LeetCode Bootcamp Week 7

```python
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def rightSideView(self, root: Optional[TreeNode]) -> List[int]:

        if not root: return []
        q, ll = deque([root]), []
        while q:
            rv = None
            for _ in range(len(q)):
                rv = q.popleft()
                if rv.left: q.append(rv.left)
                if rv.right: q.append(rv.right)
            ll.append(rv.val)
        return ll
```

Problem List

Run | Submit | Premium

Description | Editorial | Solutions | Accepted | Submissions

All Submissions

Accepted 217 / 217 testcases passed

XoITACHIXo submitted at Apr 29, 2025 12:04

Editorial | Solution

Runtime

0 ms | Beats 100.00%

Analyze Complexity

Memory

17.69 MB | Beats 87.87%

Code | Python3

```python
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def rightSideView(self, root: Optional[TreeNode]) -> List[int]:
```

Code

Python3 | Auto

```python
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def rightSideView(self, root: Optional[TreeNode]) -> List[int]:

        if not root: return []
        q, ll = deque([root]), []
        while q:
            rv = None
            for _ in range(len(q)):
                rv = q.popleft()
```

Saved | Ln 14, Col 36

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 | Case 2 | Case 3 | Case 4

Input

root =

[1,2,3,null,5,null,4]

Output

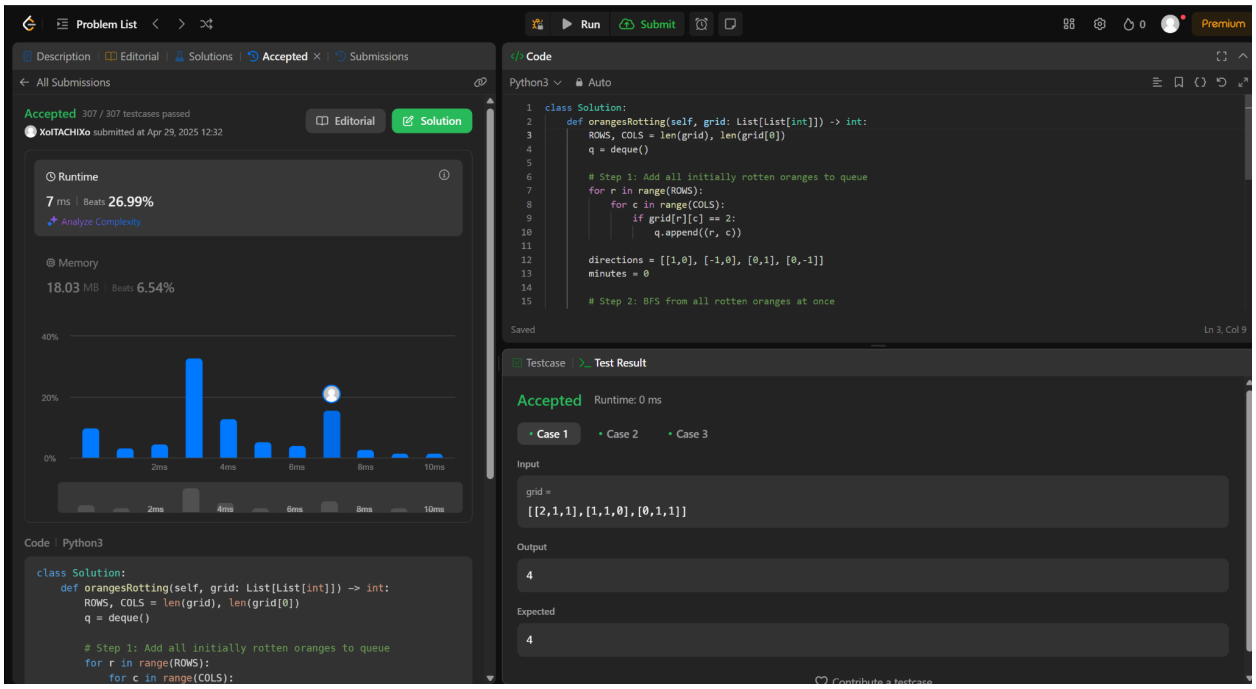
[1,3,4]

Expected

[1,3,4]

Contribute a testcase

```python
class Solution:
    def orangesRotting(self, grid: List[List[int]]) -> int:
        ROWS, COLS = len(grid), len(grid[0])
        q = deque()

        # Step 1: Add all initially rotten oranges to queue
        for r in range(ROWS):
            for c in range(COLS):
                if grid[r][c] == 2:
                    q.append((r, c))

        directions = [[1,0], [-1,0], [0,1], [0,-1]]
        minutes = 0

        # Step 2: BFS from all rotten oranges at once
        while q:
            n = len(q)
            for _ in range(n):
                r, c = q.popleft()
                for dr, dc in directions:
                    new_r, new_c = r + dr, c + dc
                    if 0 <= new_r < ROWS and 0 <= new_c < COLS and
grid[new_r][new_c] == 1:
                        grid[new_r][new_c] = 2
                        q.append((new_r, new_c))
            if q:
                minutes += 1

        # Step 3: Check if any fresh orange remains
        for r in range(ROWS):
            for c in range(COLS):
                if grid[r][c] == 1:
                    return -1

        return minutes
```




```python
class Solution:
    def minSubArrayLen(self, target: int, nums: List[int]) -> int:
        min_len = float("inf")
        left = 0
        cur_sum = 0

        for right in range(len(nums)):
            cur_sum += nums[right]

            while cur_sum >= target:
                if right - left + 1 < min_len:
                    min_len = right - left + 1
                cur_sum -= nums[left]
                left += 1

        return min_len if min_len != float("inf") else 0
```

All Submissions

Accepted 21 / 21 testcases passed

XoITACHIXo submitted at Apr 29, 2025 12:35

Editorial | Solution

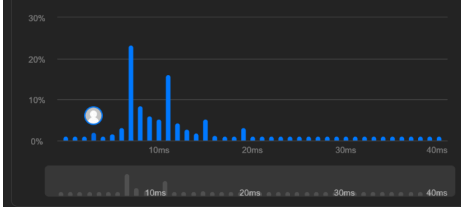
Runtime

3 ms | Beats 99.85%

Analyze Complexity

Memory

28.44 MB | Beats 11.25%



Code | Python3

```python
class Solution:
    def minSubArrayLen(self, target: int, nums: List[int]) -> int:
        min_len = float("inf")
        left = 0
        cur_sum = 0

        for right in range(len(nums)):
            cur_sum += nums[right]

            while cur_sum >= target:
                if right - left + 1 < min_len:
                    min_len = right - left + 1
                cur_sum -= nums[left]
                left += 1

        return min_len if min_len != float("inf") else 0
```

Python3 | Auto

```python
        for right in range(len(nums)):
            cur_sum += nums[right]

            while cur_sum >= target:
                if right - left + 1 < min_len:
                    min_len = right - left + 1
                cur_sum -= nums[left]
                left += 1

        return min_len if min_len != float("inf") else 0
```

Saved

Ln 1, Col 1

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 | Case 2 | Case 3

Input

target =
7

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

Output

2

Expected