100

Rajasekhar Reddy Kolagotla

Other

PDF generated at: 14 Jul 2025 05:55:26 UTC

View this report on HackerRank

Score

100% • 80 / 80

scored in TIP102: Unit 6 Version A (Standard) - Summer 2025 in 29 min 58 sec on 13 Jul 2025 22:21:06 PDT

Candidate Information

Email	rajasekhar1131997@gmail.com
Test	TIP102: Unit 6 Version A (Standard) - Summer 2025
Candidate Packet	View
Taken on	13 Jul 2025 22:21:06 PDT
Time taken	29 min 58 sec/ 90 min
Personal Member ID	126663
Email Address with CodePath	rajasekhar1131997@gmail.com
Github username with CodePath	Rajasekhar1131997
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	4 min 51 sec	-	20/20	-
✓	2	Insert Node Into Sorted List Coding	9 min 22 sec	-	20/20	-

✓	3	Longer List Coding	5 min 26 sec	-	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	2 min 26 sec	-	5/5	-
✓	5	What is the time complexity of mystery_function()? Multiple Choice	1 min 14 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	1 min 34 sec	-	5/5	-
✓	7	Debug this code Coding	4 min 52 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**

```python
#!/bin/python

import math
import os
import random
import re
import sys
import ast

class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

class SinglyLinkedList:
    def __init__(self):
        self.head = None
        self.tail = None

    def insert_node(self, val):
        node = ListNode(val)

        if not self.head:
            self.head = node
```

```
        else:
            self.tail.next = node

        self.tail = node

def print_linked_list(head):
    current = head
    while current:
        if current.next:
            sys.stdout.write(str(current.val) + " -> ")
        else:
            sys.stdout.write(str(current.val) + "\n")
        current = current.next


#
# Complete the 'create_linked_list' function below.
#
# The function is expected to return an INTEGER_SINGLY_LINKED_LIST.
# The function accepts INTEGER_ARRAY values as parameter.
#

def create_linked_list(values):
    # Write your code here
    if not values:
        return None
    head = ListNode(values[0])
    current = head
    for value in values[1:]:
        current.next = ListNode(value)
        current = current.next
    return head

if __name__ == '__main__':
    outfile = open(os.environ['OUTPUT_PATH'], 'w')

    def ll_to_str(head):
        list_str = ""
        curr = head
        while curr:
            list_str += str(curr.val)
            if curr.next:
                list_str += " -> "
            curr = curr.next
        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	0.0327 sec	11 KB
Empty List	Easy	Hidden	Success	0	0.0294 sec	10.9 KB
Single Element	Easy	Hidden	Success	0	0.0323 sec	11 KB
Two Elements	Easy	Hidden	Success	0	0.0288 sec	10.9 KB
List with Negative Numbers	Easy	Hidden	Success	0	0.0311 sec	10.8 KB
List with Duplicates	Easy	Hidden	Success	0	0.0281 sec	10.9 KB
List with Mixed Numbers	Hard	Hidden	Success	0	0.0281 sec	11 KB

List with Zero	Hard	Hidden	Success	0	0.0279 sec	10.9 KB
Pass/Fail Case	Easy	Hidden	Success	20	0.0299 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
```

```
import sys
import ast

class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next


# Function to insert a node into a sorted linked list
def insert_sorted(head, value):

    new_node = ListNode(value)

    if not head or value < head.val:
        new_node.next = head
        return new_node
    current = head
    while current.next and current.next.val < value:
        current = current.next

    new_node.next = current.next
    current.next = new_node

    return head


import sys

# Helper function to create a linked list from a list of values
def create_linked_list(values):
    if not values:
        return None
    head = ListNode(values[0])
    current = head
    for value in values[1:]:
        current.next = ListNode(value)
        current = current.next
    return head

# Helper function to convert linked list to a list
def linked_list_to_list(head):
    result = []
    while head:
        result.append(head.val)
        head = head.next
```

```
51      return result
52
53  if __name__ == "__main__":
54      input_data = sys.stdin.read().strip().split("\n")
55      results = []
56
57      for line in input_data:
58          values, value = eval(line)  # Parse input as list of values and a new
    value
59          head = create_linked_list(values)
60          updated_head = insert_sorted(head, value)
61          results.append(linked_list_to_list(updated_head))
62
63      for res in results:
64          print(res)
65
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 0	Easy	Hidden	Success	0	0.0323 sec	10.9 KB
Testcase 1	Easy	Hidden	Success	0	0.0298 sec	10.9 KB
Testcase 2	Easy	Hidden	Success	0	0.0277 sec	10.9 KB
Testcase 3	Easy	Hidden	Success	0	0.0471 sec	10.8 KB
Testcase 4	Easy	Hidden	Success	0	0.0293 sec	10.9 KB
Testcase 5	Easy	Hidden	Success	0	0.0293 sec	10.8 KB

Testcase 6	Easy	Hidden	Success	0	0.0364 sec	10.9 KB
Testcase 7	Easy	Hidden	Success	0	0.0332 sec	10.8 KB
Testcase 8	Easy	Hidden	Success	0	0.0336 sec	10.9 KB
Testcase 9	Easy	Hidden	Success	0	0.0283 sec	10.9 KB
Pass/Fail Testcases	Easy	Hidden	Success	20	0.0375 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**

