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

Explanation: merged array = [1,2,3,4] and median is (2 + 3) / 2 = 2.5.

Constraints:

nums1.length == m

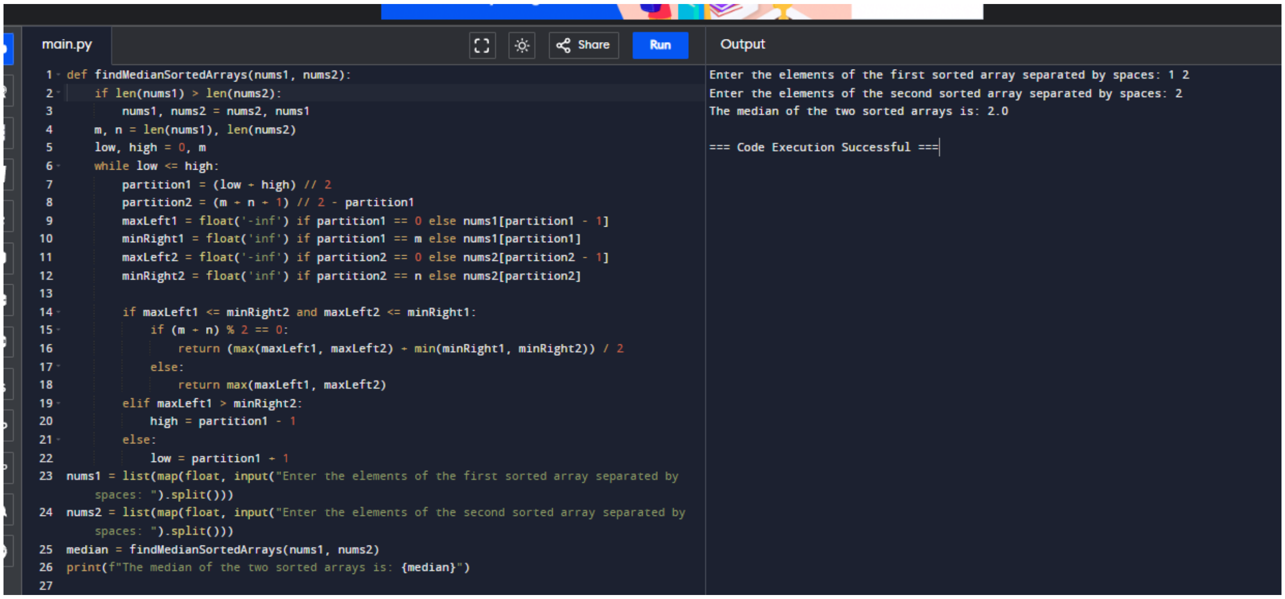
nums2.length == n

0 <= m <= 1000

0 <= n <= 1000

1 <= m + n <= 2000

-106 <= nums1[i], nums2[i] <= 106



```python
def findMedianSortedArrays(nums1, nums2):
    if len(nums1) > len(nums2):
        nums1, nums2 = nums2, nums1
    m, n = len(nums1), len(nums2)
    low, high = 0, m
    while low <= high:
        partition1 = (low + high) // 2
        partition2 = (m + n + 1) // 2 - partition1
        maxLeft1 = float('-inf') if partition1 == 0 else nums1[partition1 - 1]
        minRight1 = float('inf') if partition1 == m else nums1[partition1]
        maxLeft2 = float('-inf') if partition2 == 0 else nums2[partition2 - 1]
        minRight2 = float('inf') if partition2 == n else nums2[partition2]

        if maxLeft1 <= minRight2 and maxLeft2 <= minRight1:
            if (m + n) % 2 == 0:
                return (max(maxLeft1, maxLeft2) + min(minRight1, minRight2)) / 2
            else:
                return max(maxLeft1, maxLeft2)
        elif maxLeft1 > minRight2:
            high = partition1 - 1
        else:
            low = partition1 + 1
nums1 = list(map(float, input("Enter the elements of the first sorted array separated by spaces: ").split()))
nums2 = list(map(float, input("Enter the elements of the second sorted array separated by spaces: ").split()))
median = findMedianSortedArrays(nums1, nums2)
