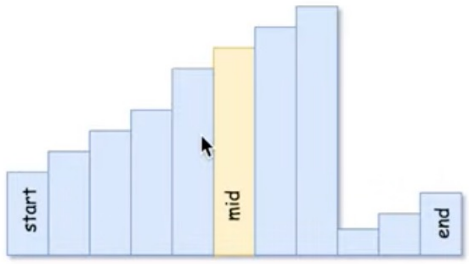
Leetcode 33 Search in Rotated Sorted Array (Medium)

There is an integer array `nums` sorted in ascending order (with **distinct** values).

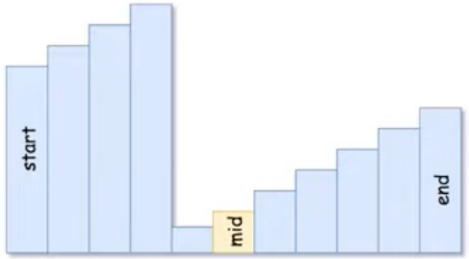
Prior to being passed to your function, `nums` is **possibly rotated** at an unknown pivot index `k` (`1 <= k < nums.length`) such that the resulting array is `[nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]]` (**0-indexed**). For example, `[0,1,2,4,5,6,7]` might be rotated at pivot index `3` and become `[4,5,6,7,0,1,2]`.

Given the array `nums` **after** the possible rotation and an integer `target`, return *the index of* `target` *if it is in* `nums`, *or* `-1` *if it is not in* `nums`.

You must write an algorithm with `O(log n)` runtime complexity.



Case 1 (mid >= start)



Case 2 (mid < start)

```python
def search(self, nums: List[int], target: int) -> int:
    start, end = 0, len(nums) - 1
    while start + 1 < end:
        mid = (start + end) // 2
        if nums[mid] == target:
            return mid
        if nums[mid] > nums[end]:
            if nums[start] <= target < nums[mid]:
                end = mid
            else:
                start = mid
        else:
            if nums[mid] < target <= nums[end]:
                start = mid
            else:
                end = mid

    if nums[start] == target:
        return start
    if nums[end] == target:
        return end

    return -1
```

TC: O(log n) ; SC: O(1)

Leetcode 162 – Find Peak Element (Medium)

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

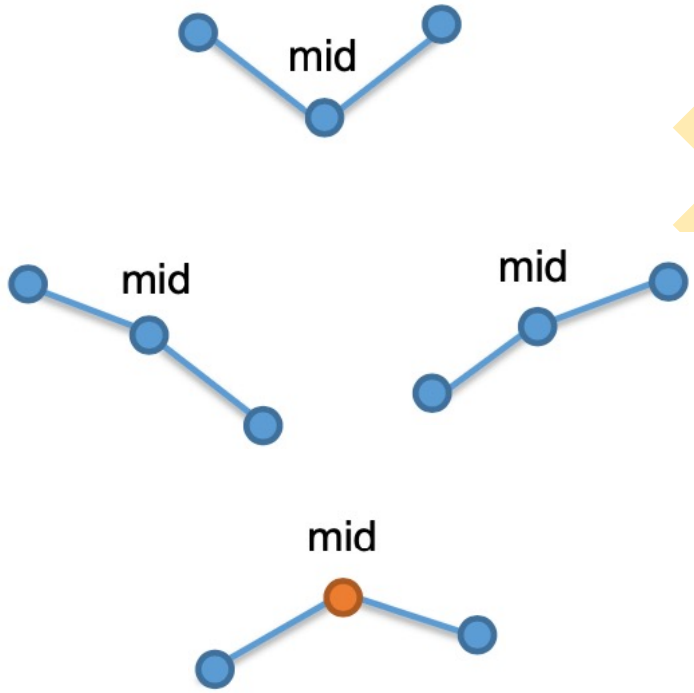
Given a **0-indexed** integer array `nums`, find a peak element, and return its index. If the array contains multiple peaks, return the index to **any of the peaks**.

You may imagine that `nums[-1] = nums[n] = -∞`. In other words, an element is always considered to be strictly greater than a neighbor that is outside the array.

You must write an algorithm that runs in `O(log n)` time.

Constraints:

- `1 <= nums.length <= 1000`
- `-2^31 <= nums[i] <= 2^31 - 1`
- `nums[i] != nums[i + 1]` for all valid `i`.




```
Input: nums = [1,2,1,3,5,6,4]
Output: 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.
```

Leetcode 162 – Find Peak Element (Medium)

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

Given a **0-indexed** integer array `nums`, find a peak element, and return its index. If the array contains multiple peaks, return the index to **any of the peaks**.

