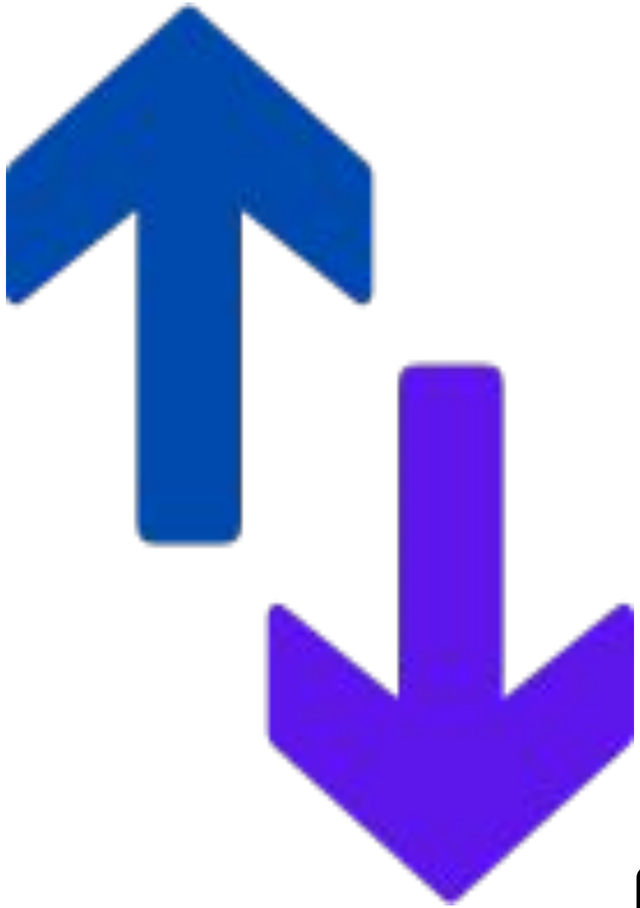
Two Pointers



Africa To Silicon Valley

Lecture Flow

- Prerequisites
- Definitions
- Different Variants
- Things to Pay Attention (common pitfalls)
- Practice Questions
- Resources
- Quote of the Day

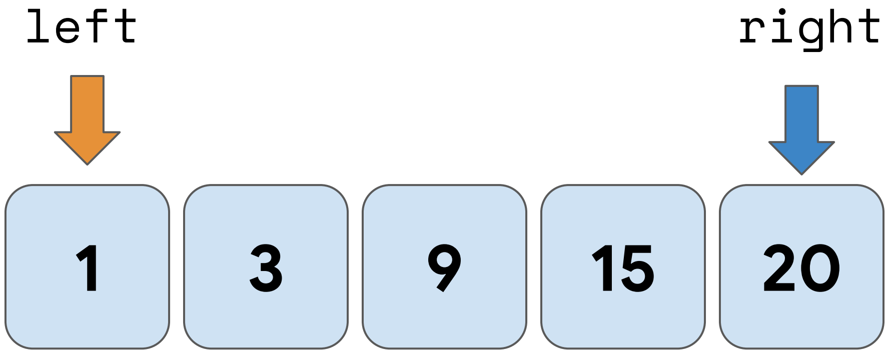
Prerequisites

- Basic flow control (for, while loops)
- Linear data structures

Definition

Definition

- Two Pointers technique is the use of two different pointers (usually to keep track of array or string indices) to solve a problem with the benefit of saving time and space.



Variants

A2SV
Africa To Silicon Valley

Parallel Pointers

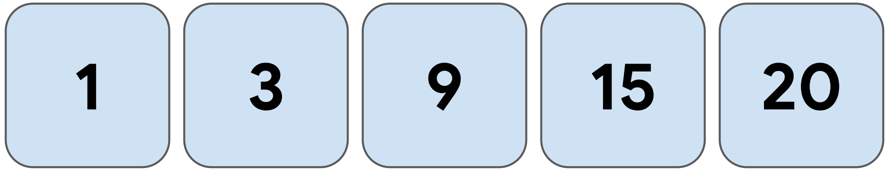
Africa To Silicon Valley

Variants - Parallel Pointers

Given an array of integers, determine if it is sorted in non-decreasing order.

Input: An array of integers.

Output: True or false, whether or not the array is sorted.

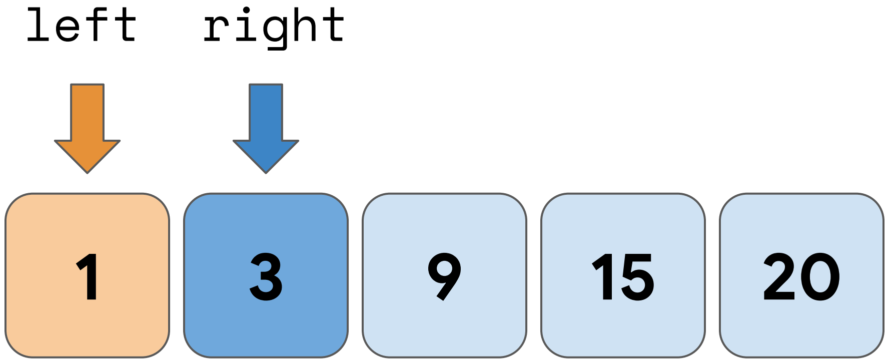


Parallel Pointers - Approach Pattern

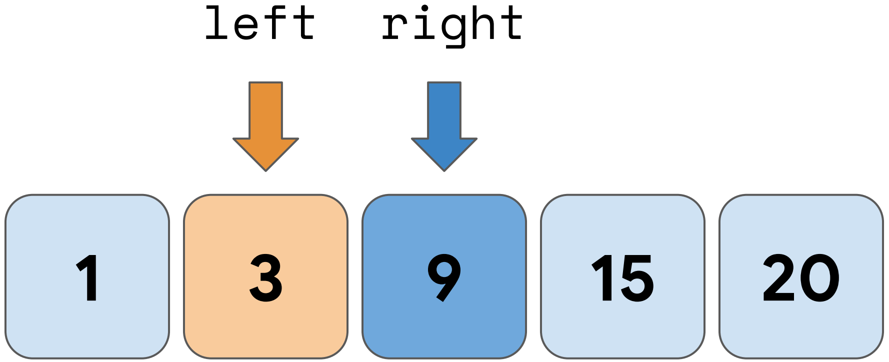
- In this problem pattern, we only need to look at two consecutive values.
- This is because for three consecutive numbers `a`, `b`, and `c`, if `a <= b` and `b <= c`, then `a <= c`.

- The two pointers will iterate parallel to each other, until the right-most one reaches the end of the array.

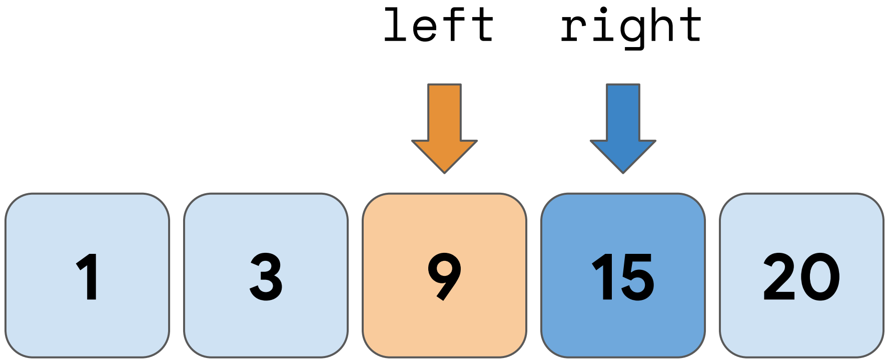
Parallel Pointers - Approach Pattern



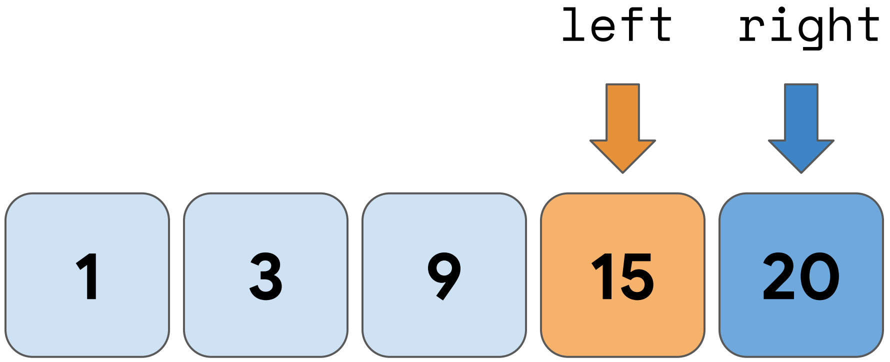
Parallel Pointers - Approach Pattern



Parallel Pointers - Approach Pattern



Parallel Pointers - Approach Pattern



Africa To Silicon Valley

Practice

Problem Link

Parallel Pointers - Practice solution

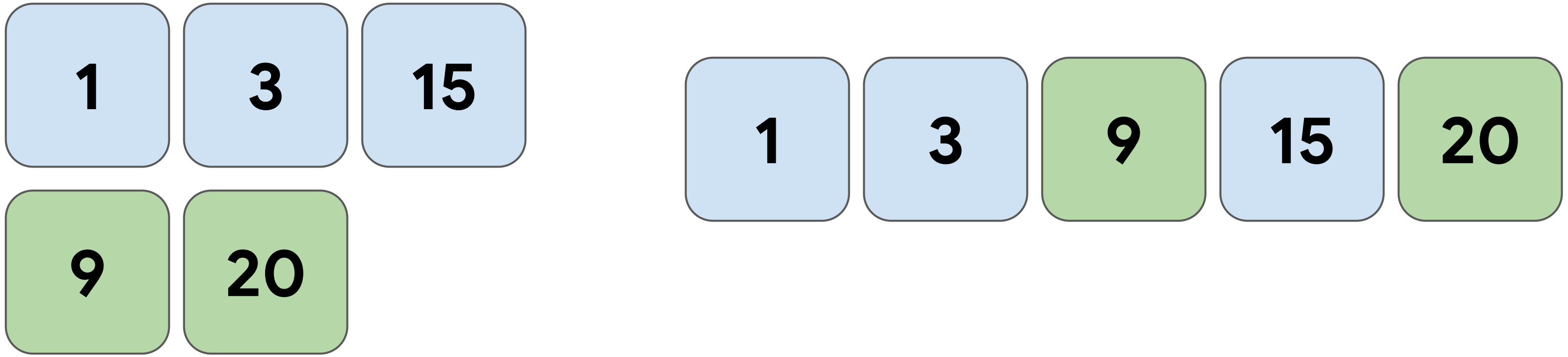
```
def isSorted(nums):
    left = 0
    right = 1
    size = len(nums)
    while right < size:
        if nums[left] > nums[right]:
            return False
        left += 1
        right += 1
    return True
```

Pointers on Separate Arrays

Pointers on Separate Arrays - Problem Pattern

- You are given two arrays, sorted in non-decreasing order. Merge them into one sorted array.

