October 14

Problem 1: Find the Peak Element

Problem Statement:

A peak element in an array is an element that is strictly greater than its neighbors. Given an integer array nums, find a peak element, and return its index. If the array contains multiple peaks, return the index of any of the peaks.

You may imagine that nums[-1] = nums[n] = -∞ (i.e., the elements outside the array boundaries are considered to be -∞).

You must write an algorithm that runs in O(log n) time.

Link to problem:

https://leetcode.com/problems/find-peak-element/

Example 1:

- Input: nums = [1,2,3,1]
- Output: 2
- Explanation: 3 is a peak element, and its index is 2.

Example 2:

- Input: nums = [1,2,1,3,5,6,4]
- Output: 5
- Explanation: Your function can return either index 1 where the peak is 2, or index 5 where the peak is 6.

Solution:

```
class Solution {
   public int findPeakElement(int[] nums) {
      int left = 0, right = nums.length - 1;

      while (left < right) {
         int mid = left + (right - left) / 2;

         if (nums[mid] > nums[mid + 1]) {
            // The peak is in the left half
            right = mid;
         } else {
            // The peak is in the right half
            left = mid + 1;
```

```
        }
    }

    return left;
  }
}
```

Explanation:

- We use a binary search approach. At each step, we compare the middle element with its neighbor on the right (mid + 1).
- If nums[mid] > nums[mid + 1], the peak must be in the left half or at mid.
- If nums[mid] < nums[mid + 1], the peak must be in the right half.
- The process continues until the left and right pointers converge to the peak element.

Time Complexity: O(log n) (Binary Search)

Space Complexity: O(1)

Problem 2: Find Minimum in Rotated Sorted Array

Problem Statement:

Suppose an array of length n sorted in ascending order is **rotated** between 1 and n times. For example, the array nums = [0,1,2,4,5,6,7] might become:

- [4,5,6,7,0,1,2] if it was rotated 4 times.
- [0,1,2,4,5,6,7] if it was rotated 7 times.

Notice that **rotating** an array [a[0], a[1], a[2], ..., a[n-1]] 1 time results in the array [a[n-1], a[0], a[1], a[2], ..., a[n-2]].

Given the sorted rotated array nums of **unique** elements, return *the minimum element of this array*.

You must write an algorithm that runs in O(log n) time.

Link to problem:

https://leetcode.com/problems/find-minimum-in-rotated-sorted-array/description/

Example 1:

- Input: nums = [3,4,5,1,2]
- Output: 1
- Explanation: The original array was [1,2,3,4,5] rotated 3 times.

Example 2:

- Input: nums = [4,5,6,7,0,1,2]
- Output: 0
- Explanation: The original array was [0,1,2,4,5,6,7] rotated 4 times.

Solution:

```
class Solution {
    public int findMin(int[] nums) {
        int left = 0, right = nums.length - 1;

        while (left < right) {
            int mid = left + (right - left) / 2;

            if (nums[mid] > nums[right]) {
                // The minimum is in the right half
                left = mid + 1;
            } else {
                // The minimum is in the left half (including mid)
                right = mid;
            }
        }

        return nums[left];
    }
}
```

Explanation:

- Since the array is rotated, we know the smallest element is the inflection point where the rotation happens.
- We use binary search to find the minimum element.
- If nums[mid] > nums[right], it means the minimum is in the right half.
- Otherwise, the minimum is in the left half.
- We continue the search until left == right, which gives us the index of the minimum element.

Time Complexity: O(log n)

Space Complexity: O(1)