print(f"The median of the two sorted arrays is: {median}")
```

```
Enter the elements of the first sorted array separated by spaces: 1 2
Enter the elements of the second sorted array separated by spaces: 2
The median of the two sorted arrays is: 2.0

=== 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

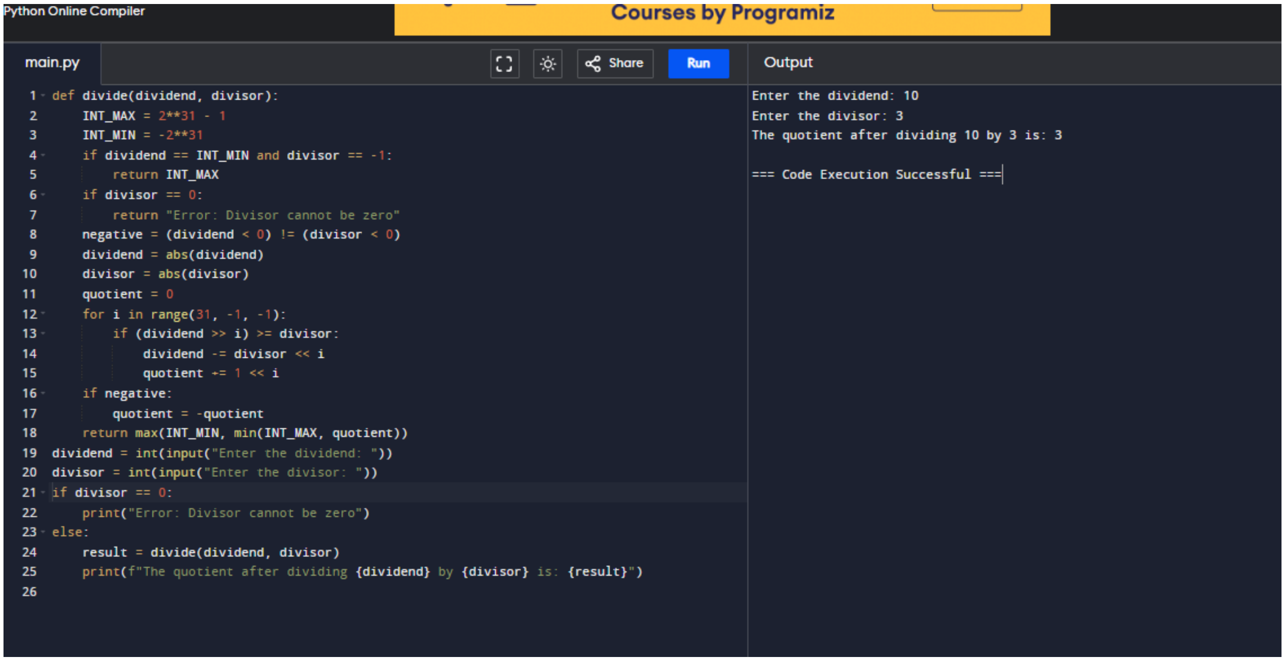
Output: -2

Explanation: 7/-3 = -2.33333.. which is truncated to -2.

Constraints:

-231 <= dividend, divisor <= 231 - 1

divisor != 0



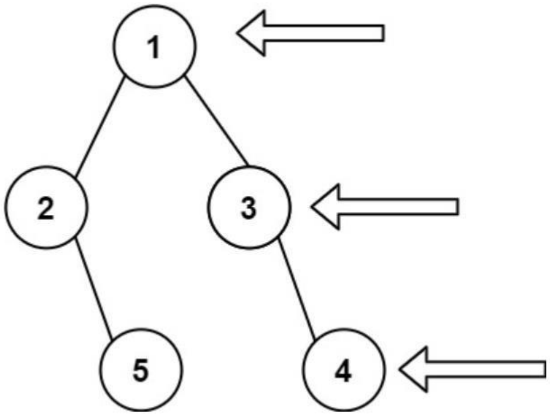
```python
def divide(dividend, divisor):
    INT_MAX = 2**31 - 1
    INT_MIN = -2**31
    if dividend == INT_MIN and divisor == -1:
        return INT_MAX
    if divisor == 0:
        return "Error: Divisor cannot be zero"
    negative = (dividend < 0) != (divisor < 0)
    dividend = abs(dividend)
    divisor = abs(divisor)
    quotient = 0
    for i in range(31, -1, -1):
        if (dividend >> i) >= divisor:
            dividend -= divisor << i
            quotient += 1 << i
    if negative:
        quotient = -quotient
    return max(INT_MIN, min(INT_MAX, quotient))
dividend = int(input("Enter the dividend: "))
divisor = int(input("Enter the divisor: "))
if divisor == 0:
    print("Error: Divisor cannot be zero")
else:
    result = divide(dividend, divisor)
    print(f"The quotient after dividing {dividend} by {divisor} is: {result}")
```

```
Enter the dividend: 10
Enter the divisor: 3
The quotient after dividing 10 by 3 is: 3

=== 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.