  Input: Two sorted arrays of integers.
  Output: The merged array.



Pointers on Separate Arrays- Bruteforce

- The easiest way to do this is with the following algorithm:

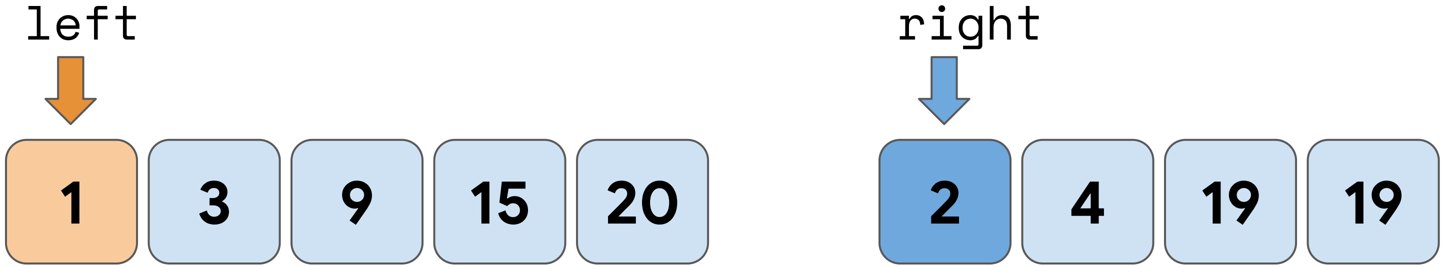
  a. Collect all the elements into one big array;
  b. Sort it with any sorting method built into your language.

- Such an algorithm will take time $O((m+n) \cdot \log(m+n))$
- m = size of array a.
- n= size of array b.

Africa To Silicon Valley

Pointers on Separate Arrays- Efficient

- Two arrays or Lists, each has been assigned a pointer

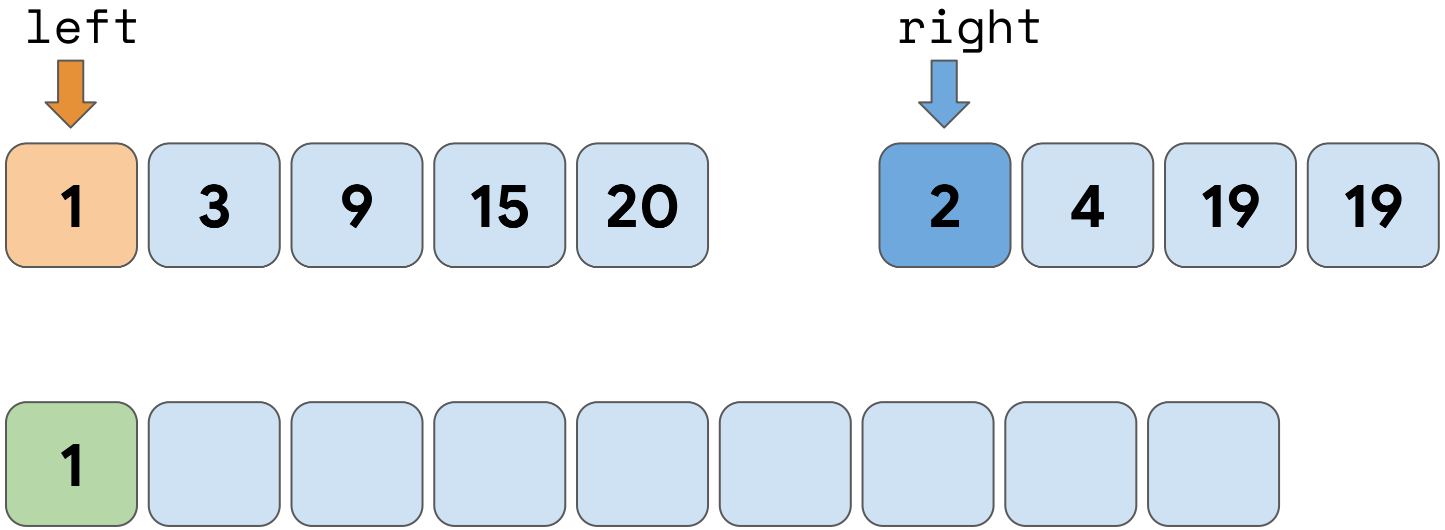


Pointers on Separate Arrays- Efficient

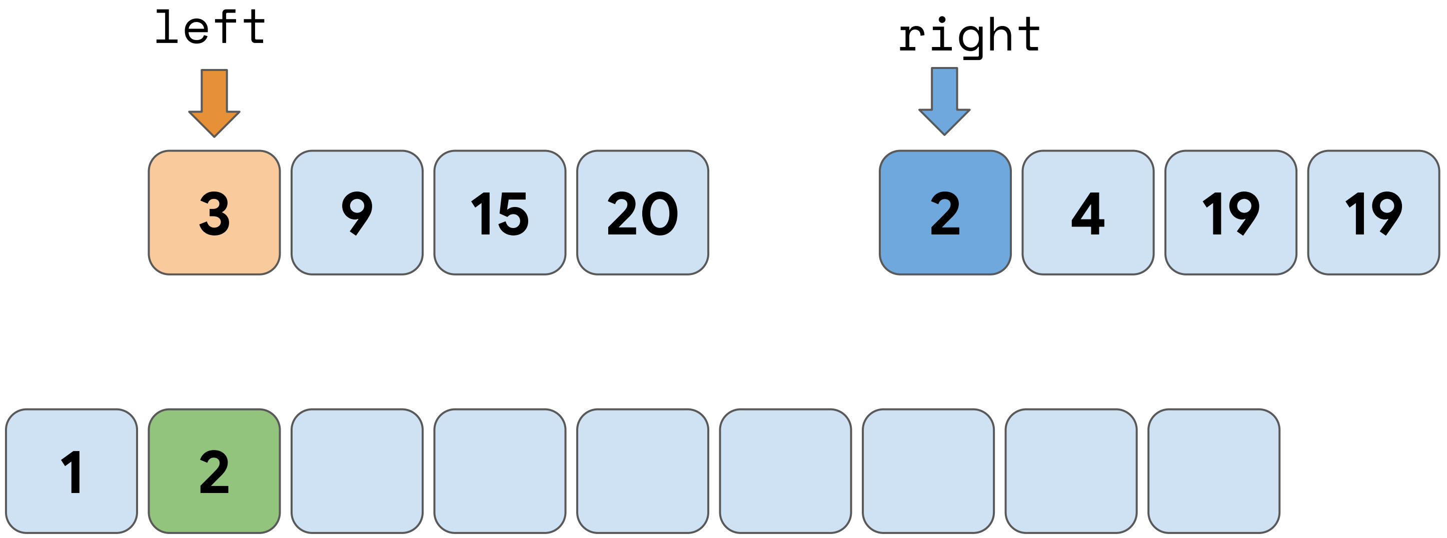
How do we merge them into one sorted array ?

- To answer this question, let's understand which element will be in the first position in the output array. Of course, this is the smallest element among all in first and second array. The smallest element of the first array is at the beginning of the array, same as with the second array.

