Q.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.

Solution :

def mySqrt(x):

if x == 0:

return 0

left, right = 1, x

while left <= right:

mid = (left + right) // 2

if mid \* mid == x:

return mid

elif mid \* mid < x:

left = mid + 1

else:

right = mid - 1

return right

Q.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.

Solution :

def findPeakElement(nums):

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

while left < right:

mid = (left + right) // 2

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

left = mid + 1

else:

right = mid

return left

Q.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.*

Solution :

def missingNumber(nums):

n = len(nums)

missing = n

for i in range(n):

missing ^= i ^ nums[i]

return missing

Q.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.

Solution :

def findDuplicate(nums):

left = 1

right = len(nums) - 1

while left <= right:

mid = (left + right) // 2

count = sum(num <= mid for num in nums)

if count > mid:

right = mid - 1

else:

left = mid + 1

return left

Q.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**.

Solution :

def intersection(nums1, nums2):

set1 = set(nums1)

intersection = []

for num in nums2:

if num in set1 and num not in intersection:

intersection.append(num)

return intersection

Q.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.

Solution :

def findMin(nums):

left = 0

right = len(nums) - 1

while left < right:

mid = (left + right) // 2

if nums[mid] > nums[right]:

left = mid + 1

else:

right = mid

return nums[left]

Q.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.

Solution :

def searchRange(nums, target):

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

start, end = -1, -1

while left <= right:

mid = (left + right) // 2

if nums[mid] == target:

start = mid

end = mid

break

elif nums[mid] < target:

left = mid + 1

else:

right = mid - 1

if start != -1:

while start > 0 and nums[start - 1] == target:

start -= 1

while end < len(nums) - 1 and nums[end + 1] == target:

end += 1

return [start, end]

Q.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**.

Solution :

from collections import defaultdict

def intersect(nums1, nums2):

freqMap = defaultdict(int)

for num in nums1:

freqMap[num] += 1

intersection = []

for num in nums2:

if num in freqMap and freqMap[num] > 0:

intersection.append(num)

freqMap[num] -= 1

return intersection