Example 1:



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:

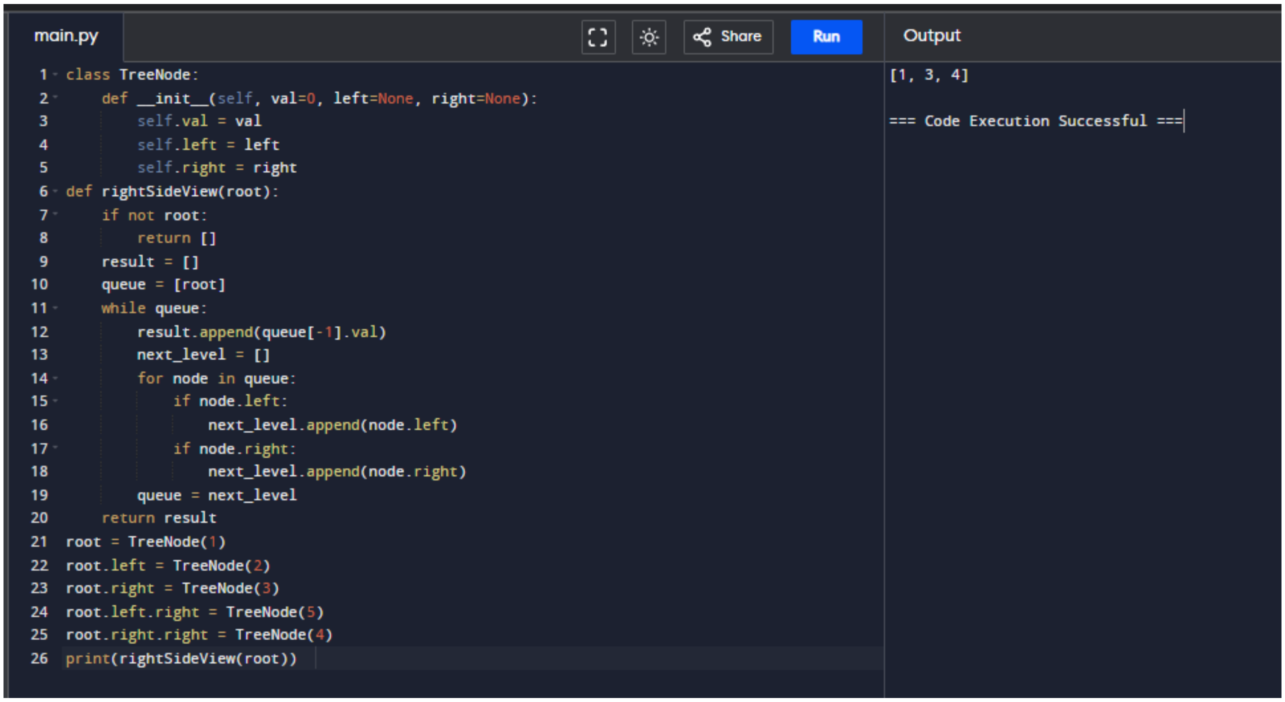
Input: root = []

Output: []

Constraints:

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

-100 <= Node.val <= 100



```python
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 []
    result = []
    queue = [root]
    while queue:
        result.append(queue[-1].val)
        next_level = []
        for node in queue:
            if node.left:
                next_level.append(node.left)
            if node.right:
                next_level.append(node.right)
        queue = next_level
    return result
root = TreeNode(1)
root.left = TreeNode(2)
root.right = TreeNode(3)
root.left.right = TreeNode(5)
root.right.right = TreeNode(4)
print(rightSideView(root))
```

```
[1, 3, 4]

=== 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:

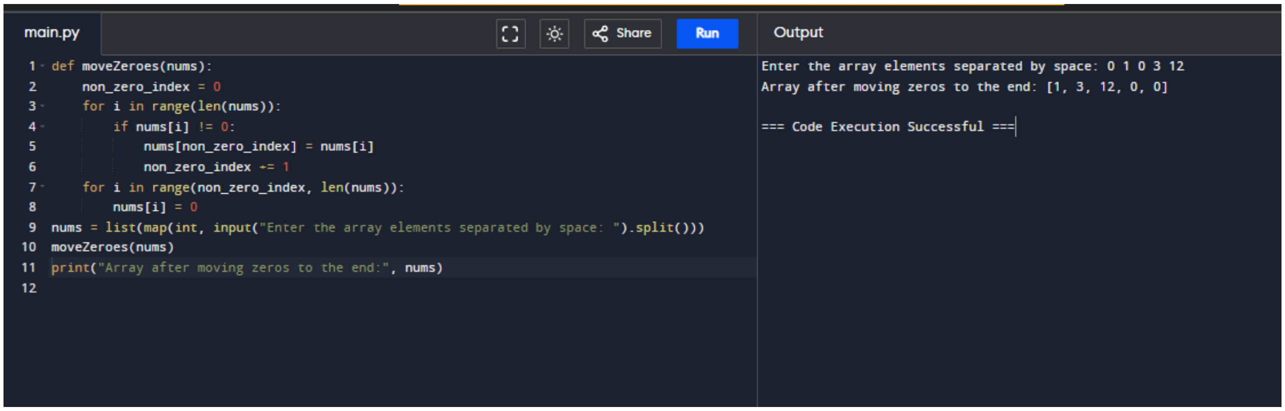
Input: nums = [0]

Output: [0]

Constraints:

1 <= nums.length <= 104

-231 <= nums[i] <= 231 – 1



