💡 \*\*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.

**def** mySqrt(x):

**if** x **==** 0:

**return** 0

left, right **=** 1, x

**while** left **<=** right:

mid **=** left **+** (right **-** left) **//** 2

**if** mid **\*** mid **>** x:

right **=** mid **-** 1

**else**:

left **=** mid **+** 1

**return** left **-** 1

*# Example usage*

x **=** 4

result **=** mySqrt(x)

print(result)

#output- 2

💡 \*\*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.

**def** findPeakElement(nums):

left, right **=** 0, len(nums) **-** 1

**while** left **<** right:

mid **=** left **+** (right **-** left) **//** 2

**if** nums[mid] **<** nums[mid **+** 1]:

left **=** mid **+** 1

**else**:

right **=** mid

**return** left

*# Example usage*

nums **=** [1, 2, 3, 1]

result **=** findPeakElement(nums)

print(result)

#output- 2

💡 \*\*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.

**def** missingNumber(nums):

n **=** len(nums)

expected\_sum **=** n **\*** (n **+** 1) **//** 2

actual\_sum **=** sum(nums)

**return** expected\_sum **-** actual\_sum

*# Example usage*

nums **=** [3, 0, 1]

result **=** missingNumber(nums)

print(result)

#output- 2

💡 \*\*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

**def** findDuplicate(nums):

slow **=** fast **=** nums[0]

**while** **True**:

slow **=** nums[slow]

fast **=** nums[nums[fast]]

**if** slow **==** fast:

**break**

slow **=** nums[0]

**while** slow **!=** fast:

slow **=** nums[slow]

fast **=** nums[fast]

**return** slow

*# Example usage*

nums **=** [1, 3, 4, 2, 2]

result **=** findDuplicate(nums)

print(result)

#output- 2

💡 \*\*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]

**def** intersection(nums1, nums2):

set1 **=** set(nums1)

set2 **=** set(nums2)

**return** list(set1**.**intersection(set2))

*# Example 1*

nums1 **=** [1, 2, 2, 1]

nums2 **=** [2, 2]

print(intersection(nums1, nums2)) *# Output: [2]*

*# Example 2*

nums1 **=** [4, 9, 5]

nums2 **=** [9, 4, 9, 8, 4]

print(intersection(nums1, nums2)) *# Output: [9, 4]*

💡 \*\*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.

**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] and it was rotated 4 times.

**def** findMin(nums):

left, right **=** 0, len(nums) **-** 1

**while** left **<** right:

mid **=** left **+** (right **-** left) **//** 2

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

left **=** mid **+** 1

**else**:

right **=** mid

**return** nums[left]

*# Example 1*

nums **=** [3, 4, 5, 1, 2]

print(findMin(nums)) *# Output: 1*

*# Example 2*

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

print(findMin(nums)) *# Output: 0*

💡 \*\*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]

**def** searchRange(nums, target):

left **=** findFirstOccurrence(nums, target)

right **=** findLastOccurrence(nums, target)

**return** [left, right]

**def** findFirstOccurrence(nums, target):

left, right **=** 0, len(nums) **-** 1

index **=** **-**1

**while** left **<=** right:

mid **=** left **+** (right **-** left) **//** 2

**if** nums[mid] **>=** target:

right **=** mid **-** 1

**if** nums[mid] **==** target:

index **=** mid

**else**:

left **=** mid **+** 1

**return** index

**def** findLastOccurrence(nums, target):

left, right **=** 0, len(nums) **-** 1

index **=** **-**1

**while** left **<=** right:

mid **=** left **+** (right **-** left) **//** 2

**if** nums[mid] **<=** target:

left **=** mid **+** 1

**if** nums[mid] **==** target:

index **=** mid

**else**:

right **=** mid **-** 1

**return** index

*# Example 1*

nums **=** [5, 7, 7, 8, 8, 10]

target **=** 8

print(searchRange(nums, target)) *# Output: [3, 4]*

*# Example 2*

nums **=** [5, 7, 7, 8, 8, 10]

target **=** 6

print(searchRange(nums, target)) *# Output: [-1, -1]*

💡 \*\*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]

**import** collections

**def** intersect(nums1, nums2):

counter1 **=** collections**.**Counter(nums1)

counter2 **=** collections**.**Counter(nums2)

**return** list((counter1 **&** counter2)**.**elements())

*# Example 1*

nums1 **=** [1, 2, 2, 1]

nums2 **=** [2, 2]

print(intersect(nums1, nums2)) *# Output: [2*

#output- [2, 2]