```
#!/bin/python

import math
import os
import random
import re
```

```
import sys
import ast

class Node:
    def __init__(self, val=None):
        self.val = val
        self.next = None

class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

class SinglyLinkedList:
    def __init__(self):
        self.head = None
        self.tail = None

    def insert_node(self, val):
        node = ListNode(val)

        if not self.head:
            self.head = node
        else:
            self.tail.next = node

        self.tail = node

# Helper function to print linked list (for testing)
def print_linked_list(head):
    current = head
    while current:
        if current.next:
            sys.stdout.write(str(current.val) + " -> ")
        else:
            sys.stdout.write(str(current.val) + "\n")
        current = current.next

# Helper function to create a linked list from a list of values
def create_linked_list(vals):
    temp = ListNode()
    current = temp
    for val in vals:
        current.next = ListNode(val)
        current = current.next
    return temp.next
```

```
53
54  #
55  # Complete the 'longer_list' function below.
56  #
57  # The function is expected to return an INTEGER_SINGLY_LINKED_LIST.
58  # The function accepts following parameters:
59  #  1. INTEGER_ARRAY head1
60  #  2. INTEGER_ARRAY head2
61  #
62
63  def longer_list(head1, head2):
64      # Write your code here
65      def find_length(head):
66          if not head:
67              return None
68          length = 0
69          current = head
70          while current:
71              length += 1
72              current = current.next
73          return length
74      len1 = find_length(head1)
75      len2 = find_length(head2)
76
77      return head1 if len1>=len2 else head2
78
79
80  if __name__ == '__main__':
81      #input_data = sys.stdin.read().strip()
82      #input_list = ast.literal_eval(input_data)
83
84      #head1 = create_linked_list(input_list[0])
85      #head2 = create_linked_list(input_list[1])
86
87      #result = longer_list(head1, head2)
88      #print_linked_list(result)
89
90      outfile = open(os.environ['OUTPUT_PATH'], 'w')
91
92      # Helper function to convert str -> linked list
93      def str_to_ll(vals_str):
94          if vals_str is None or vals_str == "None":
95              return None
96          vals = vals_str.split("->")
97          temp_head = Node("temp")
98          temp_curr = temp_head
```

```
        for val in vals:
            temp_curr.next = Node(val.strip())
            temp_curr = temp_curr.next
        return temp_head.next #Don't keep the temp head

    # Helper function to convert linked list -> str
    def ll_to_str(head):
        list_str = ""
        curr = head
        while curr:
            list_str += str(curr.val)
            if curr.next:
                list_str += "->"
            curr = curr.next
        if len(list_str) == 0:
            return "None"
        return list_str

    test_str = input()
    while(test_str != "END"):
        # Convert input string to list of param strings
        param_list = ast.literal_eval(test_str)

        # TODO: Edit parameters as needed
        head1 = str_to_ll(param_list[0])
        head2 = str_to_ll(param_list[1])

        # TODO: Edit function name and prepare result string
        result_raw = longer_list(head1, head2)
        result = ll_to_str(result_raw)

        # Write output and check for another test case
        outfile.write(str(result) + '\n')
        test_str = input()

    outfile.close()
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 0	Easy	Hidden	Success	0	0.0302 sec	11 KB