You may imagine that `nums[-1] = nums[n] = -∞`. In other words, an element is always considered to be strictly greater than a neighbor that is outside the array.

You must write an algorithm that runs in `O(log n)` time.

Constraints:

- `1 <= nums.length <= 1000`
- `-2^31 <= nums[i] <= 2^31 - 1`
- `nums[i] != nums[i + 1]` for all valid `i`.

Input: nums = [1,2,1,3,5,6,4]
Output: 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.

```python
class Solution:
    def findPeakElement(self, nums: List[int]) -> int:
        start, end = 0, len(nums) - 1
        while start+1<end:
            mid = (start + end) // 2
            if nums[mid] < nums[mid - 1]:
                end = mid
            elif nums[mid] < nums[mid + 1]:
                start = mid
            else:
                return mid

        if nums[start] >nums[end]:
            return start
        else:
            return end
```

TC: O(log n) ; SC: O(1)

Leetcode 4 – Median of Two Sorted Arrays (Hard)

Given two sorted arrays `nums1` and `nums2` of size `m` and `n` respectively, return **the median** of the two sorted arrays.

The overall run time complexity should be `O(log (m+n))`.

```
Input: nums1 = [1,3], nums2 = [2]
Output: 2.00000
Explanation: merged array = [1,2,3] and median is 2.
```

```
Input: nums1 = [1,2], nums2 = [3,4]
Output: 2.50000
Explanation: merged array = [1,2,3,4] and median is (2 + 3) / 2 =
2.5.
```

Leetcode 4 – Median of Two Sorted Arrays (Hard)

Given two sorted arrays `nums1` and `nums2` of size `m` and `n` respectively, return **the median** of the two sorted arrays.

The overall run time complexity should be `O(log (m+n))`.

Solution 1:
Concate two arrays then sort
Drawback: didn't leverage the "sorted arrays" property
TC: O((m+n)*log(m+n)) ; SC: O(m+n)

Solution 2:
Merge two sorted array into one sorted array then get median (using two pointers technique, the same as LC 88)

TC: O(m+n); SC: O(m+n)

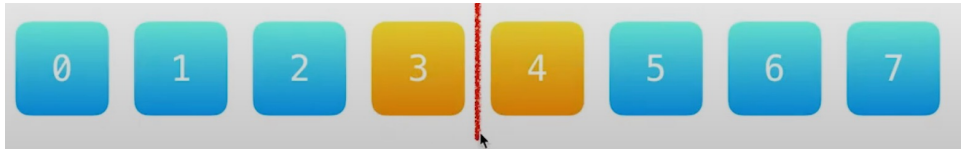
Leetcode 4 – Median of Two Sorted Arrays (Hard)

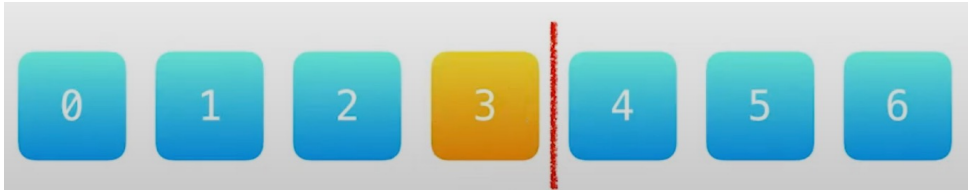
Given two sorted arrays `nums1` and `nums2` of size `m` and `n` respectively, return **the median** of the two sorted arrays.

The overall run time complexity should be `O(log (m+n))`.

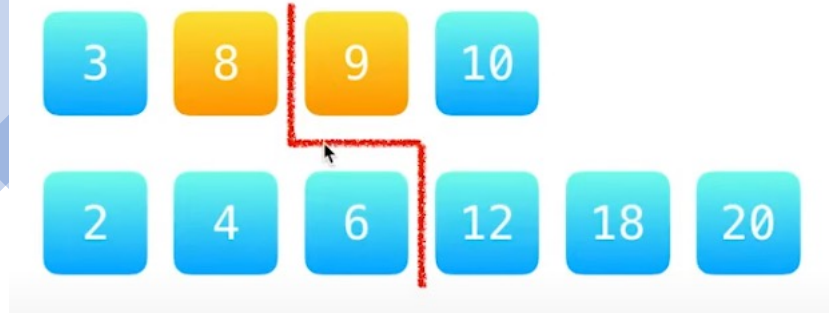
Solution 3: A modified binary search

- Median on one array





- Median with two arrays?



