Solution Review: Problem Challenge 2

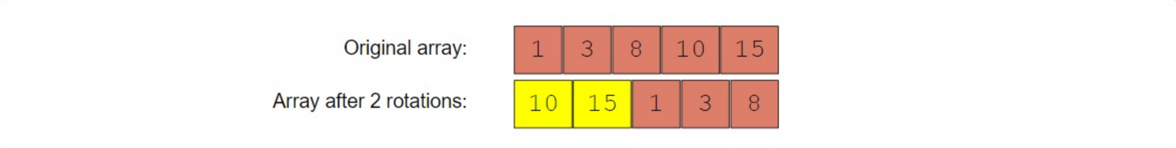
Search in Rotated Array (medium)

Given an array of numbers which is sorted in ascending order and also rotated by some arbitrary number, find if a given 'key' is present in it.

Write a function to return the index of the 'key' in the rotated array. If the 'key' is not present, return -1. You can assume that the given array does not have any duplicates.

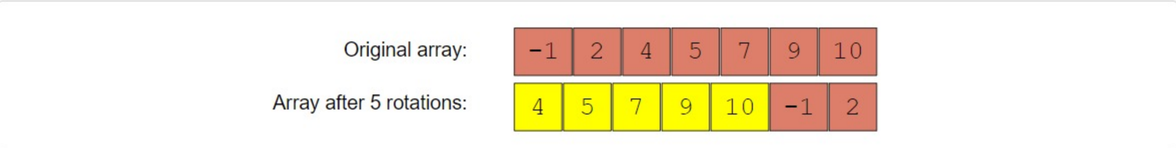
Example 1:

```
Input: [10, 15, 1, 3, 8], key = 15
Output: 1
Explanation: '15' is present in the array at index '1'.
```



Example 2:

```
Input: [4, 5, 7, 9, 10, -1, 2], key = 10
Output: 4
Explanation: '10' is present in the array at index '4'.
```



Solution

The problem follows the **Binary Search** pattern. We can use a similar approach as discussed in Order-agnostic Binary Search and modify it similar to Search Bitonic Array to search for the 'key' in the rotated array.

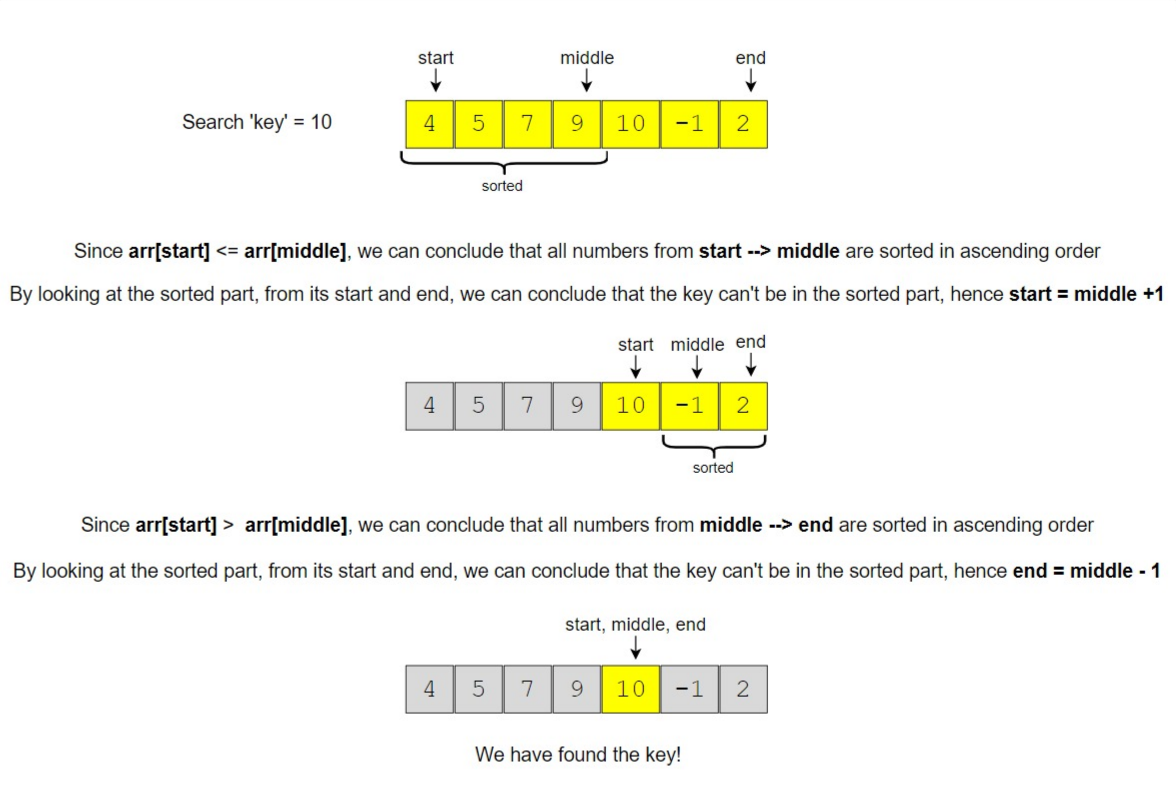
After calculating the `middle`, we can compare the numbers at indices `start` and `middle`. This will give us two options:

1. If `arr[start] <= arr[middle]`, the numbers from `start` to `middle` are sorted in ascending order.
2. Else, the numbers from `middle+1` to `end` are sorted in ascending order.