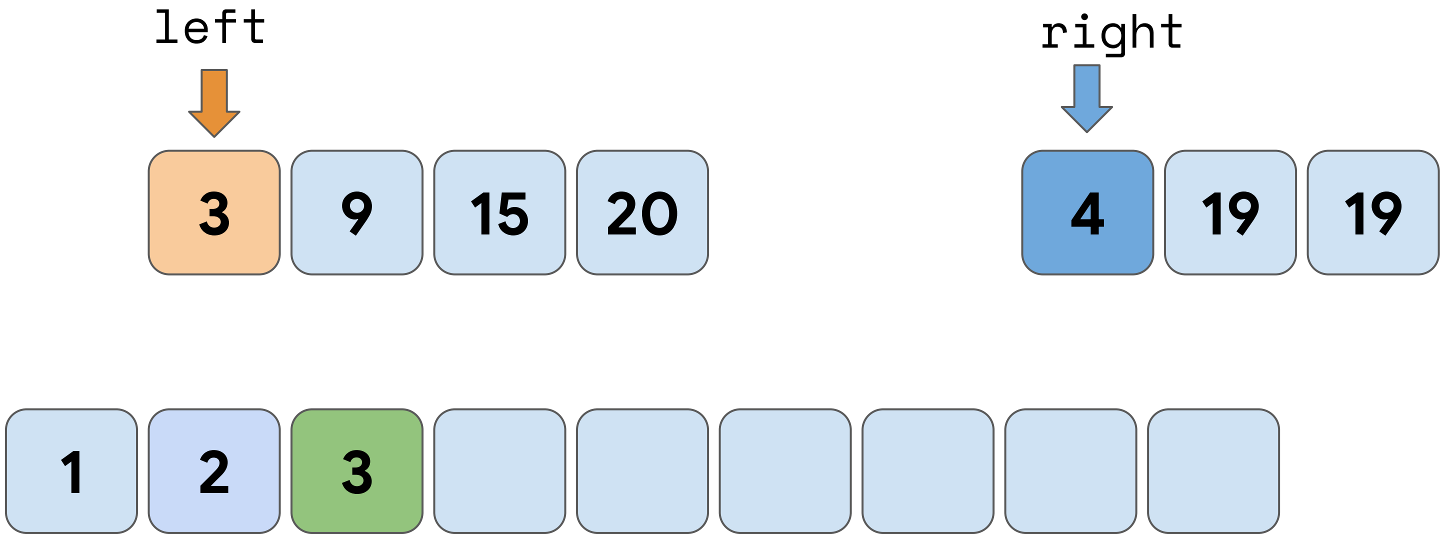
Pointers on Separate Arrays- Efficient



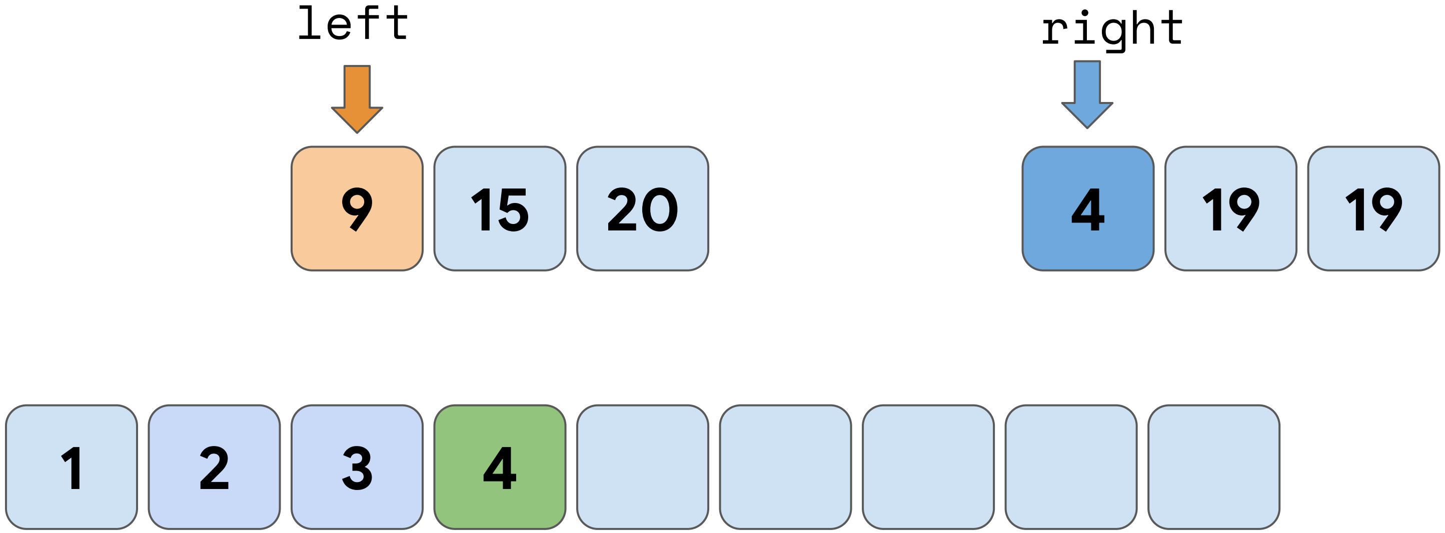
Pointers on Separate Arrays- Efficient



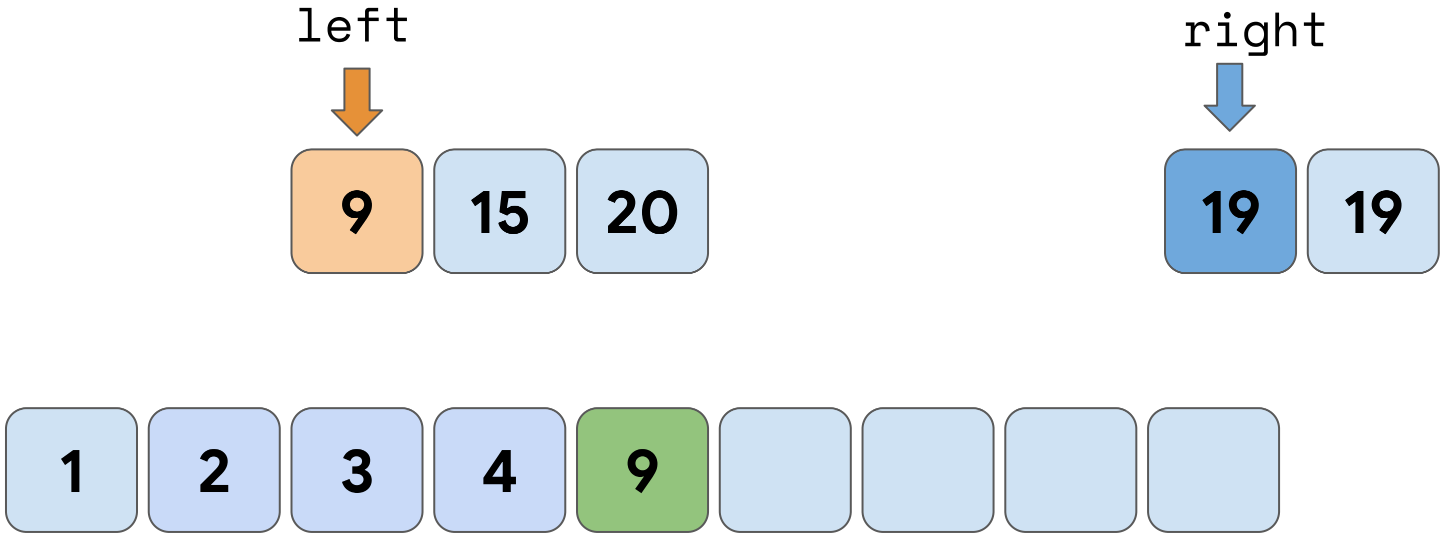
Pointers on Separate Arrays- Efficient



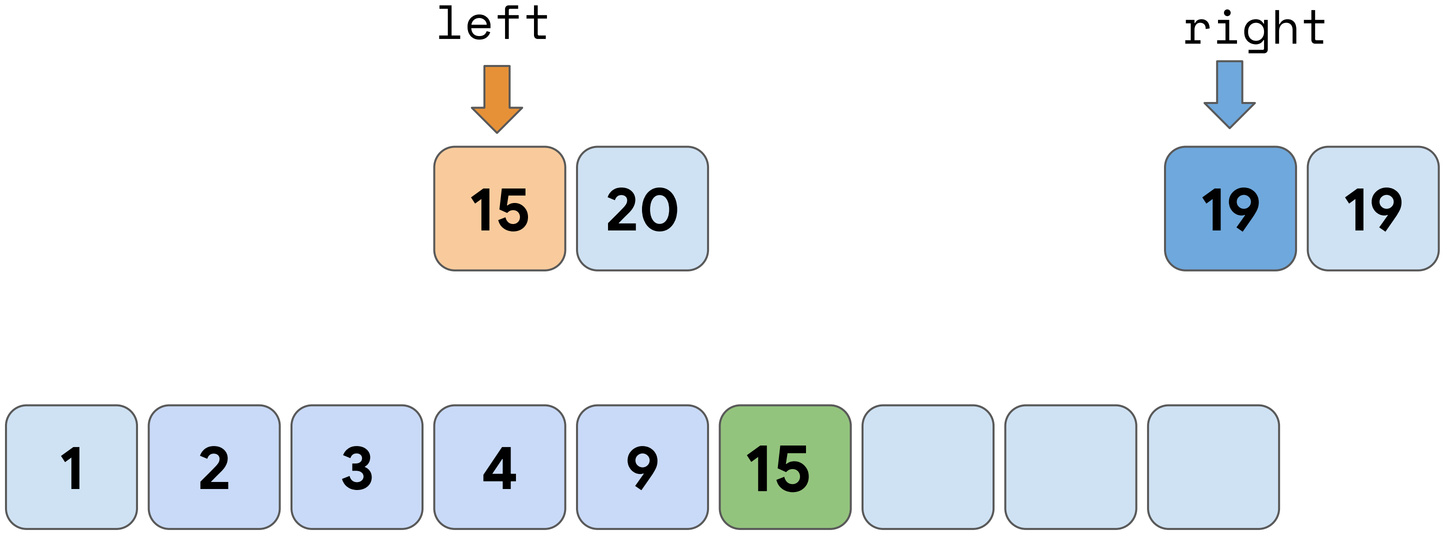
Pointers on Separate Arrays- Efficient



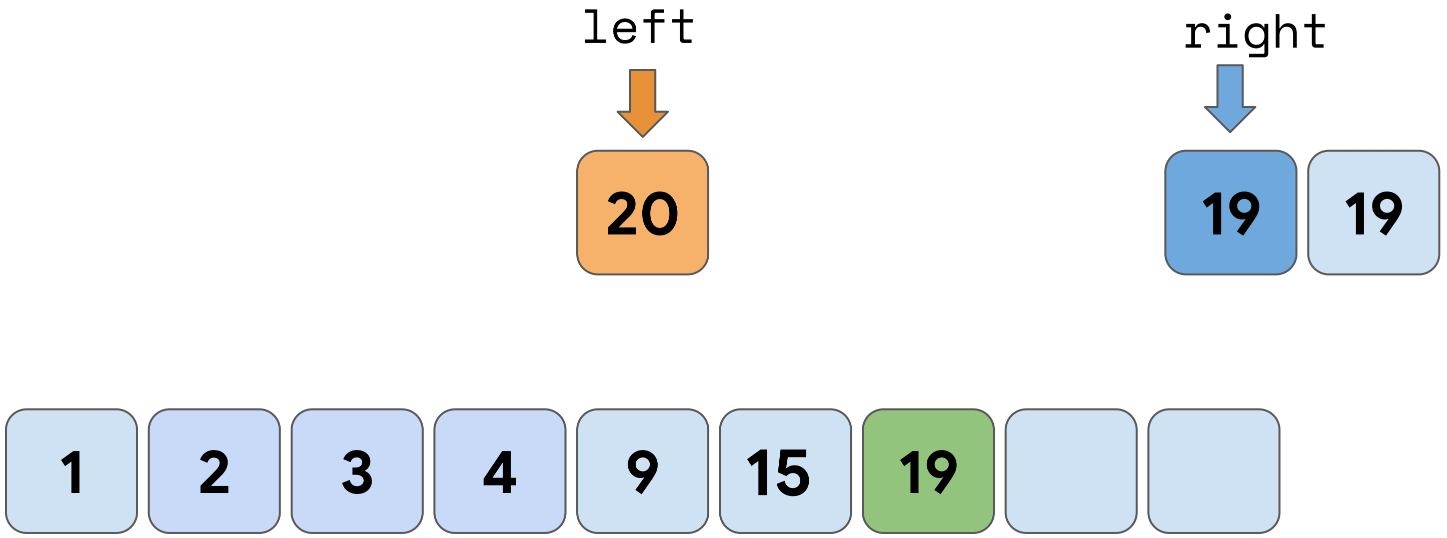
Pointers on Separate Arrays- Efficient



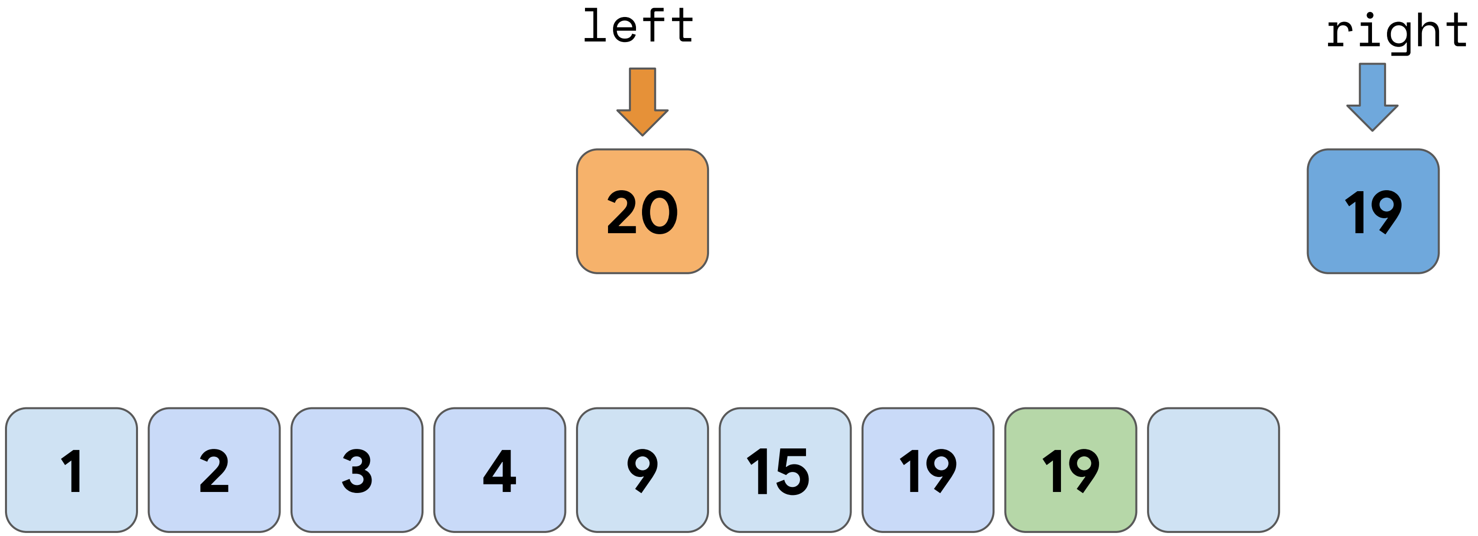
Pointers on Separate Arrays- Efficient



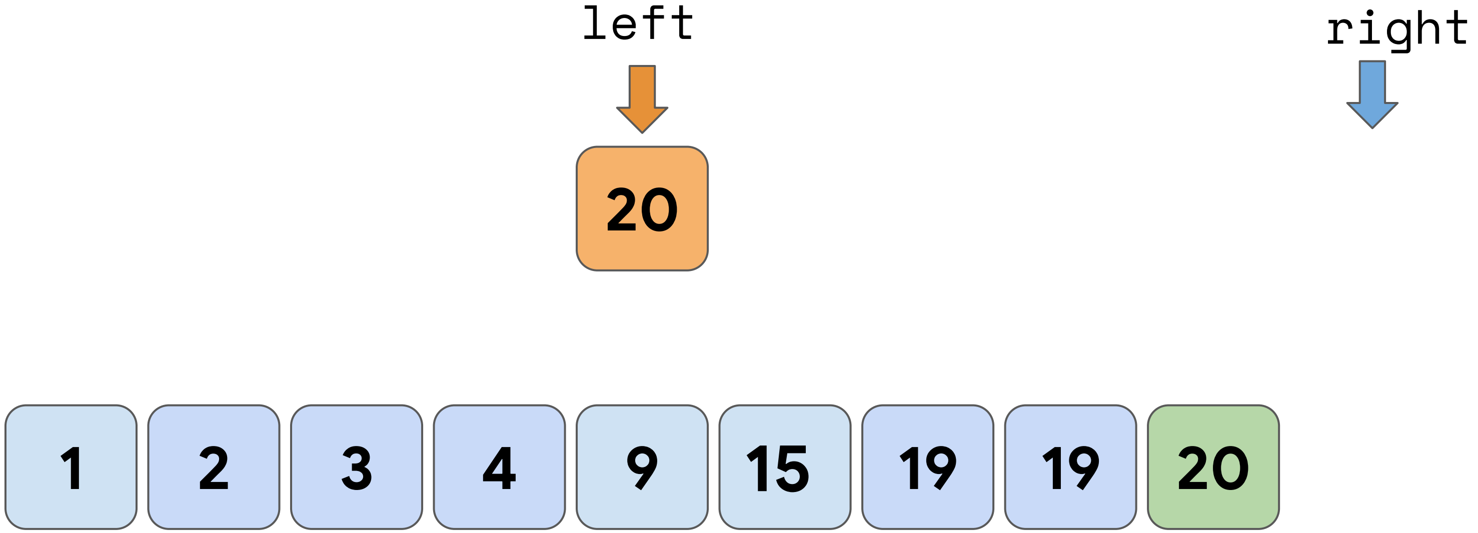
Pointers on Separate Arrays- Efficient



Pointers on Separate Arrays- Efficient



Pointers on Separate Arrays- Efficient



Pointers on Separate Arrays- Efficient

Practice

Problem Link

Pointers on Separate Arrays- Efficient

```
def mergeLists(list1, list2):
    merged = []
    first, second = 0, 0

