SAVEETHA SCHOOL OF ENGINEERING

DEPARTMENT OF COMPUTERSCIENCE AND ENGINEERING

CSA0889 – Python Programming

Assignment - 6

1. Given two sorted arrays nums1 and nums2 of size m and n respectively, return the median of the two sorted arrays.

The overall run time complexity should be O(log (m+n)).

Example 1:

Input: nums1 = [1,3], nums2 = [2]

Output: 2.00000

Explanation: merged array = [1,2,3] and median is 2.

Example 2:

Input: nums1 = [1,2], nums2 = [3,4]

Output: 2.50000

main.py

```python
def findMedianSortedArrays(nums1, nums2):
    if len(nums1) > len(nums2):
        nums1, nums2 = nums2, nums1
    m, n = len(nums1), len(nums2)
    imin, imax, half_len = 0, m, (m + n + 1) // 2
    while imin <= imax:
        i = (imin + imax) // 2
        j = half_len - i
        if i < m and nums1[i] < nums2[j-1]:
            imin = i + 1
        elif i > 0 and nums1[i-1] > nums2[j]:
            imax = i - 1
        else:
            if i == 0: max_of_left = nums2[j-1]
            elif j == 0: max_of_left = nums1[i-1]
            else: max_of_left = max(nums1[i-1], nums2[j-1])
            if (m + n) % 2 == 1:
                return max_of_left
            if i == m: min_of_right = nums2[j]
            elif j == n: min_of_right = nums1[i]
            else: min_of_right = min(nums1[i], nums2[j])
            return (max_of_left + min_of_right) / 2.0
nums1 = [1, 3]
nums2 = [2]
print(findMedianSortedArrays(nums1, nums2))  # Output: 2.0
nums1 = [1, 2]
nums2 = [3, 4]
print(findMedianSortedArrays(nums1, nums2))  # Output: 2.5
```

Output

```
2
2.5

=== Code Execution Successful ===
```

2. Given two integers dividend and divisor, divide two integers without using multiplication, division, and mod operator.

The integer division should truncate toward zero, which means losing its fractional part. For example, 8.345 would be truncated to 8, and -2.7335 would be truncated to -2.

Return the quotient after dividing dividend by divisor.

Note: Assume we are dealing with an environment that could only store integers within the 32-bit

signed integer range: [−231, 231 − 1]. For this problem, if the quotient is strictly greater than 231

- 1, then return 231 - 1, and if the quotient is strictly less than -231, then return -231.

Example 1:

Input: dividend = 10, divisor = 3

Output: 3

Explanation: 10/3 = 3.33333.. which is truncated to 3.

Example 2:

Input: dividend = 7, divisor = -3

Output: -2

main.py   Share   Run

```python
def divide(dividend, divisor):
    # Handle overflow case
    if dividend == -2**31 and divisor == -1:
        return 2**31 - 1
    negative = (dividend < 0) != (divisor < 0)
    dividend, divisor = abs(dividend), abs(divisor)
    quotient = 0
    while dividend >= divisor:
        temp, multiple = divisor, 1
        while dividend >= (temp << 1):
            temp <<= 1
            multiple <<= 1
        dividend -= temp
        quotient += multiple
    if negative:
        quotient = -quotient
    return max(min(quotient, 2**31 - 1), -2**31)
dividend1, divisor1 = 10, 3
print(divide(dividend1, divisor1))  # Output: 3

dividend2, divisor2 = 7, -3
print(divide(dividend2, divisor2))  # Output: -2
```

Output

```
3
-2

=== Code Execution Successful ===
```

3. Given the root of a binary tree, imagine yourself standing on the right side of it, return the values of the nodes you can see ordered from top to bottom.

Input: root = [1, 2, 3, null, 5, null, 4]

Output: [1,3,4]

Example 2:

Input: root = [1,null,3]

Output: [1,3]

Example 3:

Input: root = []

Output: []

Constraints:

The number of nodes in the tree is in the range [0, 100].

-100 <= Node.val <= 100

ain.py Share Run Output

```python
from collections import deque
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def rightSideView(root):
    if not root:
        return []

    right_view = []
    queue = deque([root])

    while queue:
        level_length = len(queue)
        for i in range(level_length):
            node = queue.popleft()
            if i == level_length - 1:
                right_view.append(node.val)
            if node.left:
                queue.append(node.left)
            if node.right:
                queue.append(node.right)

    return right_view
root1 = TreeNode(1)
root1.left = TreeNode(2)
```

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

=== Code Execution Successful ===
```

```
main.py                                   Share   Run

    while queue:
        level_length = len(queue)
        for i in range(level_length):
            node = queue.popleft()
            if i == level_length - 1:
                right_view.append(node.val)
            if node.left:
                queue.append(node.left)
            if node.right:
                queue.append(node.right)

    return right_view
root1 = TreeNode(1)
root1.left = TreeNode(2)
root1.right = TreeNode(3)
root1.left.right = TreeNode(5)
root1.right.right = TreeNode(4)

print(rightSideView(root1))  # Output: [1, 3, 4]

root2 = TreeNode(1)
root2.right = TreeNode(3)

print(rightSideView(root2))  # Output: [1, 3]
root3 = None

print(rightSideView(root3))  # Output: []
```

```
Output
[1, 3, 4]
[1, 3]
[]

=== Code Execution Successful ===
```

4. Given an integer array nums, move all 0's to the end of it while maintaining the relative order

of the non-zero elements.

Note that you must do this in-place without making a copy of the array.

Example 1:

Input: nums = [0,1,0,3,12]

Output: [1,3,12,0,0]

Example 2:

Input: nums = [0]

Output: [0]

Constraints:

1 <= nums.length <= 104

-31 <= nums[i] <= 231 – 1

main.py

```python
def moveZeroes(nums):
    last_non_zero_found_at = 0

    # Move all non-zero elements to the front of the array
    for current in range(len(nums)):
        if nums[current] != 0:
            # Swap elements at current and last_non_zero_found_at pointers
            nums[last_non_zero_found_at], nums[current] = nums[current], nums[last_non_zero_found_at]
            last_non_zero_found_at += 1

# Example usage:
nums1 = [0, 1, 0, 3, 12]
moveZeroes(nums1)
print(nums1)  # Output: [1, 3, 12, 0, 0]

nums2 = [0]
moveZeroes(nums2)
print(nums2)  # Output: [0]

nums3 = [4, 2, 4, 0, 0, 3, 0, 5, 1, 0]
moveZeroes(nums3)
print(nums3)  # Output: [4, 2, 4, 3, 5, 1, 0, 0, 0, 0]
```

Output

```
[1, 3, 12, 0, 0]
[0]
[4, 2, 4, 3, 5, 1, 0, 0, 0, 0]

=== Code Execution Successful ===
```

5. Given a positive integer num, return true if num is a perfect square or false otherwise.

A perfect square is an integer that is the square of an integer. In other words, it is the product of

some integer with itself.

You must not use any built-in library function, such as sqrt.

Example 1:

Input: num = 16

Output: true

Explanation: We return true because 4 * 4 = 16 and 4 is an integer.

Example 2:

Input: num e= 14

Output: fals

Explanation: We return false because 3.742 * 3.742 = 14 and 3.742 is not an integer.

Constraints:

1 <= num <= 231 -1

main.py

```python
def isPerfectSquare(num):
    if num < 1:
        return False

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

    return False

# Example usage:
print(isPerfectSquare(16))  # Output: True
print(isPerfectSquare(14))  # Output: False
```

Output

```
True
False

=== Code Execution Successful ===
```