



Venkata Naga Sri Sai Pranavi Kolipaka
Other

PDF generated at: 9 Jul 2025 21:40:32 UTC
[View this report on HackerRank](#)

Score

100% • 80 / 80
scored in TIP102: Unit 6 Version A (Standard) - Summer 2025 in 65 min 19 sec on 9 Jul 2025 13:33:17 PDT

Candidate Information

Email	kolipakavnssaipranavi@gmail.com
Test	TIP102: Unit 6 Version A (Standard) - Summer 2025
Candidate Packet	View
Taken on	9 Jul 2025 13:33:17 PDT
Time taken	65 min 19 sec/ 90 min
Personal Member ID	129054
Email Address with CodePath	kolipakavnssaipranavi@gmail.com
Github username with CodePath	Pranavi2002
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

Coding Questions • 60 / 60

Status	No.	Question	Time Taken	Skill	Score	Code Quality
	1	Create a Linked List Coding	5 min 24 sec	-	20/20	-
	2	Insert Node Into Sorted List Coding	22 min 33 sec	-	20/20	-

	3	Longer List Coding	9 min	-	20/20	-
--	---	--------------------	-------	---	-------	---

Multiple Choice + Debugging • 20 / 20

Status	No.	Question	Time Taken	Skill	Score	Code Quality
	4	Which of the following options best represents the linked list with head new_head after running the following code snippet? Multiple Choice	7 min 31 sec	-	5/5	-
	5	What is the time complexity of mystery_function()? Multiple Choice	3 min 27 sec	-	5/5	-
	6	Which of the following options best represents the linked list with head new_head after running the following code snippet? Multiple Choice	11 min 18 sec	-	5/5	-
	7	Debug this code Coding	5 min 41 sec	-	5/5	-

1. Create a Linked List

Correct

Coding

Question description

Given a list of integers, write a function `create_linked_list(values)` that creates a singly linked list where each node contains one of the integers from the list in the same order. The function should return the head of the linked list.

Example:

- **Input:** `values = [1, 2, 3, 4]`
- **Output:** The linked list should be represented as `1 -> 2 -> 3 -> 4`.

Constraints:

- The list of integers will have at least one element and will not exceed 1000 elements.

Notes:

- A helper function `print_linked_list(head)` is provided to test your implementation. It prints the values in the linked list, separated by arrows (`->`).

Candidate's Solution

Language used: Python 3

```
1  #!/bin/python
2
3  import math
4  import os
5  import random
6  import re
7  import sys
8  import ast
9
10 class ListNode:
11     def __init__(self, val=0, next=None):
12         self.val = val
13         self.next = next
14
15 class SinglyLinkedList:
16     def __init__(self):
17         self.head = None
18         self.tail = None
19
20     def insert_node(self, val):
21         node = ListNode(val)
22
23         if not self.head:
24             self.head = node
25         else:
```

```
26         self.tail.next = node
27
28         self.tail = node
29
30 def print_linked_list(head):
31     current = head
32     while current:
33         if current.next:
34             sys.stdout.write(str(current.val) + " -> ")
35         else:
36             sys.stdout.write(str(current.val) + "\n")
37         current = current.next
38
39
40
41 #
42 # Complete the 'create_linked_list' function below.
43 #
44 # The function is expected to return an INTEGER_SINGLY_LINKED_LIST.
45 # The function accepts INTEGER_ARRAY values as parameter.
46 #
47
48 def create_linked_list(values):
49     # Write your code here
50     if not values:
51         return None
52     head = ListNode(values[0])
53     current = head
54     for val in values[1:]:
55         current.next = ListNode(val)
56         current = current.next
57     return head
58
59 if __name__ == '__main__':
60     outfile = open(os.environ['OUTPUT_PATH'], 'w')
61
62     def ll_to_str(head):
63         list_str = ""
64         curr = head
65         while curr:
66             list_str += str(curr.val)
67             if curr.next:
68                 list_str += " -> "
69             curr = curr.next
70         if len(list_str) == 0:
71             return "None"
```

```

72         return list_str
73
74     input_data = input()
75     while(input_data != "END"):
76         param_list = ast.literal_eval(input_data)
77         result_raw = create_linked_list(param_list)
78         result = ll_to_str(result_raw)
79         outfile.write(str(result) + '\n')
80         input_data = input()
81     outfile.close()