    while first < len(list1) and second < len(list2):

        if list1[first] < list2[second]:
            merged.append(list1[first])
            first += 1

        else:
            merged.append(list2[second])
            second += 1

    merged.extend(list1[first:])
    merged.extend(list2[second:])

    return merged
```

Pointers on Separate Arrays- Efficient

Checkpoint - Link

Africa To Silicon Valley

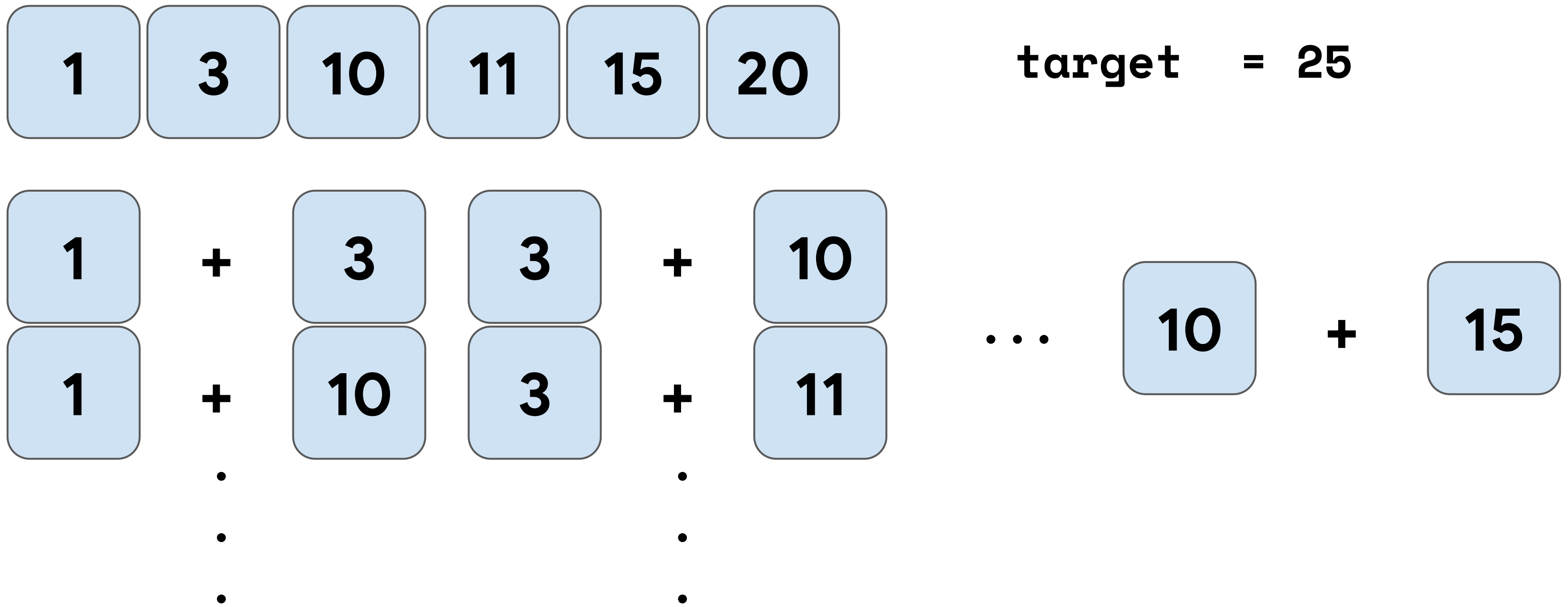
Colliding Pointers

Colliding Pointers - Problem Pattern

Given an array of integers that is sorted in non-decreasing order, find two numbers such that they add up to a specific target number (it is guaranteed that at least one pair exists).

- Input: A sorted array of integers and a target number.
- Output: Two numbers from the array that sum up to the target number.

