

91.7

ryan

Other

PDF generated at: 1 Jul 2024 16:11:41 UTC
[View this report on HackerRank](#)

Score

91.7% • 55 / 60
scored in CodePath TIP101: Unit 4 Assessment, Version A - Summer 2024 in 47 min 32 sec on 1 Jul 2024 08:22:25 PDT

Candidate Information

Email	concepting@protonmail.com
Test	CodePath TIP101: Unit 4 Assessment, Version A - Summer 2024
Candidate Packet	View
Taken on	1 Jul 2024 08:22:25 PDT
Time taken	47 min 32 sec/ 60 min
Work Experience	< 1 years
Invited by	CodePath

Skill Distribution



There is no associated skills data that can be shown for this assessment



Tags Distribution



There is no associated tags data that can be shown for this assessment

Questions

Status	No.	Question	Time Taken	Skill	Score
✓	1	Move Zeros Multiple Choice	3 min 23 sec	-	5/5
✓	2	Find Middle Element Multiple Choice	9 min 23 sec	-	5/5
✓	3	Reverse a String Multiple Choice	6 min 10 sec	-	5/5

	4	Applying the Two-Pointer Technique in Lists Multiple Choice	2 min 52 sec	-	0/5
	5	Find the Target Coding	5 min 33 sec	-	20/20
	6	Find the Minimum Sublist Sum Coding	20 min 4 sec	-	20/20

1. Move Zeros

 Correct

Multiple Choice

Question description

Consider the following function, which is designed to move all zeroes in a list to its end while maintaining the order of non-zero elements using the two-pointer technique. After running the function, what is the final state of the input list `lst`?

```
def move_zeros(nums):  
    left = 0  
    for right in range(len(nums)):  
        if nums[right] != 0:  
            nums[left], nums[right] = nums[right], nums[left]  
            left += 1  
    return nums
```

```
# Input nums  
nums = [0, 1, 0, 3, 12]
```

Candidate's Solution

Options: (Expected answer indicated with a tick)

☒ [1, 3, 12, 0, 0]



☐ [0, 0, 1, 3, 12]

☐ [1, 3, 0, 12, 0]

☐ [12, 3, 1, 0, 0]

 No comments.

2. Find Middle Element

 Correct

Multiple Choice

Question description

Suppose you have the following function intended to find the middle element of a list using the two-pointer technique. However, there's a logical mistake in the implementation. Identify the issue that would prevent the function from correctly identifying the middle element for lists with an even number of elements.

```
def find_middle(lst):  
    slow = fast = 0  
    while fast < len(lst) and fast + 1 < len(lst):  
        slow += 1  
        fast += 2  
    return lst[slow]
```

```
# Example list  
lst = [1, 2, 3, 4, 5, 6]
```

Note: For lists with an even number of elements, return the lower of the two middle elements.

Candidate's Solution

Options: (Expected answer indicated with a tick)



It does not handle lists with an odd number of elements correctly.



It returns the element one position after the middle for lists with an even number of elements.



It incorrectly uses addition for the fast pointer instead of doubling its value.



It does not account for lists with an even number of elements, thus failing to provide a middle element in such cases.



No comments.

3. Reverse a String

 Correct

Multiple Choice

Question description

Consider the following function designed to reverse a given string in-place using the two-pointer technique. However, there's an error in the logic of swapping the characters:

```
def reverse_string(s):  
    left, right = 0, len(s) - 1  
    while left < right:  
        temp = s[left]  
        s[left] = s[right]  
        left += 1  
        right -= 1
```

To achieve the expected outcome of reversing the string `s = ["h","e","l","l","o"]` to `["o","l","l","e","h"]`, what correction is needed in the function?

Candidate's Solution

Options: (Expected answer indicated with a tick)

☐ Replace `temp = s[left]` with `temp = s[right]`

☒ Add `s[right] = temp` after the line `s[left] = s[right]`



☐ Swap the lines `left += 1` and `right -= 1`

☐ Replace `while left < right:`

 No comments.

4. Applying the Two-Pointer Technique in Lists

 Incorrect

Multiple Choice

Question description

Consider a list of integers where you need to identify if there are two distinct numbers that add up to a specific target value. How can the two-pointer technique be applied effectively in this scenario?

Candidate's Solution

Options: (Expected answer indicated with a tick)

☐ By first sorting the list, then initializing one pointer at the start and another at the end of the array, moving them towards each other based on the sum comparison to the target.

☒ By using one pointer to iterate through the list and another to keep track of the current maximum value until the target sum is reached.

☐ By initializing two pointers at the start of the list and moving both simultaneously towards the end until the target sum is found.



☐ By placing both pointers in the middle of the list and moving them outward in opposite directions to find two elements that match the target sum.

 No comments.

5. Find the Target

 Correct

Coding

Question description

You are given a string `s` consisting of numerical characters and a target sum `target`. Write a function to find if there is a pair of adjacent numbers in the string that add up to `target`. The function should return a boolean value: `True` if such a pair exists, and `False` otherwise.