```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Multiple Elements	Easy	Hidden	Success	0	0.0425 sec	10.8 KB
Empty List	Easy	Hidden	Success	0	0.0318 sec	11 KB
Single Element	Easy	Hidden	Success	0	0.0335 sec	10.9 KB
Two Elements	Easy	Hidden	Success	0	0.0289 sec	10.9 KB
List with Negative Numbers	Easy	Hidden	Success	0	0.0307 sec	10.9 KB
List with Duplicates	Easy	Hidden	Success	0	0.0339 sec	11 KB
List with Mixed Numbers	Hard	Hidden	Success	0	0.0278 sec	11 KB

List with Zero	Hard	Hidden	Success	0	0.0288 sec	10.9 KB
Pass/Fail Case	Easy	Hidden	Success	20	0.0278 sec	11 KB

! No comments.

2. Insert Node Into Sorted List

✓ Correct

Coding

Question description

You are given the **head of a sorted singly linked list** and a **value** . Create a new node with value **value** and insert it into the provided linked list while maintaining the sorted order of the nodes. Your task is to implement the `insert_sorted()` function that inserts the new node into the correct position **without disrupting the sorted order**.

Example 1:

Input: head = 1 -> 3 -> 5, value = 4

Output: 1 -> 3 -> 4 -> 5

Example 2:

Input: head = 2 -> 6 -> 8, value = 1

Output: 1 -> 2 -> 6 -> 8

Candidate's Solution

Language used: Python 3

```
1 import math
2 import os
3 import random
4 import re
```

```
5 import sys
6 import ast
7
8 class ListNode:
9     def __init__(self, val=0, next=None):
10         self.val = val
11         self.next = next
12
13 # Function to insert a node into a sorted linked list
14 def insert_sorted(head, value):
15     value_node = ListNode(value)
16
17     if head is None or value < head.val:
18         value_node.next = head
19         return value_node
20
21     current = head
22     while current.next and current.next.val < value:
23         current = current.next
24
25     value_node.next = current.next
26     current.next = value_node
27     return head
28 import sys
29
30 # Helper function to create a linked list from a list of values
31 def create_linked_list(values):
32     if not values:
33         return None
34     head = ListNode(values[0])
35     current = head
36     for value in values[1:]:
37         current.next = ListNode(value)
38         current = current.next
39     return head
40
41 # Helper function to convert linked list to a list
42 def linked_list_to_list(head):
43     result = []
44     while head:
45         result.append(head.val)
46         head = head.next
47     return result
48
49 if __name__ == "__main__":
50     input_data = sys.stdin.read().strip().split("\n")
```



```

51     results = []
52
53     for line in input_data:
54         values, value = eval(line) # Parse input as list of values and a new
value
55         head = create_linked_list(values)
56         updated_head = insert_sorted(head, value)
57         results.append(linked_list_to_list(updated_head))
58
59     for res in results:
60         print(res)
61

```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 0	Easy	Hidden	Success	0	0.0295 sec	10.9 KB
Testcase 1	Easy	Hidden	Success	0	0.0301 sec	10.9 KB
Testcase 2	Easy	Hidden	Success	0	0.0264 sec	10.9 KB
Testcase 3	Easy	Hidden	Success	0	0.0278 sec	10.9 KB
Testcase 4	Easy	Hidden	Success	0	0.0304 sec	10.9 KB
Testcase 5	Easy	Hidden	Success	0	0.0277 sec	10.9 KB
Testcase 6	Easy	Hidden	Success	0	0.0317 sec	10.9 KB

Testcase 7	Easy	Hidden	Success	0	0.0664 sec	10.9 KB
Testcase 8	Easy	Hidden	Success	0	0.0265 sec	10.9 KB
Testcase 9	Easy	Hidden	Success	0	0.029 sec	10.9 KB
Pass/Fail Testcases	Easy	Hidden	Success	20	0.0312 sec	10.9 KB

🚫 No comments.