Colliding Pointers - Bruteforce



`target = 25`

Colliding Pointers - Bruteforce

- We could implement a nested loop finding all possible pairs of elements and adding them.

```
for i in range(n-1):
    for j in range(i+1, n):
        if arr[i] + arr[j] == target:
            return [arr[i], arr[j]]

 return []
```

Time complexity: $O(n^2)$

Colliding Pointers - Efficient Solution

- One pointer starts from beginning and other from the end and they proceed towards each other.
- Since the array is sorted we can make some general observations
    - Smaller sums come from the left half of the array
    - Larger sums come from the right half of the array

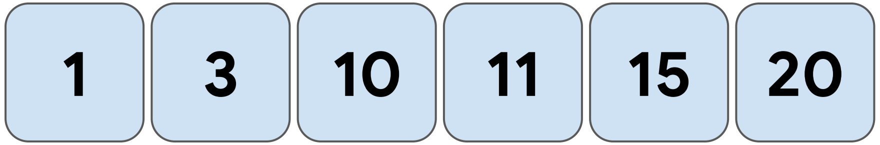
1	3	10	11	15	20

Colliding Pointers - Efficient Solution

- Therefore, using two pointers starting at the end points of the array, we can choose to increase or decrease our current sum however we like.

  The basic idea is that:

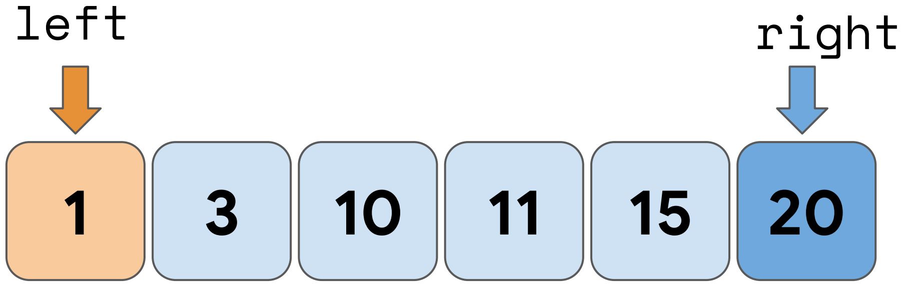
  - If our current sum is too small, move closer to the right.
  - If our current sum is too large, move closer to the left.



Colliding Pointers - Efficient Solution

`target = 25`

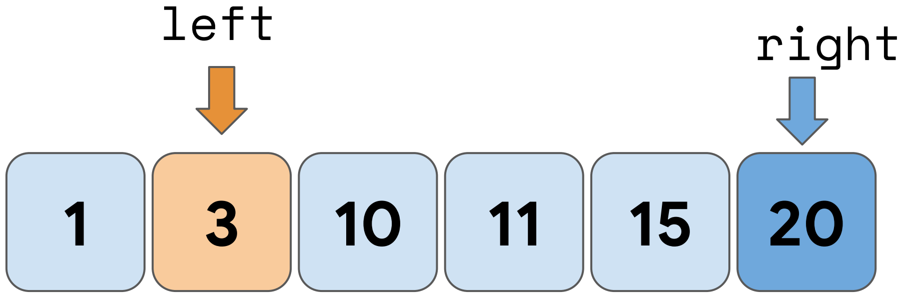
`current_sum = 21`



Colliding Pointers - Efficient Solution

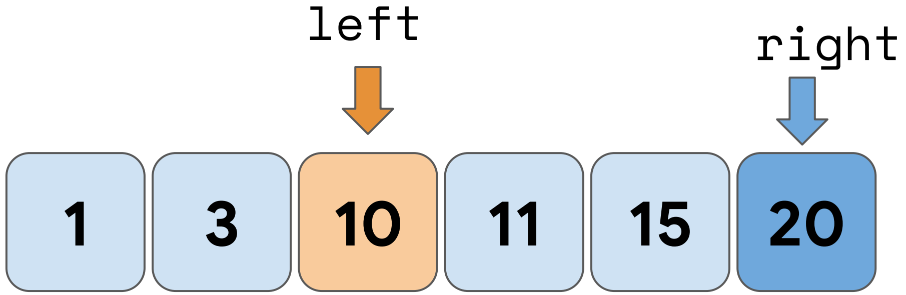
`target = 25`

`current_sum = 23`



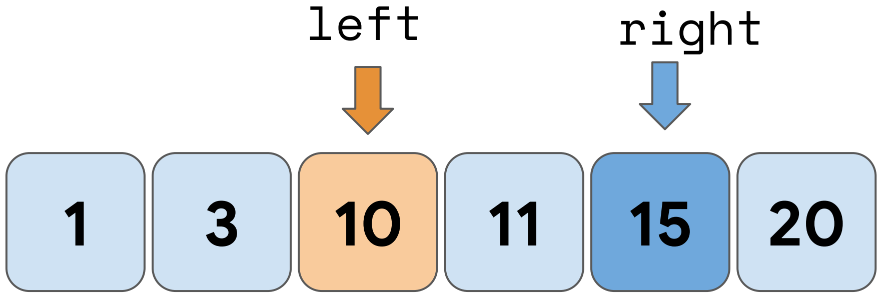
Colliding Pointers - Efficient Solution

```
target   = 25                    current_sum   = 30
```



Colliding Pointers - Efficient Solution

```
target = 25                current_sum = 25
```



Practice

Problem Link

Colliding Pointers - Implementation

```python
def twoSum(nums, target):
    left = 0
    right = len(nums) - 1
    cur_sum = 0
    while nums[left] + nums[right] != target:
        cur_sum = nums[left] + nums[right]
        if cur_sum < target:
            left += 1
        else:
            right -= 1

    return [nums[left], nums[right]]
```

Africa To Silicon Valley

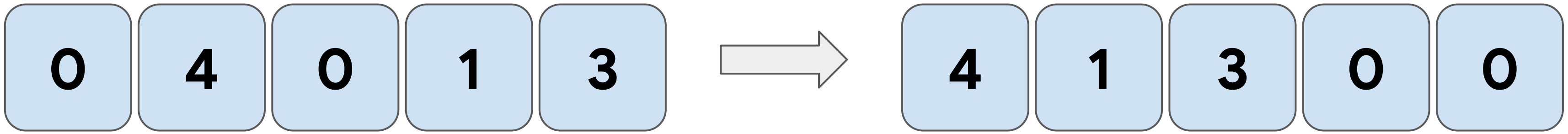
Seeker and Placeholder

Africa To Silicon Valley

Seeker and Placeholder- Problem Pattern

You are given an array, group all non-zero elements to the beginning of the array while maintaining their relative order. The modification must be done in-place.

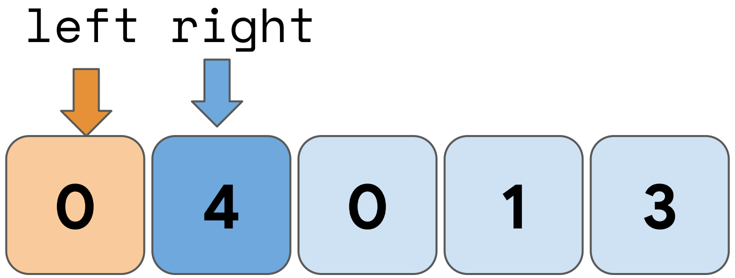
Input: An array of integers.
Output: The array, modified in-place, such that all non-zero numbers are at the beginning of the array.



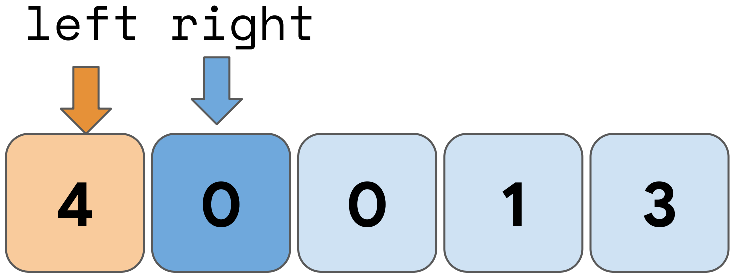
Seeker and Placeholder- Approach Pattern

- One pointer will iterate over the array, finding non-zero elements. The other pointer will point to the next valid position for a non-zero element.

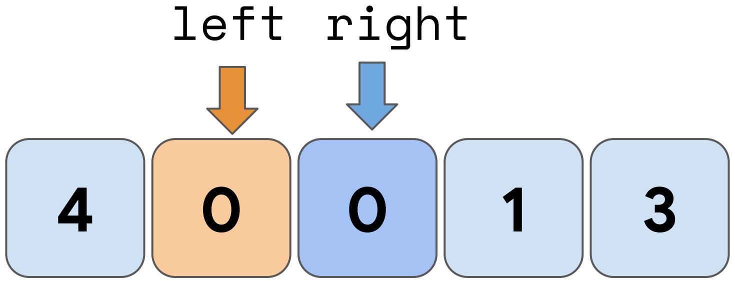
Seeker and Placeholder- Approach Pattern



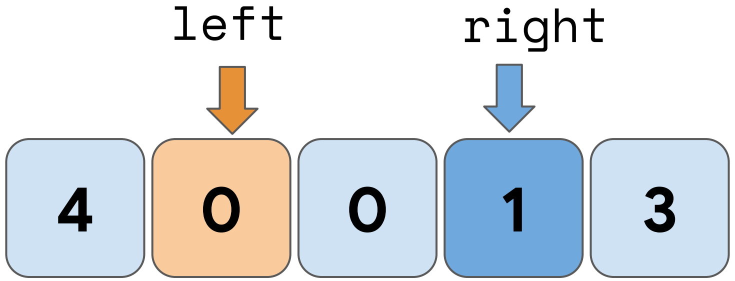
Seeker and Placeholder- Approach Pattern



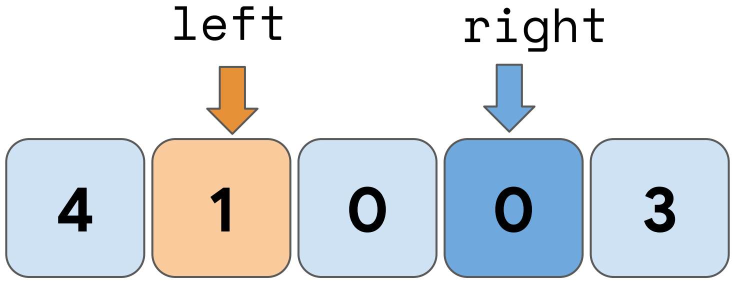
Seeker and Placeholder- Approach Pattern



