



Solution Review: Union & Intersection of Linked Lists

This review provides a detailed analysis of the different ways to solve the Union and Intersection of Linked Lists challenge.

We'll cover the following ^

- Solution: Union
 - Time Complexity
- Solution: Intersection
 - Time Complexity

Solution: Union

main.py

LinkedList.py

Node.py

```
1 from LinkedList import LinkedList
2 from Node import Node
3
4
5 def union(list1, list2):
6     # Return other List if one of them is empty
7     if (list1.is_empty()):
8         return list2
9     elif (list2.is_empty()):
10        return list1
```

```
11
12     start = list1.get_head()
13
14     # Traverse the first list till the tail
15     while start.next_element:
16         start = start.next_element
17
18     # Link last element of first list to the first element of second
19     start.next_element = list2.get_head()
20     list1.remove_duplicates()
21     return list1
22
23
24 ulist1 = LinkedList()
25 ulist2 = LinkedList()
26 ulist1.insert_at_head(8)
27 ulist1.insert_at_head(22)
28 ulist1.insert_at_head(15)
```

Nothing too tricky going on here. We traverse to the tail of the first list and link it to the first node of the second list. All we have to do now is remove duplicates from the combined list.

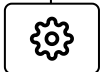
Time Complexity#

If we did not have to care for duplicates, The runtime complexity of this algorithm would be $O(m)$ where m is the size of the first list. However, because of duplicates, we need to traverse the whole union list. This increases the time complexity to $O(l^2)$ where $l = m + n$. Here, m is the size of the first list, and n is the size of the second list.

Solution: Intersection#

LinkedList.py

Node.py



```
from LinkedList import LinkedList
from Node import Node

def intersection(list1, list2):

    result = LinkedList()
    current_node = list1.get_head()

    # Traversing list1 and searching in list2
    # insert in result if the value exists
    while current_node is not None:
        value = current_node.data
        if list2.search(value) is not None:
            result.insert_at_head(value)
        current_node = current_node.next_element

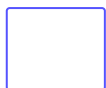
    # Remove duplicates if any
    result.remove_duplicates()
    return result

ilist1 = LinkedList()
ilist2 = LinkedList()

ilist1.insert_at_head(14)
ilist1.insert_at_head(22)
ilist1.insert_at_head(15)

ilist2.insert_at_head(21)
ilist2.insert_at_head(14)
ilist2.insert_at_head(15)

lst = intersection(ilist1, ilist2)
lst.print_list()
```



You are already familiar with this approach. We simply **list1** and search for all elements in **list2**. If any of these values are found in **list2**, it is added to the **result** linked list.



Since we insert at head, as shown on **line 25**, insert works in



Time Complexity#

The time complexity will be $\max(O(mn), O(\min(m, n)^2))$ where **m** is the size of the first list and **n** is the size of the second list.

The term $O(mn)$ comes from the lines **12-16**. We traverse the two lists searching the elements of **list1** in **list2**, and generate an intermediate list of size $\min(m, n)$.

Next, we remove the duplicates using the **remove_duplicates** method that takes **quadratic** time. Hence the term, $\min(m, n)^2$.

Note: The solution provided above is not the optimal solution for this problem. We can write a more efficient solution using hashing. We will cover that approach in [Hashing Chapter: Challenge 12](#)

If you've made it this far, you've become very experienced in the art of linked lists. Just one more challenge to go! See you there.

Interviewing soon? We've partnered with Hired so that companies apply to you instead of you applying to them. [See how](#) ⓘ



← Back

Next →




Challenge 9: Union & Intersection of Li...

Challenge 10: Return the Nth node fro...

✓

Mark as Completed

?

 Report an Issue

