**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.**

**Ans.**

**def** floorSqrt(x):

**if** (x **==** 0 **or** x **==** 1):

**return** x

    start **=** 1

    end **=** x**//**2

**while** (start <**=** end):

        mid **=** (start **+** end) **//** 2

**if** (mid**\***mid **==** x):

**return** mid

**if** (mid **\*** mid < x):

            start **=** mid **+** 1

            ans **=** mid

**else**:

            end **=** mid**-**1

**return** ans

**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.**

**Ans.** Solution from my Leetcode :- <https://leetcode.com/gopsa2001/>

class Solution:

    def findPeakElement(self, nums: List[int]) -> int:

        left, right = 0, len(nums)-1

        while left < right:

            mid = (left + right) // 2

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

                right = mid

            else:

                left = mid + 1

        return left

**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.

**Ans.** Solution from my Leetcode :- <https://leetcode.com/gopsa2001/>

class Solution:

    def missingNumber(self, nums: List[int]) -> int:

        nums.sort()

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

        while left <= right:

            middle = (left + right) // 2

            if nums[middle] == middle:

                left = middle + 1

            else:

                right = middle - 1

        return left

**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

**Ans.** Solution from my Leetcode :- <https://leetcode.com/gopsa2001/>

class Solution:

    def findDuplicate(self, nums: List[int]) -> int:

        nums.sort()

        l,r = 0,len(nums)-1

        while l<=r:

            mid = (l+r)//2

            if nums[mid]<mid+1:

                r = mid - 1

            else:

                l = mid + 1

        return l

**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]

**Ans.** Solution from my Leetcode :- <https://leetcode.com/gopsa2001/>

class Solution:

    def intersection(self, nums1: List[int], nums2: List[int]) -> List[int]:

        x=set(nums1)

        y=set(nums2)

        return list(x.intersection(y))

**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.

**Ans.** Solution from my Leetcode :- <https://leetcode.com/gopsa2001/>

class Solution:

    def findMin(self, nums: List[int]) -> int:

        low=0

        high=len(nums)-1

        res=nums[0]

        while low<=high:

            if nums[low]<nums[high]:

                res=min(res,nums[low])

                break

            mid=(low+high)//2

            res=min(res,nums[mid])

            if nums[mid]>=nums[low]:

                low=mid+1

            else:

                high=mid-1

        return res

**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]

**Ans.**

def searchRange(nums, target):

start = -1

end = -1

left = 0

right = len(nums) - 1

while left <= right:

mid = (left + right) // 2

if nums[mid] == target:

start = mid

end = mid

break

if nums[mid] > target:

right = mid - 1

else:

left = mid + 1

if start == -1:

return [-1, -1]

left = 0

right = start - 1

while left <= right:

mid = (left + right) // 2

if nums[mid] == target:

start = mid

right = mid - 1

else:

left = mid + 1

left = end + 1

right = len(nums) - 1

while left <= right:

mid = (left + right) // 2

if nums[mid] == target:

end = mid

left = mid + 1

else:

right = mid - 1

return [start, end]

**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]

**Ans.** Solution from my Leetcode :- <https://leetcode.com/gopsa2001/>

class Solution:

    def intersect(self, nums1: List[int], nums2: List[int]) -> List[int]:

        nums1.sort()

        nums2.sort()

        n1 = len(nums1)

        n2 = len(nums2)

        nums1.append(inf)

        nums2.append(inf)

        i = j = 0

        ans = []

        while i != n1 and j != n2:

            if nums1[i] == nums2[j]:

                ans.append(nums1[i])

                i += 1

                j += 1

            elif nums1[i] < nums2[j]:

                i += 1

            else:

                j+=1

        return ans