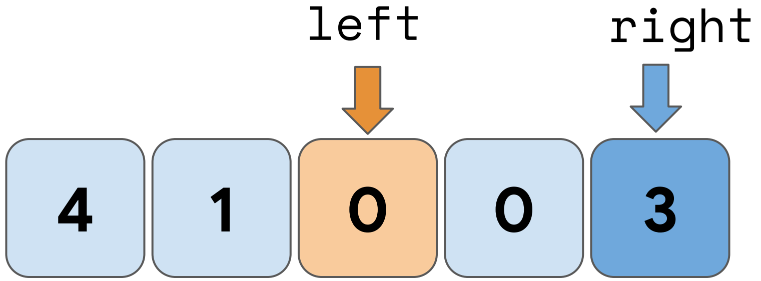
Seeker and Placeholder- Approach Pattern



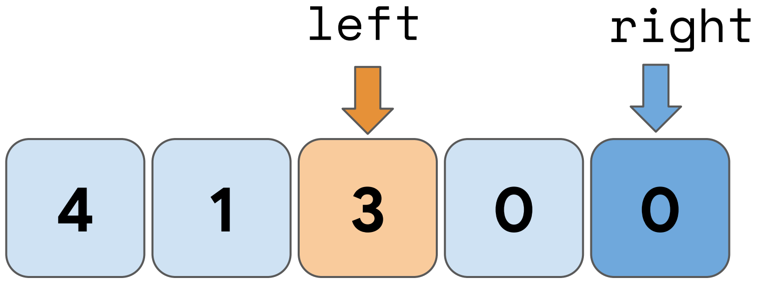
Seeker and Placeholder- Approach Pattern



Seeker and Placeholder- Approach Pattern



Seeker and Placeholder- Approach Pattern



Practice

Problem Link

Seeker and Placeholder- Implementation

```
def moveNonZeroes(nums):
    holder = 0
    seeker = 0

    while seeker < len(nums):
        if nums[seeker] != 0:
            nums[seeker], nums[holder] = nums[holder], nums[seeker]
            holder += 1
        seeker += 1
```

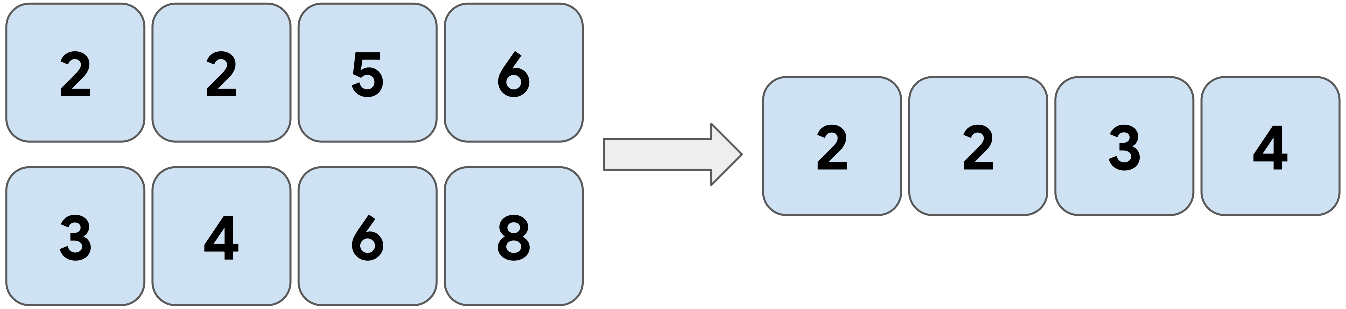
Africa To Silicon Valley

For-While Combo

For-While Combo - Problem Pattern

You are given two arrays, sorted in non-decreasing order. For each element of the second array, find the number of elements in the first array are that strictly less than it.

- Input: Two sorted arrays.
- Output: A single sorted array containing all elements from both arrays.



For-While Combo - Approach Pattern

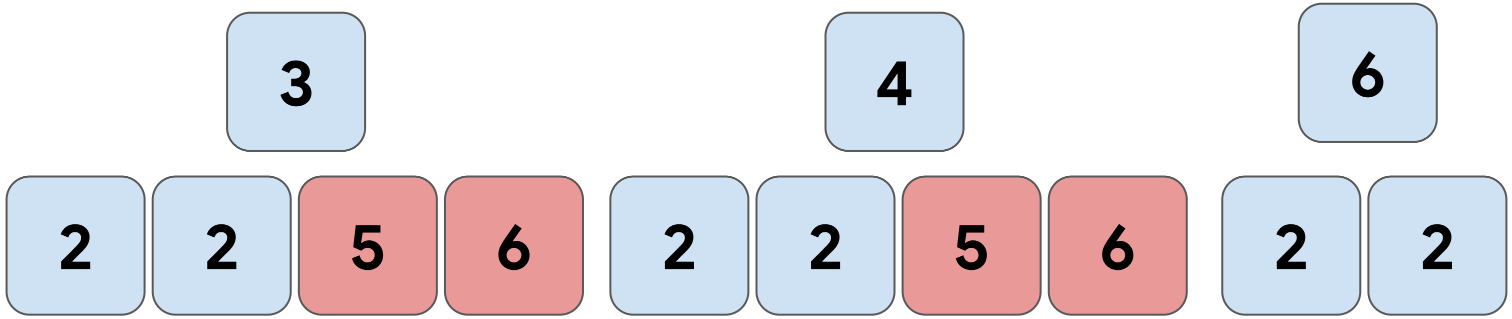
- The for-while combination is used when one pointer moves one step at a time, but the other one moves multiple steps at a time.

- This is usually applied to problems in which the position of one pointer is directly dependent on the position of the other.

A2SV
Africa To Silicon Valley

For-While Combo - Bruteforce

- What would a brute force approach look like?
- For every element in the second array, iterate over the first array and count the ones that are smaller than it.
- This is very inefficient - O(n2)

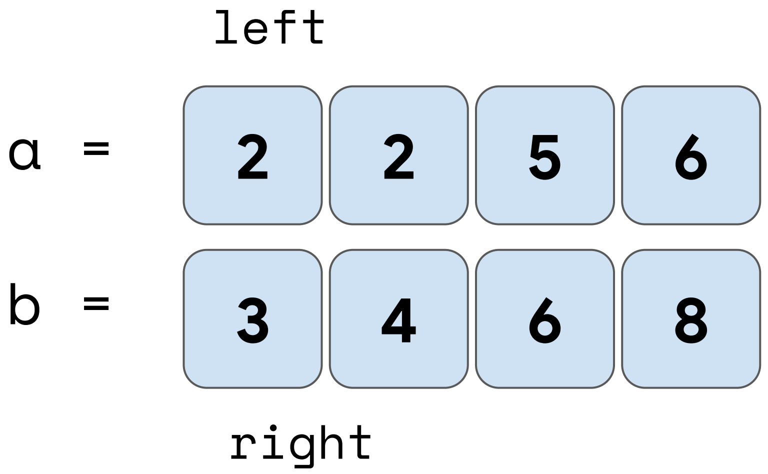


For-While Combo - Efficient

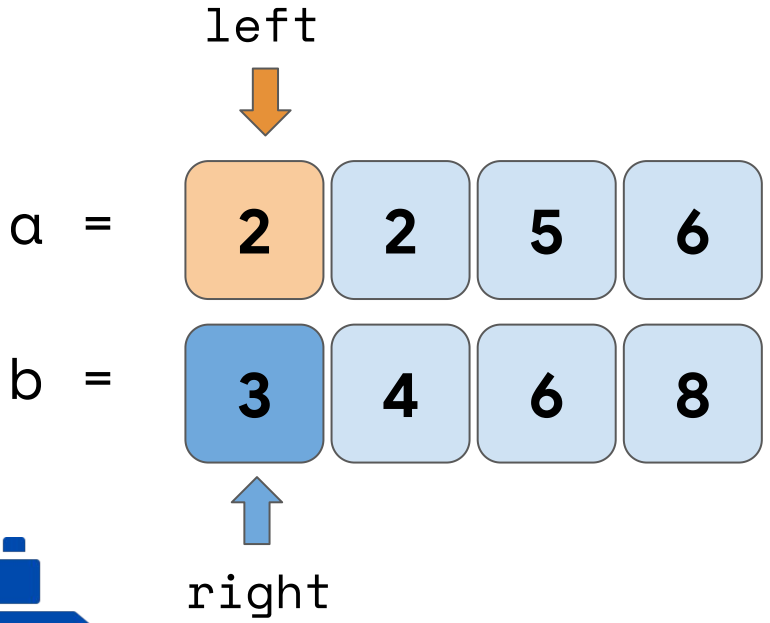
- Remember that both arrays are sorted
- This means that if there are n elements that are less than a specific target, then there is at least n guaranteed elements for the next target too.

For-While Combo - Efficient

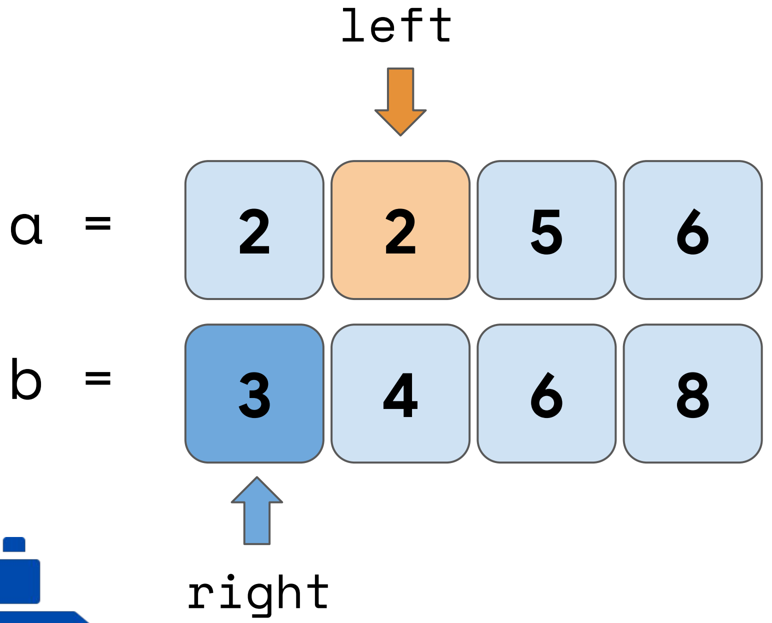
- We are going to move pointer **right** over array **b** and move **left** over array **a** until we find an a_{left} from array **a** that is greater than or equal to b_{right}. At this point, the value of **left** will represent the number of elements in **a** that are less than b_{right}.



