Assignment 11 Solutions

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.

In [1]:
```
def mySqrt(x):
    if x == 0:
        return 0

    left, right = 1, x
    while left <= right:
        mid = left + (right - left) // 2
        if mid * mid == x:
            return mid
        elif mid * mid < x:
            left = mid + 1
        else:
            right = mid - 1

    return right
```

In [2]:
```
print(mySqrt(4))
print(mySqrt(8))
```

```
2
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.

Example 2:

Input: nums = [1,2,1,3,5,6,4]

Output: 5

Explanation: Your function can return either index number 1 where the peak element is 2, or index number 5 where the peak element is 6.

In [3]:
```python
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
```

In [5]:
```python
print(findPeakElement([1, 2, 3, 1]))  # Output: 2
print(findPeakElement([1, 2, 1, 3, 5, 6, 4]))  # Output: 5
```

```
2
5
```

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.

Example 3:

Input: nums = [9,6,4,2,3,5,7,0,1]

Output: 8

Explanation: n = 9 since there are 9 numbers, so all numbers are in the range [0,9]. 8 is the missing number in the range since it does not appear in nums.

In [6]:
```python
def missingNumber(nums):
    missing = len(nums)  # Initialize missing as the last element in the range [0, n]

    for i, num in enumerate(nums):
        missing ^= i ^ num  # XOR the index and the element with missing

    return missing
```

In [7]:
```python
print(missingNumber([3, 0, 1]))  # Output: 2
print(missingNumber([0, 1]))  # Output: 2
print(missingNumber([9, 6, 4, 2, 3, 5, 7, 0, 1]))  # Output: 8
```

```
2
2
8
```

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

Example 2:

Input: nums = [3,1,3,4,2]

Output: 3

In [8]:
```
def findDuplicate(nums):
    slow = fast = nums[0]  # Start with the first element as both slow and fast pointers

    # Move slow and fast pointers until they meet in the cycle
    while True:
        slow = nums[slow]  # Move slow pointer by one step
        fast = nums[nums[fast]]  # Move fast pointer by two steps

        if slow == fast:
            break

    # Move the slow pointer to the beginning of the array
    slow = nums[0]

    # Move both pointers at the same speed until they meet at the entrance of the cycle
    while slow != fast:
        slow = nums[slow]
        fast = nums[fast]

    return slow
```

In [9]:
```
print(findDuplicate([1, 3, 4, 2, 2]))  # Output: 2
print(findDuplicate([3, 1, 3, 4, 2]))  # Output: 3
```

```
2
3
```

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]

Example 2:

Input: nums1 = [4,9,5], nums2 = [9,4,9,8,4]

Output: [9,4]

Explanation: [4,9] is also accepted.

In [10]:
```
def intersection(nums1, nums2):
    set1 = set(nums1)  # Convert nums1 to a set
    set2 = set(nums2)  # Convert nums2 to a set

    return list(set1.intersection(set2))  # Find the intersection of the two sets and convert it back to a list
```

In [11]:
```
print(intersection([1, 2, 2, 1], [2, 2]))  # Output: [2]
print(intersection([4, 9, 5], [9, 4, 9, 8, 4]))  # Output: [9, 4]
```

```
[2]
[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.

Example 3:

Input: nums = [11,13,15,17]

Output: 11

Explanation: The original array was [11,13,15,17] and it was rotated 4 times.

In [12]:
```
def findMin(nums):
    left, right = 0, len(nums) - 1

    while left < right:
        mid = left + (right - left) // 2

        if nums[mid] < nums[right]:
            right = mid
        else:
            left = mid + 1

    return nums[left]
```

In [13]:
```
print(findMin([3, 4, 5, 1, 2]))  # Output: 1
print(findMin([4, 5, 6, 7, 0, 1, 2]))  # Output: 0
print(findMin([11, 13, 15, 17]))  # Output: 11
```

```
1
0
11
```

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]

Example 3:

Input: nums = [], target = 0

Output: [-1,-1]

In [15]:
```
def searchRange(nums, target):
```

In [15]:
```python
def searchRange(nums, target):
    def findLeftmost(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
            else:
                left = mid + 1

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

        return index

    def findRightmost(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
            else:
                right = mid - 1

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

        return index

    leftmost = findLeftmost(nums, target)
    rightmost = findRightmost(nums, target)

    return [leftmost, rightmost]
```

In [16]:
```python
print(searchRange([5, 7, 7, 8, 8, 10], 8))  # Output: [3, 4]
print(searchRange([5, 7, 7, 8, 8, 10], 6))  # Output: [-1, -1]
print(searchRange([], 0))  # Output: [-1, -1]
```

```
[3, 4]
[-1, -1]
[-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]

Example 2:

Input: nums1 = [4,9,5], nums2 = [9,4,9,8,4]

Output: [4,9]

Explanation: [9,4] is also accepted.

In [17]:
```python
from collections import Counter

def intersect(nums1, nums2):
    freq = Counter(nums1)
    result = []

    for num in nums2:
        if freq.get(num, 0) > 0:
            result.append(num)
            freq[num] -= 1

    return result
```

In [18]:
```python
print(intersect([1, 2, 2, 1], [2, 2]))  # Output: [2, 2]
```

```
print(intersect([4, 9, 5], [9, 4, 9, 8, 4]))  # Output: [4, 9]
```

```
[2, 2]
[9, 4]
```

In [ ]: