

ASSIGNMENT-6


NAME: B. Siva Shirish

REG-NO: 192324016

SUBJECT: Python

CODE: CSA0898

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

main.py	Run	Output
<pre>1 def findMedianSortedArrays(nums1, nums2): 2 nums = sorted(nums1 + nums2) 3 n = len(nums) 4 if n % 2 == 0: 5 return (nums[n // 2 - 1] + nums[n // 2]) / 2 6 else: 7 return nums[n // 2] 8 9 nums1 = [1, 3] 10 nums2 = [2] 11 print(findMedianSortedArrays(nums1, nums2)) 12</pre>		2 === 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: $[-2^{31}, 2^{31} - 1]$. For this problem, if the quotient is strictly greater than $2^{31} - 1$, then return $2^{31} - 1$, and if the quotient is strictly less than -2^{31} , then return -2^{31} .

main.py	Run	Output
<pre>1 def divide(dividend: int, divisor: int) -> int: 2 MAX_INT = 2**31 - 1 3 MIN_INT = -2**31 4 if dividend == MIN_INT and divisor == -1: 5 return MAX_INT 6 negative = (dividend < 0) != (divisor < 0) 7 dividend = abs(dividend) 8 divisor = abs(divisor) 9 10 quotient = 0 11 while dividend >= divisor: 12 temp, multiple = divisor, 1 13 while dividend >= (temp << 1): 14 temp <<= 1 15 multiple <<= 1 16 dividend -= temp 17 quotient += multiple 18 19 if negative: 20 quotient = -quotient 21 return max(MIN_INT, min(MAX_INT, quotient)) 22 dividend = 10 23 divisor = 3 24 print(divide(dividend, divisor)) # Output: 3 25</pre>	<div>Run</div>	<pre>3 === Code Execution Successful ===</pre>

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.

```
Programiz Python Online Compiler


main.py [Icons] [Share] [Run] Output

1 from collections import deque
2 class TreeNode:
3     def __init__(self, val=0, left=None, right=None):
4         self.val = val
5         self.left = left
6         self.right = right
7
8 def rightSideView(root: TreeNode):
9     if not root:
10        return []
11
12    right_view, queue = [], deque([root])
13
14    while queue:
15        level_length = len(queue)
16        for i in range(level_length):
17            node = queue.popleft()
18            if i == level_length - 1:
19                right_view.append(node.val)
20            if node.left:
21                queue.append(node.left)
22            if node.right:
23                queue.append(node.right)
24
```




[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]`

main.py	Run	Output
<pre>1 def moveZeroes(nums): 2 n = len(nums) 3 last_non_zero_found_at = 0 4 5 for i in range(n): 6 if nums[i] != 0: 7 nums[last_non_zero_found_at], nums[i] = nums[i], nums[last_non_zero_found_at] 8 last_non_zero_found_at += 1 9 10 11 nums = [0, 1, 0, 3, 12] 12 moveZeroes(nums) 13 print(nums) 14</pre>		<pre>[1, 3, 12, 0, 0] === Code Execution Successful ===</pre>

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

main.py	   Share	Run	Output
<pre>1 def isPerfectSquare(num): 2 if num < 0: 3 return False 4 if num == 0: 5 return True 6 x = num 7 y = (x + 1) // 2 8 while y < x: 9 x = y 10 y = (x + num // x) // 2 11 return x * x == num 12 13 num = 16 14 print(isPerfectSquare(num))</pre>			True === Code Execution Successful ===