```python
def moveZeroes(nums):
    non_zero_index = 0
    for i in range(len(nums)):
        if nums[i] != 0:
            nums[non_zero_index] = nums[i]
            non_zero_index += 1
    for i in range(non_zero_index, len(nums)):
        nums[i] = 0
nums = list(map(int, input("Enter the array elements separated by space: ").split()))
moveZeroes(nums)
print("Array after moving zeros to the end:", nums)
```

```
Enter the array elements separated by space: 0 1 0 3 12
Array after moving zeros to the end: [1, 3, 12, 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:

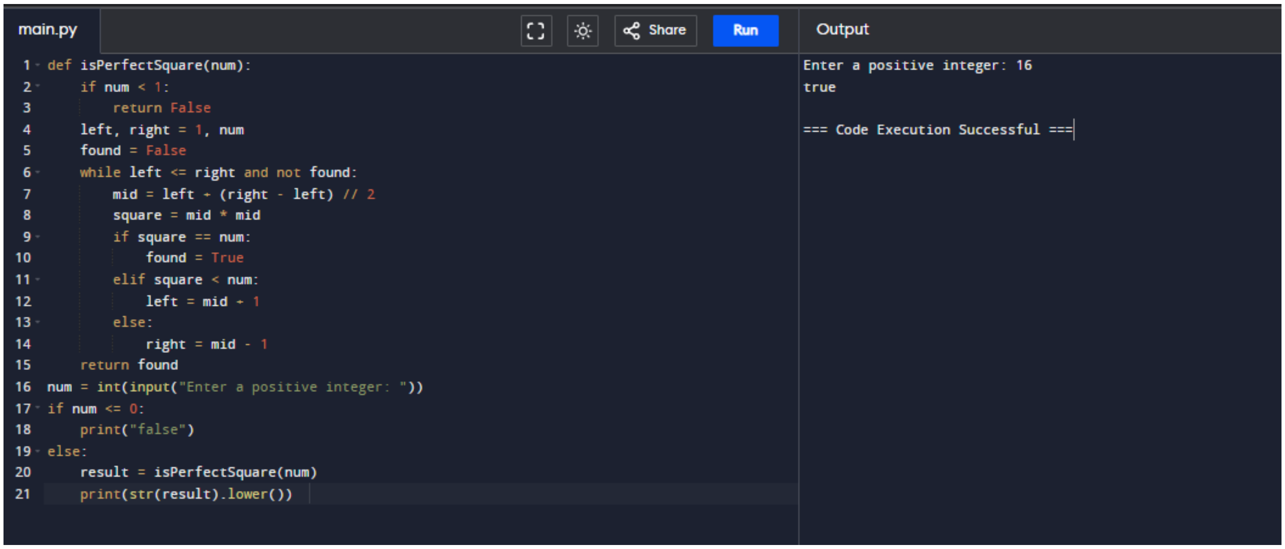
Input: num e= 14

Output: false

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

Constraints:

1 <= num <= 231 – 1



```python
def isPerfectSquare(num):
    if num < 1:
        return False
    left, right = 1, num
    found = False
    while left <= right and not found:
        mid = left + (right - left) // 2
        square = mid * mid
        if square == num:
            found = True
        elif square < num:
            left = mid + 1
        else:
            right = mid - 1
    return found
num = int(input("Enter a positive integer: "))
if num <= 0:
    print("false")
else:
    result = isPerfectSquare(num)
    print(str(result).lower())
```

```
Enter a positive integer: 16
true

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