Daily Coding Problem #58

Problem

This problem was asked by Amazon.

An sorted array of integers was rotated an unknown number of times.

Given such an array, find the index of the element in the array in faster than linear time. If the element doesn't exist in the array, return null.

For example, given the array [13, 18, 25, 2, 8, 10] and the element 8, return 4 (the index of 8 in the array).

You can assume all the integers in the array are unique.

Solution

We can obviously do this problem in linear time if we looked at each element in the array. However, we need to do it faster than linear time. A big clue should be that the array of integers was previously sorted, and then rotated. If it was just sorted, we could do a binary search. However, this array was also rotated, so we can't do a regular binary search. We can modify it slightly to get to where we want it, however.

In our solution, we first find the rotation point using binary search. We do this by:

- Checking the midpoint for the rotation point (by comparing it to the previous number and seeing if it's larger)
- Moving our check up or down the array:

    - If the number we're looking at is larger than the first item in the array, then the rotation must occur later, so add `dist`
    - If not, then it must occur before, so subtract `dist`

- And then update dist by dividing it by 2 and taking its floor (so it's proper binary search).

Then, once we have the rotation point, we can do binary search as usual by remembering to offset the correct amount.

The code would look like this:

```
def shifted_array_search(lst, num):
    # First, find where the breaking point is in the shifted array
    i = len(lst) // 2
    dist = i // 2
    while True:
        if lst[0] > lst[i] and lst[i - 1] > lst[i]:
            break
        elif dist == 0:
            break
        elif lst[0] <= lst[i]:
            i = i + dist
        elif lst[i - 1] <= lst[i]:
            i = i - dist
        else:
            break
        dist = dist // 2

    # Now that we have the bottom, we can do binary search as usual,
    # wrapping around the rotation.
    low = i
    high = i - 1
    dist = len(lst) // 2
    while True:
        if dist == 0:
            return None


        guess_ind = (low + dist) % len(lst)
        guess = lst[guess_ind]
        if guess == num:
            return guess_ind

        if guess < num:
            low = (low + dist) % len(lst)
        if guess > num:
            high = (len(lst) + high - dist) % len(lst)

        dist = dist // 2
```

This solution runs in O(log n). However, this is definitely not the only solution! There are many other possible ways to implement this, but as long as you have the idea of doing binary search, you've got it.