3. Longer List

✓ Correct

Coding

Question description

Implement a function `longer_list()` that accepts the heads of two singly linked lists, `head_a` and `head_b`. Return the head of the linked list with the greatest length.

Candidate's Solution

Language used: Python 3

```
1  #!/bin/python
2
3  import math
4  import os
5  import random
6  import re
7  import sys
8  import ast
9
10 class Node:
11     def __init__(self, val=None):
```

```
12         self.val = val
13         self.next = None
14
15     class ListNode:
16         def __init__(self, val=0, next=None):
17             self.val = val
18             self.next = next
19
20     class SinglyLinkedList:
21         def __init__(self):
22             self.head = None
23             self.tail = None
24
25         def insert_node(self, val):
26             node = ListNode(val)
27
28             if not self.head:
29                 self.head = node
30             else:
31                 self.tail.next = node
32
33                 self.tail = node
34
35     # Helper function to print linked list (for testing)
36     def print_linked_list(head):
37         current = head
38         while current:
39             if current.next:
40                 sys.stdout.write(str(current.val) + " -> ")
41             else:
42                 sys.stdout.write(str(current.val) + "\n")
43             current = current.next
44
45     # Helper function to create a linked list from a list of values
46     def create_linked_list(vals):
47         temp = ListNode()
48         current = temp
49         for val in vals:
50             current.next = ListNode(val)
51             current = current.next
52         return temp.next
53
54     # Complete the 'longer_list' function below.
55     #
56     # The function is expected to return an INTEGER_SINGLY_LINKED_LIST.
57     # The function accepts following parameters:
```

```
58 # 1. INTEGER_ARRAY head1
59 # 2. INTEGER_ARRAY head2
60 #
61
62 def longer_list(head1, head2):
63     # Write your code here
64     length1 = get_length(head1)
65     length2 = get_length(head2)
66
67     if length1 >= length2:
68         return head1
69     else:
70         return head2
71
72 def get_length(head):
73     length = 0
74     while head:
75         length += 1
76         head = head.next
77     return length
78
79 if __name__ == '__main__':
80     #input_data = sys.stdin.read().strip()
81     #input_list = ast.literal_eval(input_data)
82
83     #head1 = create_linked_list(input_list[0])
84     #head2 = create_linked_list(input_list[1])
85
86     #result = longer_list(head1, head2)
87     #print_linked_list(result)
88
89     outfile = open(os.environ['OUTPUT_PATH'], 'w')
90
91     # Helper function to convert str -> linked list
92     def str_to_ll(vals_str):
93         if vals_str is None or vals_str == "None":
94             return None
95         vals = vals_str.split("->")
96         temp_head = Node("temp")
97         temp_curr = temp_head
98         for val in vals:
99             temp_curr.next = Node(val.strip())
100             temp_curr = temp_curr.next
101         return temp_head.next #Don't keep the temp head
102
103     # Helper function to convert linked list -> str
```

```

104 def ll_to_str(head):
105     list_str = ""
106     curr = head
107     while curr:
108         list_str += str(curr.val)
109         if curr.next:
110             list_str += "->"
111         curr = curr.next
112     if len(list_str) == 0:
113         return "None"
114     return list_str
115
116 test_str = input()
117 while(test_str != "END"):
118     # Convert input string to list of param strings
119     param_list = ast.literal_eval(test_str)
120
121     # TODO: Edit parameters as needed
122     head1 = str_to_ll(param_list[0])
123     head2 = str_to_ll(param_list[1])
124
125     # TODO: Edit function name and prepare result string
126     result_raw = longer_list(head1, head2)
127     result = ll_to_str(result_raw)
128
129     # Write output and check for another test case
130     outfile.write(str(result) + '\n')
131     test_str = input()
132
133     outfile.close()

