Daily Coding Problem #311

Problem

This problem was asked by Sumo Logic.

Given a array that's sorted but rotated at some unknown pivot, in which all elements are distinct, find a "peak" element in `O(log N)` time.

An element is considered a peak if it is greater than both its left and right neighbors. It is guaranteed that the first and last elements are lower than all others.

Solution

Let us consider what will happen if we check a random element in the list. If this element is greater than both of its neighbors, we have magically found a solution, and can return it. Otherwise, there will be at least one neighbor with a higher value.

Suppose the higher element is on the right side. We now know it must be possible to find a peak on this side. Why? Because the last element is guaranteed to be lower than all the rest, and all the elements are distinct, so there must be some maximal element in between. By symmetry, the same is true for the left side.

However, we do not have to search one by one to find this maximal element. Note that by choosing the middle element in our first guess, we cut the search space in half. We can continue this binary search on subsequent guesses by updating our start and end indices.

Eventually, either the element will be found, or there will be only one element left, in which case we can return that element.

```
def peak(array):
    n = len(array)
    start, end = 0, n - 1

    while start + 1 <= end:
        mid = start + (end - start) // 2

        if array[mid - 1] < array[mid] > array[mid + 1]:
            return array[mid]
        elif array[mid] < array[mid + 1]:
            start = mid + 1
        else:
            end = mid - 1


    return array[start]
```

Since we are using binary search to search for a peak, this algorithm is logarithmic in the length of the input array. The space complexity is `O(1)`, since we only need to check values in the input array.