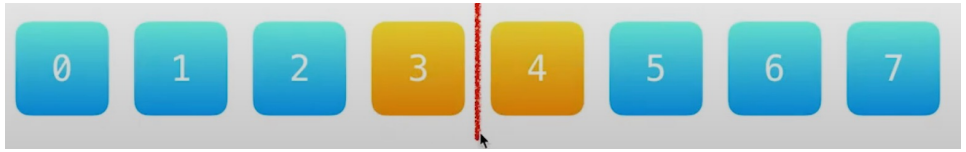
Leetcode 4 – Median of Two Sorted Arrays (Hard)

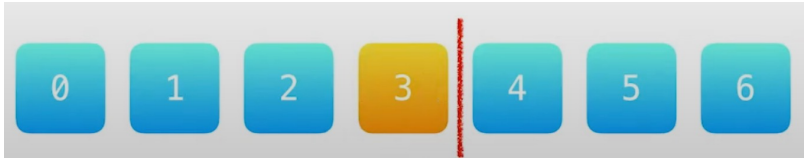
Given two sorted arrays `nums1` and `nums2` of size `m` and `n` respectively, return **the median** of the two sorted arrays.

The overall run time complexity should be `O(log (m+n))`.

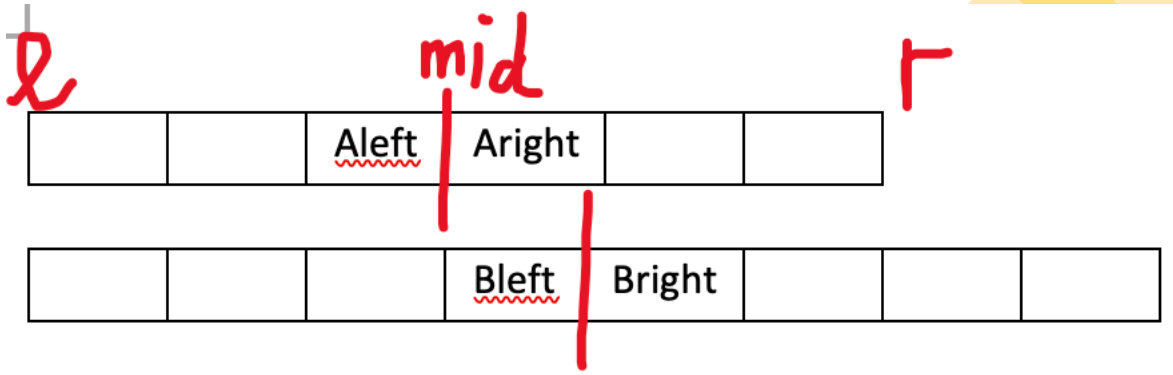
Solution 3: A modified binary search

- Median on one array





- Median with two arrays?

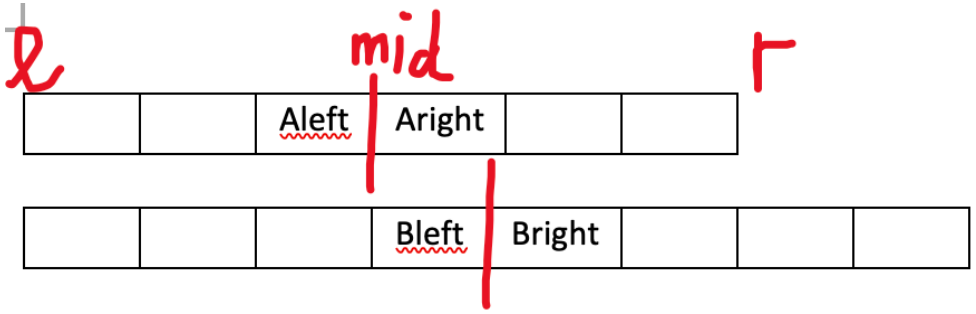


1. Size of left === size of right (if even); size of left == size of right + 1 (if odd)
2. Aleft<= Bright && Bleft <= Aright
    1. Aleft > Bright => r = mid – 1
    2. Bleft > Aright => l = mid + 1

Leetcode 4 – Median of Two Sorted Arrays (Hard)

Given two sorted arrays `nums1` and `nums2` of size `m` and `n` respectively, return **the median** of the two sorted arrays.

The overall run time complexity should be `O(log (m+n))`.