```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 0	Easy	Hidden	Success	0	0.0288 sec	10.9 KB
Both Lists Are Empty	Easy	Hidden	Success	0	0.03 sec	11 KB

One List Is Empty, the Other Is Not	Easy	Hidden	Success	0	0.0452 sec	11.1 KB
One List Is Empty, the Other Is Not	Easy	Hidden	Success	0	0.0302 sec	11.1 KB
Both Lists Have the Same Length 1 -> 2 -> 3	Easy	Hidden	Success	0	0.0303 sec	11.1 KB
Both Lists Have the Same Length 4 -> 5 -> 6	Easy	Hidden	Success	0	0.0284 sec	11 KB
Lists with One Element Each 7	Easy	Hidden	Success	0	0.0359 sec	11 KB
Lists with One Element Each 8	Easy	Hidden	Success	0	0.03 sec	11 KB
One List Is Longer Than the Other	Easy	Hidden	Success	0	0.0314 sec	11.1 KB
One List Is Longer Than the Other	Easy	Hidden	Success	0	0.0285 sec	11.1 KB
Lists with Negative and Positive Numbers	Easy	Hidden	Success	0	0.0351 sec	11.1 KB
Lists with Repeated Elements	Easy	Hidden	Success	0	0.0313 sec	11.1 KB

Single Element in Each List but Different Values	Easy	Hidden	Success	0	0.0286 sec	10.9 KB
Pass/Fail Case	Easy	Hidden	Success	20	0.0282 sec	11 KB

🚫 No comments.

4. Which of the following options best represents the linked list with head `new_head` after running the following code snippet?

✅ Correct

Multiple Choice

Question description

Which of the following options best represents the linked list with head `new_head` after running the following code snippet?

```
class Node:
    def __init__(self, value, next_node = None):
        self.value = value
        self.next = next_node

def mystery_function(head):
    if head is None:
        return None

    if head.next is None:
        return None

    current = head
    while current.next.next:
        current = current.next
    current.next = None
    return head
```

```
# Input List: 1 -> 2 -> 3
head = Node(1, Node(2, Node(3)))

new_head = mystery_function(head)
```

Candidate's Solution

Options: (Expected answer indicated with a tick)



<p>1 -> 2 -> 3<!--
- notionvc: 0042e942-202a-425f-82bc-4b645615a0a2 --></p>



<p>1 -> 2<!--
notionvc: 5a98fe1f-1523-4b2c-9eb1-4885fcf50a20 --></p>



<p>3 -> 2 -> 1<!--
- notionvc: 91d5499f-e65e-4b33-9dc7-afbd00e2fb4f --></p>



<p>None</p>

 No comments.

5. What is the time complexity of mystery_function()?

 Correct

Multiple Choice

Question description

What is the time complexity of `mystery_function()`?

```
# Definition for singly-linked list.
class SinglyLinkedListNode:
    def __init__(self, node_data):
        self.data = node_data
        self.next = None

def mystery_function(head):
    if not head or not head.next:
        return None

    current = head
    while current.next and current.next.next:
        current = current.next
    current.next = None

    return head

# Example Usage:
# Input List: a -> b -> c -> d
head = ListNode('a', ListNode('b', ListNode('c', ListNode('d'))))

new_head = mystery_function(head)
```

Candidate's Solution

Options: (Expected answer indicated with a tick)

☐ $O(1)$

☐ $O(\log n)$

☒ $O(n)$ ☐ $O(n^2)$

 No comments.

6. Which of the following options best represents the linked list with head `new_head` after running the following code snippet?

