

# LeetCode Bootcamp Week 7

```
# 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

Description Editorial Solutions Accepted Submissions

All Submissions

Accepted 217 / 217 testcases passed

XoTACHIXo submitted at Apr 29, 2025 12:04

Editorial Solution

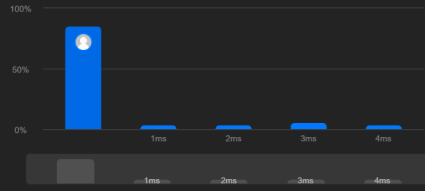
Runtime

0 ms | Beats 100.00%

Analyze Complexity

Memory

17.69 MB | Beats 87.87%



Code | Python3

# 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]:

Run Submit

Python3 Auto

1 # Definition for a binary tree node.  
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 class Solution:  
8 def rightSideView(self, root: Optional[TreeNode]) -> List[int]:  
9  
10 if not root: return []  
11 q, ll = deque([root]), []  
12 while q:  
13 rv = None  
14 for \_ in range(len(q)):  
15 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 feedback

```

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

```

Problem List

<

>

🔍

Description

Editorial

Solutions

Accepted

Submissions

← All Submissions

Accepted 307 / 307 testcases passed

XoTACHIXo submitted at Apr 29, 2025 12:32

Editorial

Solution

Runtime

7 ms | Beats 26.99%

Analyze Complexity

Memory

18.03 MB | Beats 6.54%

Code | Python3

```

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

```

Run

Submit

🔍

📄

Premium

Python3

Auto

```

1 class Solution:
2     def orangesRotting(self, grid: List[List[int]]) -> int:
3         ROWS, COLS = len(grid), len(grid[0])
4         q = deque()
5
6         # Step 1: Add all initially rotten oranges to queue
7         for r in range(ROWS):
8             for c in range(COLS):
9                 if grid[r][c] == 2:
10                     q.append((r, c))
11
12         directions = [[1,0], [-1,0], [0,1], [0,-1]]
13         minutes = 0
14
15         # Step 2: BFS from all rotten oranges at once

```

Saved

Ln 3, Col 9

Testcase

Test Result

Accepted

Runtime: 0 ms

Case 1

Case 2

Case 3

Input

grid =

[[2,1,1], [1,1,0], [0,1,1]]

Output

4

Expected

4

Contribute a testcase

```

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  
XoTACHiXo submitted at Apr 29, 2025 12:35

EditorialSolution

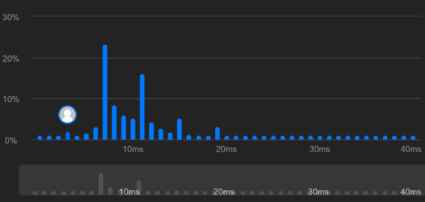
Runtime

3 ms Beats 99.85%

Analyze Complexity

Memory

28.44 MB Beats 11.25%



Code | Python3

```
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)):
```

Python3 Auto

```
6
7     for right in range(len(nums)):
8         cur_sum += nums[right]
9
10        while cur_sum >= target:
11            if right - left + 1 < min_len:
12                min_len = right - left + 1
13            cur_sum -= nums[left]
14            left += 1
15
16        return min_len if min_len != float("inf") else 0
```

SavedLn 1, Col 1

TestcaseTest Result

AcceptedRuntime: 0 ms

Case 1Case 2Case 3

Input

target =  
7

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

Output

2

Expected