1. Size of left === size of right (if even); size of left == size of right + 1 (if odd)
2. Aleft<= Bright && Bleft <= Aright
    1. Aleft > Bright => r = mid – 1
    2. Bleft > Aright => l = mid + 1

TC: O(log(min(m,n))) ; SC: O(1)

```python
def findMedianSortedArrays(self, nums1: List[int], nums2: List[int]) -> float:
    # always search on the shorter array
    A, B = nums1, nums2
    if len(B) < len(A):
        A, B = B, A

    m, n = len(A), len(B)
    l, r = 0, m
    total_left = (m+n+1) // 2
    while l<=r:
        mid_A = (l+r) // 2
        mid_B = total_left - mid_A

        Aleft = A[mid_A-1] if mid_A-1>=0 else float("-inf")
        Aright = A[mid_A] if mid_A < len(A) else float("inf")
        Bleft = B[mid_B-1] if mid_B-1>=0 else float("-inf")
        Bright = B[mid_B] if mid_B < len(B) else float("inf")

        if Aleft <= Bright and Bleft <= Aright:
            # odd
            if (m+n) % 2:
                return max(Aleft, Bleft)
            else:
                return (max(Aleft, Bleft) + min(Aright, Bright)) / 2
        elif Aleft > Bright:
            r = mid_A-1
        else:
            l = mid_A+1
```

Leetcode 875 – Koko Eating Bananas (Medium)

Koko loves to eat bananas. There are `n` piles of bananas, the `i^th` pile has `piles[i]` bananas. The guards have gone and will come back in `h` hours.

Koko can decide her bananas-per-hour eating speed of `k`. Each hour, she chooses some pile of bananas and eats `k` bananas from that pile. If the pile has less than `k` bananas, she eats all of them instead and will not eat any more bananas during this hour.

Koko likes to eat slowly but still wants to finish eating all the bananas before the guards return.

Return *the minimum integer* `k` *such that she can eat all the bananas within* `h` *hours*.

```
Input: piles = [3,6,7,11], h = 8
Output: 4
```

Leetcode 875 – Koko Eating Bananas (Medium)

Koko loves to eat bananas. There are `n` piles of bananas, the `i^th` pile has `piles[i]` bananas. The guards have gone and will come back in `h` hours.

Koko can decide her bananas-per-hour eating speed of `k`. Each hour, she chooses some pile of bananas and eats `k` bananas from that pile. If the pile has less than `k` bananas, she eats all of them instead and will not eat any more bananas during this hour.

Koko likes to eat slowly but still wants to finish eating all the bananas before the guards return.

Return *the minimum integer* `k` *such that she can eat all the bananas within* `h` *hours*.

Input: piles = [3,6,7,11], h = 8
Output: 4

```python
def minEatingSpeed(self, piles: List[int], h: int) -> int:
    start, end = 1, max(piles)
    while start + 1 < end:
        mid = (start + end) // 2
        if self.total_eating_time(piles, mid) > h:
            start = mid
        else:
            end = mid

    if self.total_eating_time(piles, start) <= h:
        return start
    else:
        return end

def total_eating_time(self, piles, k):
    total_hour = 0
    for p in piles:
        total_hour += (p-1) // k + 1
    return total_hour
```

TC: O(n*log m) ; SC: O(1)

Homework

- https://leetcode.com/problems/search-insert-position/
- https://leetcode.com/problems/search-a-2d-matrix/
- https://leetcode.com/problems/find-minimum-in-rotated-sorted-array/
- https://leetcode.com/problems/split-array-largest-sum/ (guess and check)
- https://leetcode.com/problems/time-based-key-value-store/ (design)

学习计划

- **第二到第八周上课时间**：周六**11:00 AM** EST（1hr to 1.5 hrs)
- **课件讲解：**正式组同学每人负责一个topic，提前一天上传slides至网盘
- **课后作业：**正式组同学需要完成，并上传github（请前往网盘阅读primer，按照正式组TODO操作）；旁听组不做要求
- **提问答疑：**平时在微信群里提问，大家一起讨论
- **模拟面试：**鼓励正式组学员参与两两mock interview，旁听组学员也可参与

- 课件和其他资料都会上传网盘，请大家阅读网盘中的primer

Schedule & Topics

	Date	Topic	Host Name
Week 1	8/6/2022	Binary Search	Wenting Yang
Week 2	8/13/2022	Tree	Xiyu Li
Week 3	8/20/2022	Divide & Conquer	Ruochen Cao
Week 4	8/27/2022	DFS & BFS	Weijing Song
Week 5	9/3/2022	Backtracking	Yiwei Gu
Week 6	9/10/2022	Algorithm in graph	Zhiyue Wei
Week 7	9/17/2022	Dynamic Programming	Yihan Liu
Week 8	9/24/2022	Bit Manipulation	Xiaofeng Lu

旁听班报名：
https://www.qishicpc.com/activities/profile/317/