Q1

class Solution:

    def mySqrt(self, x: int) -> int:

        left,right=1,x

        while left<=right:

            mid=(left+right)//2

            if mid\*mid==x:

                return mid

            if mid\*mid>x:

                right=mid-1

            else:

                left=mid+1

        return right

Q2

class Solution:

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

n = len(nums)

l, r = 0, n - 1

while l < r:

mid = (l + r) >> 1

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

l = mid + 1

else:

r = mid

return l

Q3

class Solution:

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

m = 0

for i in range(len(nums)+1):

if i in nums:

continue

else:

m+=i

return m

Q4

class Solution:

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

d=defaultdict(lambda:0)

for num in nums:

if d[num]:

return num

else:

d[num]=1

Q5

class Solution:

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

if len(nums1) < len(nums2):

nums1,nums2 = nums2,nums1

res = []

nums1 = sorted(nums1)

nums2 = set(nums2)

for i in nums2:

l,r = 0,len(nums1)-1

while l <=r:

m = (l+r)>>1

if nums1[m] == i:

res.append(i)

break

else:

if nums1[m] < i:

l = m + 1

else:

r = m - 1

return res

Q6

class Solution:

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

        hi, lo = len(nums) - 1, 0

        while hi - 1 > lo:

            mid = (hi + lo)//2

            if nums[lo] > nums[mid]:

                hi = mid

            else:

                lo = mid

        if nums[hi] > nums[lo]:

            return nums[0]

        return nums[hi]

Q7

class Solution:

def lower\_bound(self, nums: List[int], target: int) -> int:

l, r = 0, len(nums) - 1

while l <= r:

mid = l + (r - l) //2

if nums[mid] < target:

l = mid + 1

else:

r = mid-1

return l if (0 <= l < len(nums)) and (nums[l] == target) else -1

def upper\_bound(self, nums: List[int], target: int) -> int:

l, r = 0, len(nums) - 1

while l <= r:

mid = l + r - l //2

if nums[mid] <= target:

l = mid + 1

else:

r = mid-1

return r if (0 <= r < len(nums)) and nums[r] == target else -1

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

return [self.lower\_bound(nums, target), self.upper\_bound(nums, target)]

Q8

class Solution:

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

count1 = {}

count2 = {}

for num in nums1:

count1[num] = count1.get(num, 0) + 1

for num in nums2:

count2[num] = count2.get(num, 0) + 1

arr = []

for num in count1:

if num in count2:

count = min(count1[num], count2[num])

arr.extend([num] \* count)

return arr