Use the two-pointer technique to solve this problem without converting the entire string into a list of numbers.

Note: Each character in the string `s` should be treated as a separate digit; for example, '56' in the string should be considered as '5' and '6'.

Example 1:

Input: `s = "1234"`, `target = 5`

Output: `True`

Explanation: The digits '2' and '3' are adjacent and add up to 5.

Example 2:

Input: `s = "1112"`, `target = 4`

Output: `False`

Explanation: There are no two adjacent digits that add up to 4.

Candidate's Solution

Language used: Python 3

```
1  #!/bin/python3
2
3  import math
4  import os
5  import random
6  import re
7  import sys
8
9
10
11 #
12 # Complete the 'find_pair_sum' function below.
13 #
14 # The function is expected to return a BOOLEAN.
15 # The function accepts following parameters:
16 # 1. STRING s
17 # 2. INTEGER target
18 #
19
20 def find_pair_sum(s, target):
21     # Write your code here
```



```

22     right = len(s) - 1
23     current = 0
24
25     for i in range(right):
26         current = int(s[i])
27         next = int(s[i + 1])
28         if current + next == target:
29             return True
30     return False
31
32 if __name__ == '__main__':
33     fptr = open(os.environ['OUTPUT_PATH'], 'w')
34
35     s = input()
36
37     target = int(input().strip())
38
39     result = find_pair_sum(s, target)
40
41     fptr.write(str(result) + '\n')
42
43
44     fptr.close()
45

```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 0	Easy	Hidden	Success	2	0.0336 sec	10.4 KB
Testcase 1	Easy	Hidden	Success	2	0.0374 sec	10.3 KB
Testcase 2	Easy	Hidden	Success	2	0.0324 sec	10.3 KB
Testcase 3	Easy	Hidden	Success	2	0.0269 sec	10.2 KB

Testcase 4	Easy	Hidden	Success	2	0.0295 sec	10.2 KB
Testcase 5	Easy	Hidden	Success	2	0.0359 sec	10.2 KB
Testcase 6	Easy	Hidden	Success	2	0.0306 sec	10.3 KB
Testcase 7	Easy	Hidden	Success	2	0.0294 sec	10.3 KB
Testcase 8	Easy	Hidden	Success	2	0.0331 sec	10.3 KB
Testcase 9	Easy	Hidden	Success	2	0.0342 sec	10.3 KB

🚫 No comments.

6. Find the Minimum Sublist Sum

✅ Correct

Coding

Question description

Given a list of integers `nums` and an integer `k`, write a function to find the minimum sum of any contiguous sublist of size `k`. If the size of `nums` is less than `k`, return `0`.

Example 1:

Input: `nums = [5, -1, 3, 2, -4]`, `k = 2`

Output: `-3`

Explanation: Explanation: The sublist of length 2 are `[5, -1]`, `[-1, 3]`, `[3, 2]`, and `[2, -4]`.

Their sums are 4, 2, 5, and -2 respectively. The smallest sum among these is -2, which comes from the sublist [2, -4].

Example 2:

Input: nums[4,2,-5,1,3], k =1

Output: -5

Explanation: Here the sublist is just one element, smallest element is -5.

Candidate's Solution

Language used: Python 3

```
1  #!/bin/python3
2
3  import math
4  import os
5  import random
6  import re
7  import sys
8
9
10
11 #
12 # Complete the 'find_min_sublist_sum' function below.
13 #
14 # The function is expected to return an INTEGER.
15 # The function accepts following parameters:
16 # 1. INTEGER_ARRAY nums
17 # 2. INTEGER k
18 #
19
20 def find_min_sublist_sum(nums, k):
21     # Write your code here
22     if len(nums) < k:
23         return 0
24
25     new_sum = sum(nums[:k])
26     min_sum = new_sum
27
28     for i in range(k, len(nums)):
29         new_sum += nums[i] - nums[i - k]
30         if new_sum < min_sum:
```


```

31         min_sum = new_sum
32     return min_sum
33
34
35 if __name__ == '__main__':
36     fptr = open(os.environ['OUTPUT_PATH'], 'w')
37
38     nums_count = int(input().strip())
39
40     nums = []
41
42     for _ in range(nums_count):
43         nums_item = int(input().strip())
44         nums.append(nums_item)
45
46     k = int(input().strip())
47
48     result = find_min_sublist_sum(nums, k)
49
50     fptr.write(str(result) + '\n')
51
52     fptr.close()
53

```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 0	Easy	Hidden	Success	4	0.0422 sec	10.3 KB
Testcase 1	Easy	Hidden	Success	4	0.0378 sec	10.3 KB
Testcase 2	Easy	Hidden	Success	4	0.0338 sec	10.3 KB
Testcase 3	Easy	Hidden	Success	4	0.0451 sec	10.3 KB

Testcase 4	Easy	Hidden	Success	4	0.0293 sec	10.3 KB
---------------	------	--------	---------	---	------------	---------

 No comments.