 Correct

Multiple Choice

Question description

Which of the following options best represents the linked list with head `new_head` after running the following code snippet?

```
class SinglyLinkedListNode:
    def __init__(self, node_data):
        self.data = node_data
        self.next = None

def mystery_function(head):
    if not head or not head.next:
        return head

    prev = None
    tail = head
    while tail.next:
        prev = tail
        tail = tail.next

    if not prev:
        return head

    tail.next = head.next
```

```
prev.next = head  
head.next = None
```

```
return tail
```

```
# Input List: 1 -> 2 -> 3 -> 4 -> 5  
head = ListNode(1, ListNode(2, ListNode(3, ListNode(4, ListNode(5)))))  
new_head = mystery_function(head)
```

Candidate's Solution

Options: (Expected answer indicated with a tick)

☐ `<p>1 -> 2 -> 3 -> 4 -
> 5<!-- notionvc: ed66848e-4e4f-4baa-b77a-7e95d1c80a4e --></p>`

☐ `<p>1 -> 2 -> 3 ->
4<!-- notionvc: 28341557-c82c-4431-a6e0-943274353b1a --></p>`

☐ `<p>5 -> 4 -> 3 -> 2 -
> 1<!-- notionvc: e342538b-b38e-4c90-93a9-727cb49fd119 --></p>`

☒ `<p>5 -> 2 -> 3 -> 4 -
> 1<!-- notionvc: 108b5cd0-c143-4b28-bff7-a4ac9fce6777 --></p>`



 No comments.

7. Debug this code

 Correct

Coding

Question description

The following function is supposed to remove all duplicate values from a **sorted** singly linked list so that each element appears only once. However, the implementation contains one or more errors that prevent it from working correctly.

Your task is to identify the bug(s) in the given implementation and correct them so that it successfully removes all duplicate elements from the linked list.

Example 1:

Input: 1 -> 1 -> 2 -> 3 -> 3 -> 4 -> 4 -> 5

Output: 1 -> 2 -> 3 -> 4 -> 5

Example 2:

Input: 7 -> 7 -> 8 -> 8 -> 9 -> 10 -> 10 -> 10

Output: 7 -> 8 -> 9 -> 10

Candidate's Solution

Language used: Python 3

```
1  #!/bin/python
2
3  import math
4  import os
5  import random
6  import re
7  import sys
8
9  class ListNode:
10     def __init__(self, val=0, next=None):
11         self.val = val
12         self.next = next
13
14
15
16  def remove_duplicates(head: ListNode) -> ListNode:
```

```
17     if not head:
18         return None
19
20     current = head
21     while current and current.next:
22         if current.val == current.next.val:
23             current.next = current.next.next
24         else:
25             current = current.next
26
27     return head
28
29 import sys
30
31 def parse_input():
32     """
33     Reads multiple lines of input, where each line represents a separate
34     linked list.
35     Returns a list of ListNode heads, one for each linked list.
36     """
37     lines = sys.stdin.read().strip().split("\n") # Read all lines separately
38     linked_lists = []
39
40     for line in lines:
41         if not line.strip(): # Handle empty lines (edge case)
42             linked_lists.append(None)
43             continue
44
45         values = line.strip().split(" -> ") # Split the linked list values
46         nodes = [ListNode(int(val)) for val in values]
47
48         for i in range(len(nodes) - 1):
49             nodes[i].next = nodes[i + 1] # Link nodes together
50
51         linked_lists.append(nodes[0]) # Store the head of the linked list
52
53     return linked_lists # Return a list of linked lists
54
55 def print_linked_list(head):
56     """
57     Prints the linked list in the required format.
58     """
59     values = []
60     while head:
61         values.append(str(head.val))
62         head = head.next
```

```
62     print(" -> ".join(values))
63
64 if __name__ == "__main__":
65     heads = parse_input() # Get all linked list heads
66
67     for head in heads: # Process each linked list separately
68         new_head = remove_duplicates(head)
69         print_linked_list(new_head) # Print each modified list on a new line
70
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 0	Easy	Hidden	Success	0	0.026 sec	10.3 KB
Testcase 1	Easy	Hidden	Success	0	0.0269 sec	10.3 KB
Testcase 2	Easy	Hidden	Success	0	0.0269 sec	10.1 KB
Testcase 3	Easy	Hidden	Success	0	0.0258 sec	10.1 KB
Testcase 4	Easy	Hidden	Success	5	0.0263 sec	10.3 KB

⚠️ No comments.