ASSIGMNENT -11

BINARY SEARCH

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.

SOLUTIONS: TC: O(log n), SC: O(1)

CODE:

```
class Solution:

    def mySqrt(self, x: int) -> int:
        f,l = 0, x
        res = 0
        while(f<=l):
            mid = f + ((l-f)//2)
            if mid**2 > x:
                l = mid-1
            elif mid**2< x:
                f = mid+1
                res = mid
            else:
                return mid

        return res
```

QUESTION 2:

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

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 neighbour that is outside the array.

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

SOLUTIONS: TC: O(log n), SC: O(1)

CODE:

```
class Solution:

    def findPeakElement(self, nums: List[int]) -> int:
        l,r = 0, len(nums)-1
        while(l<=r):
            m = l + ((r-l) //2)
            # if left neighbour is greater
            if m > 0 and nums[m] < nums[m-1]:
                r = m -1
            # if right neighbour is greater
            elif m < len(nums)-1 and nums[m] < nums[m+1]:
                l = m+1
            else:
                return m
```

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.

SOLUTIONS: TC: O(n), SC: O(1)

CODE:

```
class Solution:
    def missingNumber(self, nums: List[int]) -> int:
        res = len(nums)

        for i in range(len(nums)):
            res += (i - nums[i])

        return res
```

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.

SOLUTIONS: TC: O(n), SC: O(1)

CODE:

```
class Solution:

    def findDuplicate(self, nums: List[int]) -> int:
        slow, fast = 0,0
        while True: # we keep going until they meet -> find the intersection
            slow = nums[slow]
            fast = nums[nums[fast]]
            if slow == fast:
                break

        slow2 = 0
        while True:
            slow = nums[slow]
            slow2 = nums[slow2]
            if slow == slow2:
                return slow
```

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**.

SOLUTIONS: TC: O(m+n), SC: O(min(n,m))

CODE:

```
class Solution:

    def intersection(self, nums1: List[int], nums2: List[int]) -> List[int]:
        # x = set(nums1)
        # y = set(nums2)
        # return list(x.intersection(y))
      return set(nums1) & set(nums2)
```

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.`

SOLUTION: TC: O(log n) , SC: O(1)

CODE:

```
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.

SOLUTION: TC: O(log n) , SC: O(1)

CODE:

```
class Solution:
    def searchRange(self, nums: List[int], target: int) -> List[int]:
        left = self.binarySearch(nums,target,True)
        right = self.binarySearch(nums,target,False)
        return [left,right]

    def binarySearch(self,nums,target,leftbias):
        l,r = 0, len(nums)-1
        i = -1
        while l <= r:
```

```
            mid = (l+r)//2
            if nums[mid] < target:
                l = mid+1
            elif nums[mid] > target:
                r = mid -1
            else:
                i = mid
                if leftbias:
                    r = mid -1
                else:
                    l = mid + 1
        return i
```

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**.

SOLUTION: TC: O(n+m), SC: O(n)

```
class Solution:
    def intersect(self, nums1: List[int], nums2: List[int]) -> List[int]:
        # result = []
        # for i in nums1:
        #     if i in nums2:
        #         result.append(i)
        #         nums2.remove(i)
        # return result
        #Approach -2 TC: O(n+m), SC: O(n)
        c = Counter(nums1)
        result = []
        for n in nums2:
            if c[n]>0:
                result.append(n)
                c[n] -= 1    #to avoid duplicates
        return result
```