Image analytics

Upload an image

Drag and drop file here
Limit 200MB per file • PNG, JPG, JPEG

Browse files

12.5.png 69.7KB

Problem - Rotated Lists

We'll solve the following problem step-by-step:

> You are given list of numbers, obtained by rotating a sorted list an unknown number of times. Write a function to determine the minimum number of times the original sorted list was rotated to obtain the given list. Your function should have the worst-case complexity of `O(log N)`, where N is the length of the list. You can assume that all the numbers in the list are unique.
>
> Example: The list `[5, 6, 9, 0, 2, 3, 4]` was obtained by rotating the sorted list `[0, 2, 3, 4, 5, 6, 9]` 3 times.
>
> We define "rotating a list" as removing the last element of the list and adding it before the first element. E.g. rotating the list `[3, 2, 4, 1]` produces `[1, 3, 2, 4]`.
>
> "Sorted list" refers to a list where the elements are arranged in the increasing order e.g. `[1, 3, 5, 7]`.

Enter the text

1)solve the problem and write a code for me

GET RESPONSE

Here is a Python code that solves the problem:

```
def find_rotation_count(nums):
    """
    Finds the minimum number of times a sorted list was rotated to obtain the give

    Args:
    nums: A list of numbers.

    Returns:
    The minimum number of times the sorted list was rotated.
    """

    # Check if the list is empty.

    if not nums:
        return 0

    # Find the pivot index.

    pivot_index = find_pivot_index(nums)

    # If the pivot index is the first element, then the list is already sorted.

    if pivot_index == 0:
        return 0
```

```
    # Otherwise, the list is rotated.

    return pivot_index


def find_pivot_index(nums):
    """
    Finds the pivot index of a rotated sorted list.

    Args:
    nums: A list of numbers.

    Returns:
    The pivot index.
    """

    # Initialize the left and right indices.

    left = 0
    right = len(nums) - 1

    # While the left index is less than the right index.

    while left < right:

        # Calculate the mid index.

        mid = (left + right) // 2

        # If the mid element is greater than the right element.
```

```
        if nums[mid] > nums[right]:

            # Set the left index to the mid index plus one.

            left = mid + 1

        # Otherwise, the mid element is less than or equal to the right element.

        else:

            # Set the right index to the mid index.

            right = mid

    # Return the pivot index.

    return left
```