Both Lists Are Empty	Easy	Hidden	Success	0	0.028 sec	11 KB
One List Is Empty, the Other Is Not	Easy	Hidden	Success	0	0.0292 sec	11 KB
One List Is Empty, the Other Is Not	Easy	Hidden	Success	0	0.0285 sec	11 KB
Both Lists Have the Same Length 1 -> 2 -> 3	Easy	Hidden	Success	0	0.0299 sec	11 KB
Both Lists Have the Same Length 4 -> 5 -> 6	Easy	Hidden	Success	0	0.0302 sec	11 KB
Lists with One Element Each 7	Easy	Hidden	Success	0	0.028 sec	11 KB
Lists with One Element Each 8	Easy	Hidden	Success	0	0.0302 sec	11.1 KB
One List Is Longer Than the Other	Easy	Hidden	Success	0	0.0389 sec	10.9 KB
One List Is Longer Than the Other	Easy	Hidden	Success	0	0.0252 sec	11.1 KB
Lists with Negative and Positive Numbers	Easy	Hidden	Success	0	0.0313 sec	11.1 KB

Lists with Repeated Elements	Easy	Hidden	Success	0	0.0291 sec	11 KB
Single Element in Each List but Different Values	Easy	Hidden	Success	0	0.0289 sec	11 KB
Pass/Fail Case	Easy	Hidden	Success	20	0.0358 sec	11.1 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)

- ○ 1 -> 2 -> 3<!-- notionvc: 0042e942-202a-425f-82bc-4b645615a0a2 -->
- ● ** 1 -> 2<!-- notionvc: 5a98fe1f-1523-4b2c-9eb1-4885fcf50a20 --> ** ✓
- ○ 3 -> 2 -> 1<!-- notionvc: 91d5499f-e65e-4b33-9dc7-afbd00e2fb4f -->
- ○ None

ⓘ 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)

- ○ 1 -> 2 -> 3 -> 4 -> 5<!-- notionvc: ed66848e-4e4f-4baa-b77a-7e95d1c80a4e -->
- ○ 1 -> 2 -> 3 -> 4<!-- notionvc: 28341557-c82c-4431-a6e0-943274353b1a -->
- ○ 5 -> 4 -> 3 -> 2 -> 1<!-- notionvc: e342538b-b38e-4c90-93a9-727cb49fd119 -->
- ● ** 5 -> 2 -> 3 -> 4 -> 1<!-- notionvc: 108b5cd0-c143-4b28-bff7-a4ac9fce6777 --> ** ✓

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 -> 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**

```
#!/bin/python

import math
import os
import random
import re
import sys

class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
```

```
        self.next = next



def remove_duplicates(head: ListNode) -> ListNode:
    if not head:
        return None

    current = head
    while current and current.next:
        if current.val == current.next.val:
            current.next = current.next.next
        else:
            current = current.next

    return head

import sys

def parse_input():
    """
    Reads multiple lines of input, where each line represents a separate linked list.
    Returns a list of ListNode heads, one for each linked list.
    """
    lines = sys.stdin.read().strip().split("\n")  # Read all lines separately
    linked_lists = []

    for line in lines:
        if not line.strip():  # Handle empty lines (edge case)
            linked_lists.append(None)
            continue

        values = line.strip().split(" -> ")  # Split the linked list values
        nodes = [ListNode(int(val)) for val in values]

        for i in range(len(nodes) - 1):
            nodes[i].next = nodes[i + 1]  # Link nodes together

        linked_lists.append(nodes[0])  # Store the head of the linked list

    return linked_lists  # Return a list of linked lists

def print_linked_list(head):
    """
    Prints the linked list in the required format.
```

```
    """
    values = []
    while head:
        values.append(str(head.val))
        head = head.next
    print(" -> ".join(values))

if __name__ == "__main__":
    heads = parse_input()  # Get all linked list heads

    for head in heads:  # Process each linked list separately
        new_head = remove_duplicates(head)
        print_linked_list(new_head)  # Print each modified list on a new line

```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 0	Easy	Hidden	Success	0	0.0228 sec	10.1 KB
Testcase 1	Easy	Hidden	Success	0	0.0256 sec	10 KB
Testcase 2	Easy	Hidden	Success	0	0.0274 sec	10.1 KB
Testcase 3	Easy	Hidden	Success	0	0.0236 sec	10.3 KB
Testcase 4	Easy	Hidden	Success	5	0.0247 sec	10.1 KB

No comments.