Once we know which part of the array is sorted, it is easy to adjust our ranges. For example, if option-1 is true, we have two choices:

1. By comparing the 'key' with the numbers at index `start` and `middle` we can easily find out if the 'key' lies between indices `start` and `middle`; if it does, we can skip the second part => `end = middle -1`.
2. Else, we can skip the first part => `start = middle + 1`.

Let's visually see this with the above-mentioned Example-2:



Since there are no duplicates in the given array, it is always easy to skip one part of the array in each iteration. However, if there are duplicates, it is not always possible to know which part is sorted. We will look into this case in the 'Similar Problems' section.

Code

Here is what our algorithm will look like:

Java | Python3 | C++ | JS

```python
def search_rotated_array(arr, key):
  start, end = 0, len(arr) - 1
  while start <= end:
    mid = start + (end - start) // 2
    if arr[mid] == key:
      return mid

    if arr[start] <= arr[mid]:  # left side is sorted in ascending order
      if key >= arr[start] and key < arr[mid]:
        end = mid - 1
      else:  # key > arr[mid]
        start = mid + 1
    else:  # right side is sorted in ascending order
      if key > arr[mid] and key <= arr[end]:
        start = mid + 1
      else:
        end = mid - 1

  # we are not able to find the element in the given array
  return -1


def main():
  print(search_rotated_array([10, 15, 1, 3, 8], 15))
  print(search_rotated_array([4, 5, 7, 9, 10, -1, 2], 10))

main()

```

Run | Save | Reset

Time complexity

Since we are reducing the search range by half at every step, this means that the time complexity of our algorithm will be $O(logN)$ where 'N' is the total elements in the given array.

Space complexity

The algorithm runs in constant space $O(1)$.

Similar Problems

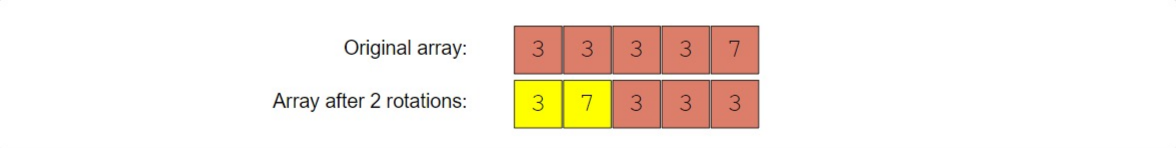
Problem 1

How do we search in a sorted and rotated array that also has duplicates?

The code above will fail in the following example!

Example 1:

```
Input: [3, 7, 3, 3, 3], key = 7
Output: 1
Explanation: '7' is present in the array at index '1'.
```



Solution

The only problematic scenario is when the numbers at indices `start`, `middle`, and `end` are the same, as in this case, we can't decide which part of the array is sorted. In such a case, the best we can do is to skip one number from both ends: `start = start + 1` & `end = end - 1`.

Code

The code is quite similar to the above solution. Only the highlighted lines have changed:

Java | Python3 | C++ | JS

```python
def search_rotated_with_duplicates(arr, key):
  start, end = 0, len(arr) - 1
  while start <= end:
    mid = start + (end - start) // 2
    if arr[mid] == key:
      return mid

    # the only difference from the previous solution,
    # if numbers at indexes start, mid, and end are same, we can't choose a side
    # the best we can do, is to skip one number from both ends as key != arr[mid]
    if arr[start] == arr[mid] and arr[end] == arr[mid]:
      start += 1
      end -= 1
    elif arr[start] <= arr[mid]:  # left side is sorted in ascending order
      if key >= arr[start] and key < arr[mid]:
        end = mid - 1
      else:  # key > arr[mid]
        start = mid + 1

    else:  # right side is sorted in ascending order
      if key > arr[mid] and key <= arr[end]:
        start = mid + 1
      else:
        end = mid - 1

  # we are not able to find the element in the given array
  return -1


def main():
  print(search_rotated_with_duplicates([3, 7, 3, 3, 3], 7))
```

Run | Save | Reset

Time complexity

This algorithm will run most of the times in $O(logN)$. However, since we only skip two numbers in case of duplicates instead of half of the numbers, the worst case time complexity will become $O(N)$.

Space complexity

The algorithm runs in constant space $O(1)$.