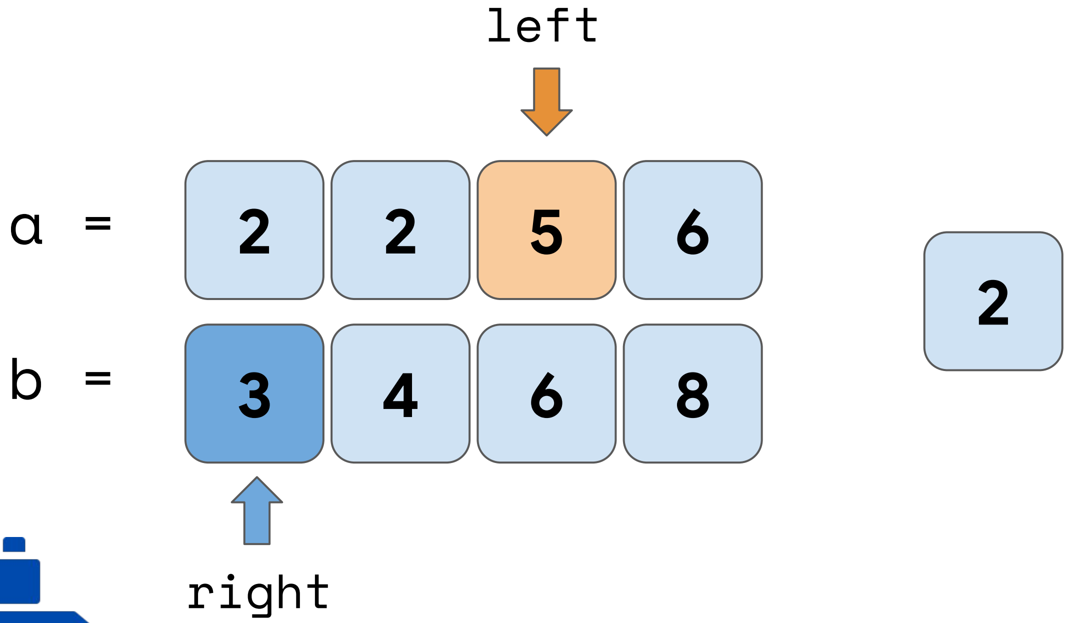
For-While Combo - Efficient



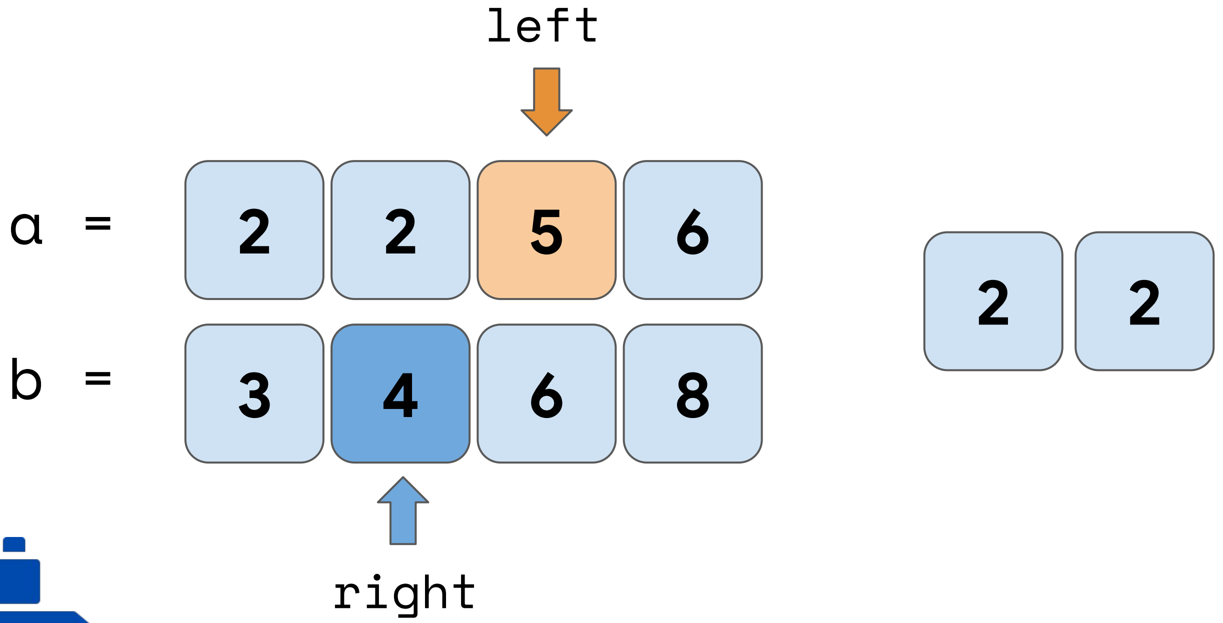
For-While Combo - Efficient



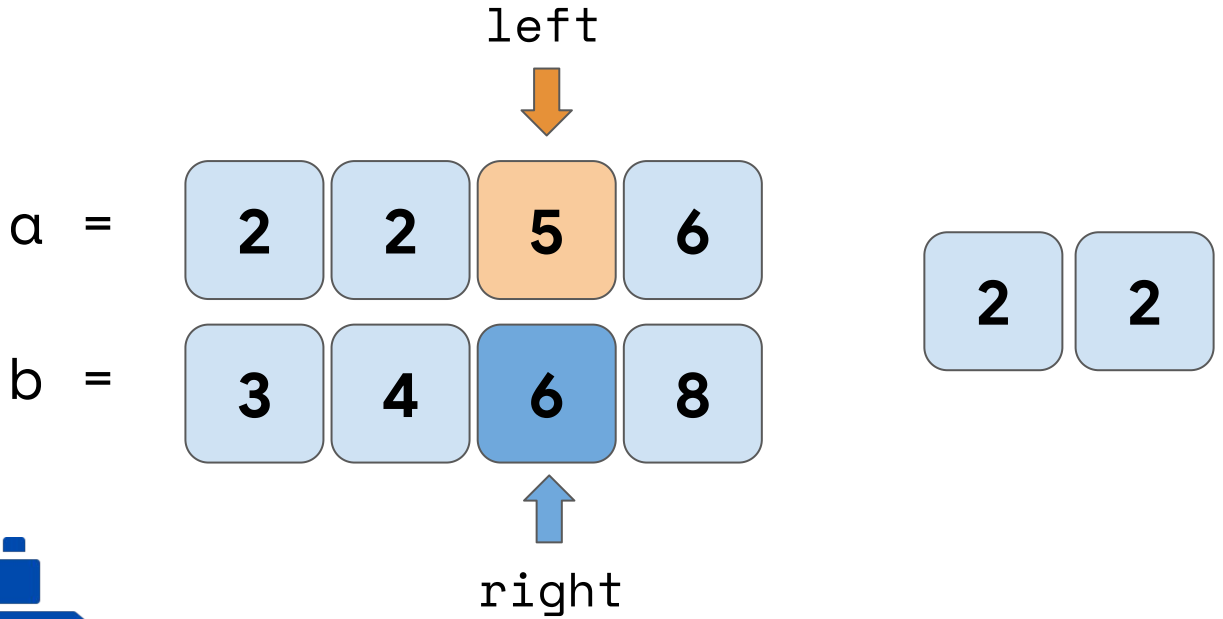
For-While Combo - Efficient



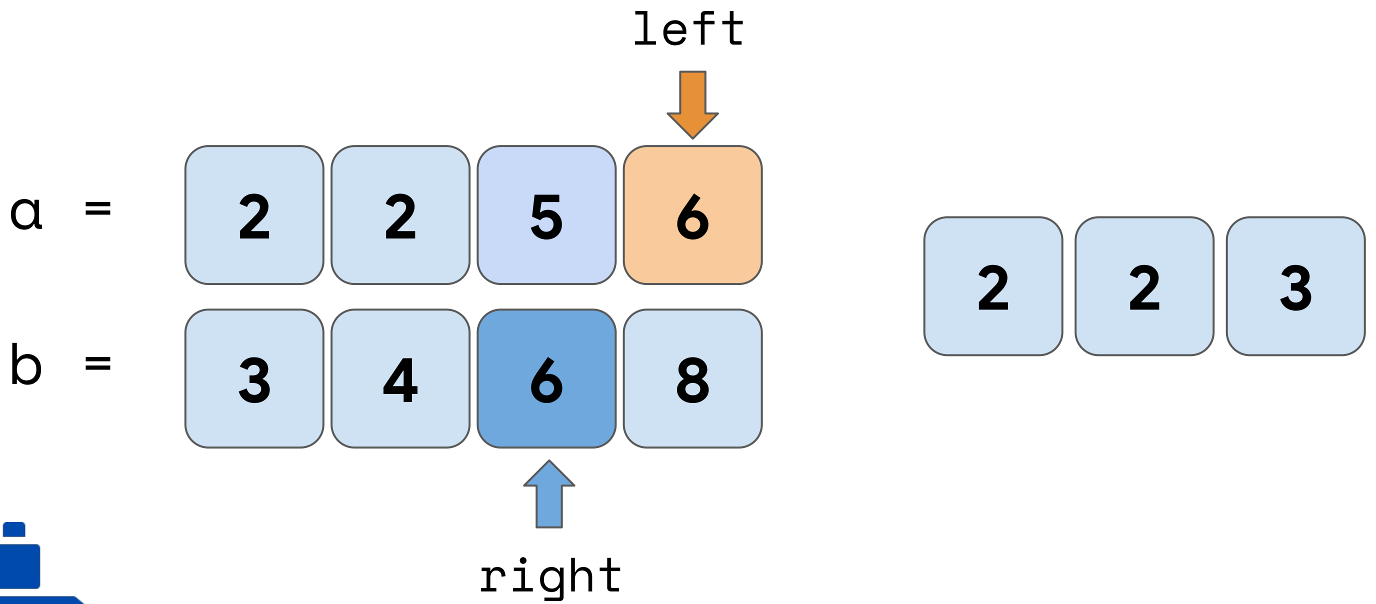
For-While Combo - Efficient



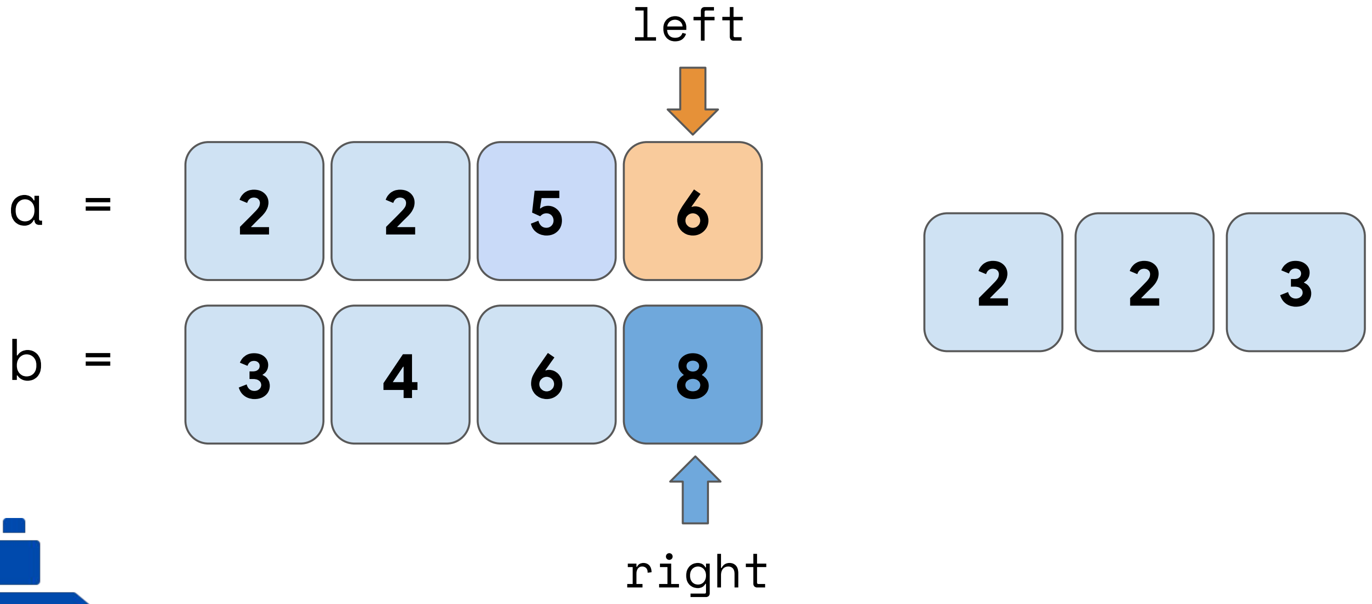
For-While Combo - Efficient



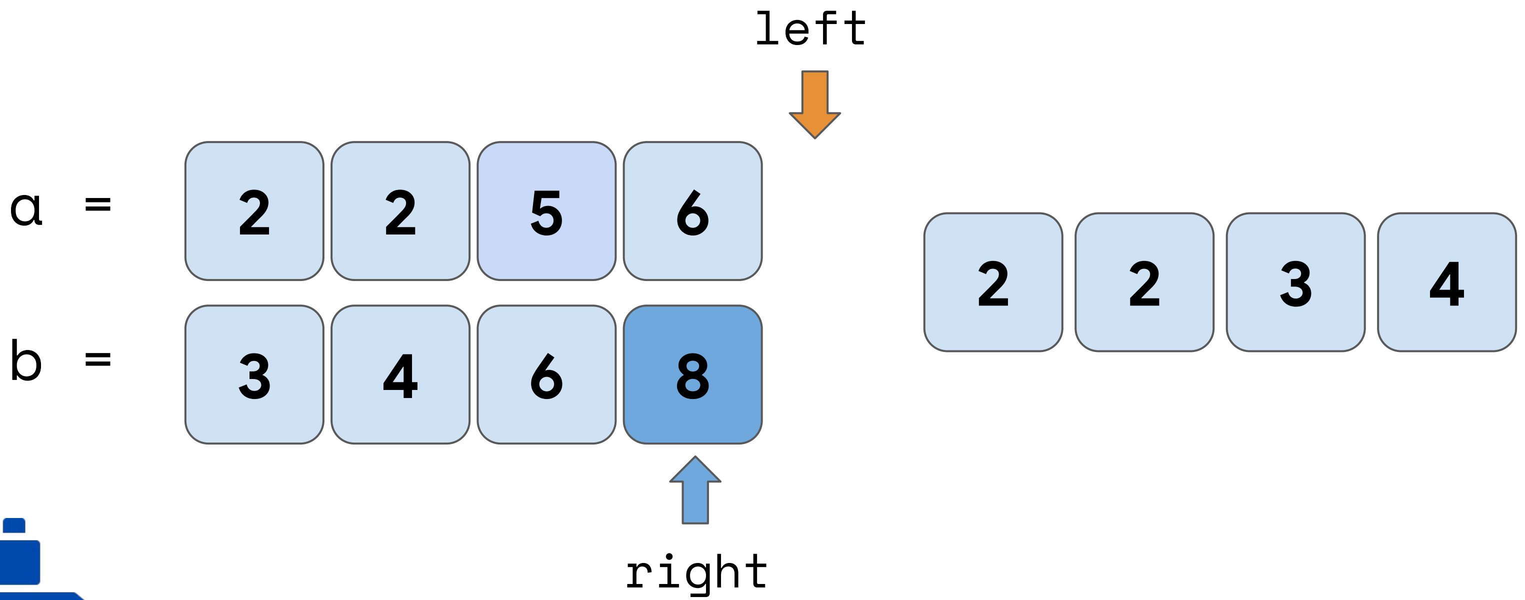
For-While Combo - Efficient



For-While Combo - Efficient



For-While Combo - Efficient



Practice

[Problem Link]

For-While combo - Implementation

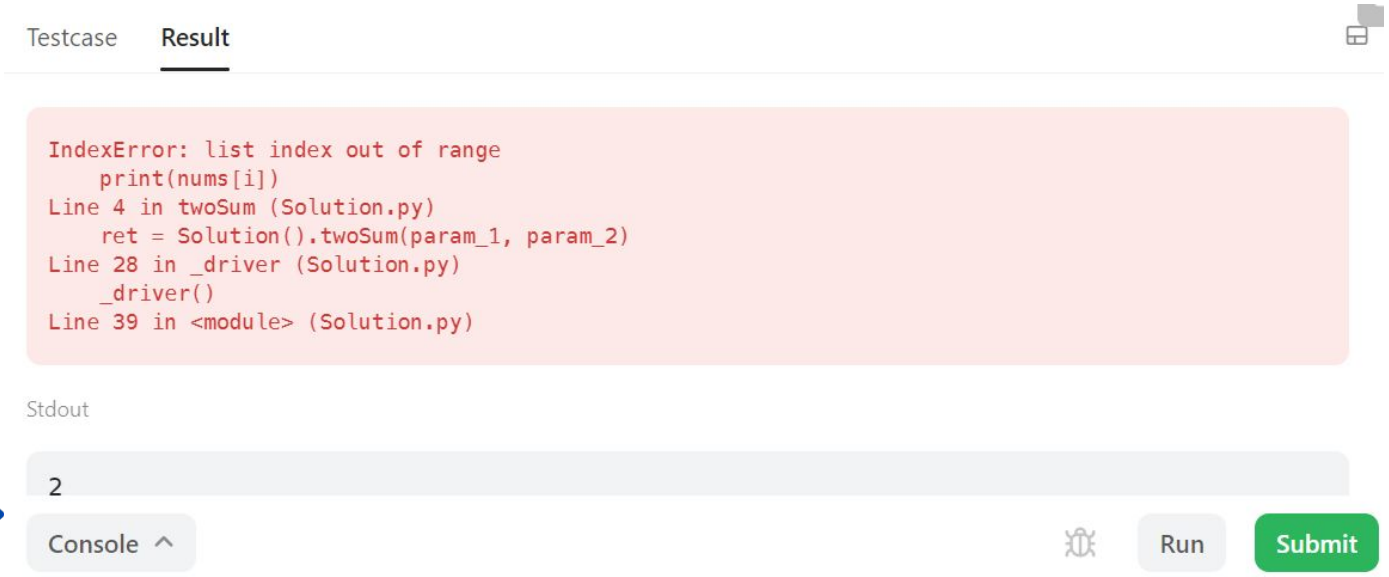
```python
def countSmaller(nums1, nums2):
    smaller_counts = []
    first = 0

    for second in range(len(nums2)):

        while first < len(nums1) and nums1[first] < nums2[second]:
            first += 1

        smaller_counts.append(first)

    return smaller_counts
```

Common Pitfalls

Africa To Silicon Valley

Common Pitfalls - Index out of bound

Trying to access elements using indices that are greater (or equal to) than the size of our array will result in this exception.



```
IndexError: list index out of range
    print(nums[i])
Line 4 in twoSum (Solution.py)
    ret = Solution().twoSum(param_1, param_2)
Line 28 in _driver (Solution.py)
    _driver()
Line 39 in <module> (Solution.py)
```

```
2
```

Common Pitfalls - Look out for pointer conditions

One common mistake we make while doing two pointer problems is not paying attention to our guard condition.

```
# left -> our left pointer
# right -> our right pointer
# what should our condition be? What else ıs wrong wıth t

      while left < right:
          pass

      while left <= right:
          pass
```

Checkpoint - Link

A2SV
Africa To Silicon Valley

Practice Problems

[Divide Players into Teams of Equal Skill]

[Boats to Save People]

[Container With Most Water]

[Sum of Square Numbers]

[Partition Labels]

[Merge Sorted Array]

[Remove Duplicates from Sorted Array]

[Rotate Array]

[Alternating Subsequence]

A2SV
Africa To Silicon Valley

Helpful resources

Codeforces Pilot Course

Geeksforgeeks

Algodaily

Quote of the day

In the end, it is not the year in your life that counts. It's the life in your years

Abraham Lincoln

A2SV
Africa To Silicon Valley