

Questions:

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`

$0 \leq m \leq 1000$

$0 \leq n \leq 1000$

$1 \leq m + n \leq 2000$

$-10^6 \leq \text{nums1}[i], \text{nums2}[i] \leq 10^6$

```
Run Save
1 def findMedianSortedArrays(nums1, nums2):
2     if len(nums1) > len(nums2):
3         nums1, nums2 = nums2, nums1
4     m, n = len(nums1), len(nums2)
5     imin, imax, half_len = 0, m, (m + n + 1) // 2
6     while imin <= imax:
7         i = (imin + imax) // 2
8         j = half_len - i
9         if i < m and nums1[i] < nums2[j-1]:
10             imin = i + 1
11         elif i > 0 and nums1[i-1] > nums2[j]:
12             imax = i - 1
13         else:
14             if i == 0:
15                 max_of_left = nums2[j-1]
16             elif j == 0:
17                 max_of_left = nums1[i-1]
18             else:
19                 max_of_left = max(nums1[i-1], nums2[j-1])
20             if (m + n) % 2 == 1:
21                 return max_of_left
22             if i == m:
23                 min_of_right = nums2[j]
24             elif j == n:
25                 min_of_right = nums1[i]
26             else:
27                 min_of_right = min(nums1[i], nums2[j])
28     return (max_of_left + min_of_right) // 2

```

1 3
2 The median is: 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} .

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:

$-2^{31} \leq \text{dividend}, \text{divisor} \leq 2^{31} - 1$

divisor $\neq 0$



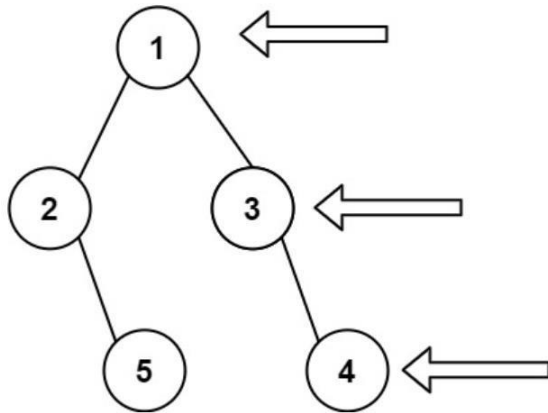
The screenshot shows a Python programming environment with a header bar containing the SIMATS ENGINEERING logo, the text "Python Programming", and a user profile for "ARIPAKA HARSHA MUKUNDHA" with ID "192324070". Below the header is a yellow bar with "Run" and "Save" buttons. The main area is a dark-themed code editor with the following Python code:

```
1 def divide(dividend, divisor):
2     MAX_INT = 2**31 - 1
3     MIN_INT = -2**31
4     if dividend == MIN_INT and divisor == -1:
5         return MAX_INT
6     if dividend == MIN_INT and divisor == 1:
7         return MIN_INT
8     negative = (dividend < 0) != (divisor < 0)
9     dividend, divisor = abs(dividend), abs(divisor)
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    if negative:
19        quotient = -quotient
20    return max(MIN_INT, min(MAX_INT, quotient))
21 dividend = int(input())
22 divisor = int(input())
23 result = divide(dividend, divisor)
24 print("The quotient is:", result)
25
```

At the bottom, there is a console output area. On the left, it shows the input "10" and "3". On the right, it shows the output "The quotient is: 3".

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]

```
Run Save
1 class TreeNode:
2     def __init__(self, val=0, left=None, right=None):
3         self.val = val
4         self.left = left
5         self.right = right
6 def rightSideView(root):
7     if not root:
8         return []
9     right_view = []
10    queue = [root]
11    while queue:
12        level_length = len(queue)
13        for i in range(level_length):
14            node = queue.pop(0)
15            if i == level_length - 1:
16                right_view.append(node.val)
17            if node.left:
18                queue.append(node.left)
19            if node.right:
20                queue.append(node.right)
21    return right_view
22 def insertLevelOrder(arr, root, i, n):
23     if i < n:
24         temp = TreeNode(arr[i] if arr[i] is not None else None)
25         root = temp
26
1, 2, 3, 5, 4
The right side view of the binary tree is: [1, 3, 4]
```

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:

$1 \leq \text{nums.length} \leq 104$

$-231 \leq \text{nums}[i] \leq 231 - 1$

 Python Programming  ARIPAKA HARSHA MUKUNDHA 192324070

Run Save

```
1 def moveZeroes(nums):
2     non_zero_index = 0
3     for i in range(len(nums)):
4         if nums[i] != 0:
5             nums[non_zero_index] = nums[i]
6             non_zero_index += 1
7     for i in range(non_zero_index, len(nums)):
8         nums[i] = 0
9 import ast
10 nums = ast.literal_eval(input())
11 moveZeroes(nums)
12 print("The array after moving zeroes to the end:", nums)
```

[0,1,0,3,12]

The array after moving zeroes to the end: [1, 3, 12, 0, 0]

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 = 14

Output: false

