Assignment-11

13 June 2023 16:10

💡 **Question 1**

Given a non-negative integer `x`, return *the square root of* `x` *rounded down to the nearest integer*. The returned integer should be **non-negative** as well.

You **must not use** any built-in exponent function or operator.

- For example, do not use `pow(x, 0.5)` in c++ or `x ** 0.5` in python.

Example 1:

Input: x = 4 Output: 2 Explanation: The square root of 4 is 2, so we return 2.

Example 2:

Input: x = 8

Output: 2

Explanation: The square root of 8 is 2.82842..., and since we round it down to the nearest integer, 2 is returned.

```
class Solution(object):
  def mySqrt(self, x):
    s=0
    e=x
    while s<=e:
      mid=(s+e)>>1
      val=mid*mid
      if val==x:
        return mid
      elif val>x:
        e=mid-1
      else:
        s=mid+1
    return int(ceil(s-1))
```

💡 **Question 2**

A peak element is an element that is strictly greater than its neighbors.

Given a **0-indexed** integer array `nums`, find a peak element, and return its index. If the array contains multiple peaks, return the index to **any of the peaks**.

You may imagine that `nums[-1] = nums[n] = -∞`. In other words, an element is always considered to be strictly greater than a neighbor that is outside the array.

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

Example 1:

Input: nums = [1,2,3,1]

Output: 2

Explanation: 3 is a peak element and your function should return the index number 2.
Example 2:

Input: nums = [1,2,1,3,5,6,4]

Output: 5

Explanation: Your function can return either index number 1 where the peak element is 2, or index number 5 where the peak element is 6.

```
class Solution(object):
  def findPeakElement(self, nums):
    s=0
    e=len(nums)-1
    while s<=e:
      mid=(s+e)>>1


      if mid>0 and nums[mid]<nums[mid-1]:
        e=mid-1
      elif mid<len(nums)-1 and nums[mid]<nums[mid+1]:
        s=mid+1
      else:
        return mid
```

💡 **Question 3**

Given an array `nums` containing `n` distinct numbers in the range `[0, n]`, return *the only number in the range that is missing from the array.*

Example 1:

Input: nums = [3,0,1]

Output: 2

Explanation: n = 3 since there are 3 numbers, so all numbers are in the range [0,3]. 2 is the missing number in the range since it does not appear in nums.

Example 2:
Input: nums = [0,1]

Output: 2

Explanation: n = 2 since there are 2 numbers, so all numbers are in the range [0,2]. 2 is the missing number in the range since it does not appear in nums.

```
class Solution(object):
    def missingNumber(self, nums):
        n=len(nums)
        n_sum=n*(n+1)/2
        actual_sum=0
        for num in nums:
            actual_sum+=num
        return n_sum-actual_sum
```

💡 **Question 4**

Given an array of integers `nums` containing `n + 1` integers where each integer is in the range `[1, n]` inclusive.

There is only **one repeated number** in `nums`, return *this repeated number*.

You must solve the problem **without** modifying the array `nums` and uses only constant extra space.

Example 1:

Input: nums = [1,3,4,2,2]

Output: 2

Example 2:

Input: nums = [3,1,3,4,2]

Output: 3

```
class Solution(object):
    def findDuplicate(self, nums):
        for num in nums:
            index=abs(num)
            if nums[index]<0:
                return index
            else:
                nums[index]=-nums[index]
        return -1
```

💡 **Question 5**

Given two integer arrays `nums1` and `nums2`, return *an array of their intersection*. Each element in the result must be **unique** and you may return the result in **any order**.

Example 1:

Input: nums1 = [1,2,2,1], nums2 = [2,2]

Output: [2]

Example 2:
Input: nums1 = [4,9,5], nums2 = [9,4,9,8,4]

Output: [9,4]

Explanation: [4,9] is also accepted.

```
class Solution(object):
    def intersection(self, nums1, nums2):
        mp={}
        ans=[]
        for num in nums2:
            if num in mp:
                mp[num]+=1
            else:
                mp[num]=1


        for num in nums1:
            if num in mp and mp[num]!=-1:
                ans.append(num)
                mp[num]=-1
        return ans
```

Question 6

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.

Example 1:

Input: nums = [3,4,5,1,2]

Output: 1

Explanation: The original array was [1,2,3,4,5] rotated 3 times.

Input: nums = [4,5,6,7,0,1,2]

Output: 0

Explanation: The original array was [0,1,2,4,5,6,7] and it was rotated 4 times.

Input: nums = [11,13,15,17]

Output: 11

Explanation: The original array was [11,13,15,17] and it was rotated 4 times.

```
class Solution(object):
  def findMin(self, nums):
    start=0
    end=len(nums)-1
    mini=99999
    while start<=end:
      mid=(start+end)>>1
      if nums[start]<=nums[mid]:
        mini=min(mini,nums[start])
        start=mid+1
      else:
        mini=min(mini,nums[mid])
        end=mid-1
    return mini
```

Question 7

Given an array of integers nums sorted in non-decreasing order, find the starting and ending position of a given target value.

If target is not found in the array, return [-1, -1].

You must write an algorithm with O(log n) runtime complexity.

Example 1:

Input: nums = [5,7,7,8,8,10], target = 8

Output: [3,4]

Example 2:

Input: nums = [5,7,7,8,8,10], target = 6

Output: [-1,-1]

Example 3:

Input: nums = [], target = 0

Output: [-1,-1]

```
class Solution(object):


  def searchRange(self, nums, target):
    def firstOccur(nums,target):
      s=0
      e=len(nums)-1
      ans=-1
      while s<=e:
        mid=(s+e)>>1


        if(nums[mid]==target):
          ans=mid
          e=mid-1
        elif nums[mid]>target:
          e=mid-1
```

```
            else:
                s=mid+1
        return ans

    def lastOccur(nums,target):
        ans=-1
        s=0
        e=len(nums)-1

        while s<=e:
            mid=(s+e)>>1

            if(nums[mid]==target):
                ans=mid
                s=mid+1
            elif nums[mid]>target:
                e=mid-1
            else:
                s=mid+1
        return ans
    return [firstOccur(nums,target),lastOccur(nums,target)]
```

💡 **Question 8**

Given two integer arrays `nums1` and `nums2`, return *an array of their intersection*. Each element in the result must appear as many times as it shows in both arrays and you may return the result in **any order**.

Example 1:

Input: nums1 = [1,2,2,1], nums2 = [2,2]

Output: [2,2]

Example 2:

Input: nums1 = [4,9,5], nums2 = [9,4,9,8,4]

Output: [4,9]

Explanation: [9,4] is also accepted.

```
class Solution(object):
    def intersect(self, nums1, nums2):
        mp={}
        for num in nums1:
            if num in mp:
                mp[num]+=1
            else:
                mp[num]=1
        ans=[]
        for num in nums2:
            if num in mp and mp[num]>0:
                mp[num]-=1
                ans.append(num)
        return ans
```