Solution Review: Merge Two Sorted Lists

This review provides a detailed analysis of the different ways to merge two sorted lists.

We'll cover the following

Solution #1: Creating a New List#

```
# Merge list1 and list2 and return resulted list
def merge_lists(lst1, lst2):
    index_arr1 = 0
    index_arr2 = 0
    index_result = 0
    result = []

    for i in range(len(lst1)+len(lst2)):
        result.append(i)
    # Traverse Both lists and insert smaller value from arr1 or arr2
    # into result list and then increment that lists index.
    # If a list is completely traversed, while other one is left then just
    # copy all the remaining elements into result list
    while (index_arr1 < len(lst1)) and (index_arr2 < len(lst2)):
        if (lst1[index_arr1] < lst2[index_arr2]):
            result[index_result] = lst1[index_arr1]
            index_result += 1
            index_arr1 += 1
```

```
19          else:
20              result[index_result] = lst2[index_arr2]
21              index_result += 1
22              index_arr2 += 1
23      while (index_arr1 < len(lst1)):
24          result[index_result] = lst1[index_arr1]
25          index_result += 1
26          index_arr1 += 1
27      while (index_arr2 < len(lst2)):
28          result[index_result] = lst2[index_arr2]
```

The solution above is a more intuitive way to solve this problem. Start by creating a new empty list. This list will be filled with all the elements of both lists in sorted order and returned. Then initialize three variables to zero to store the current index of each list. Then compare the elements of the two given lists at the current index of each, append the smaller one to the new list and increment the index of that list by 1. Repeat until the end of one of the lists is reached and append the other list to the merged list.

Time Complexity#

The time complexity for this algorithm is $O(n + m)$ where n and m are the lengths of the lists. This is because both lists are iterated over atleast once.

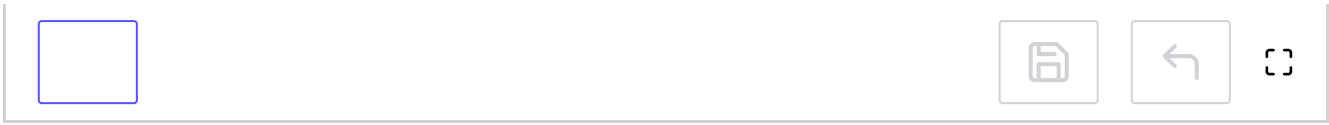
Solution #2: Merging in Place

```
def merge_arrays(lst1, lst2):
    ind1 = 0  # Creating 2 new variable to track the 'current index'
    ind2 = 0
    # While both indeces are less than the length of their lists
    while ind1 < len(lst1) and ind2 < len(lst2):
        # If the current element of list1 is greater
        # than the current element of list2
        if(lst1[ind1] > lst2[ind2]):
            # insert list2's current index to list1
            lst1.insert(ind1, lst2[ind2])
```

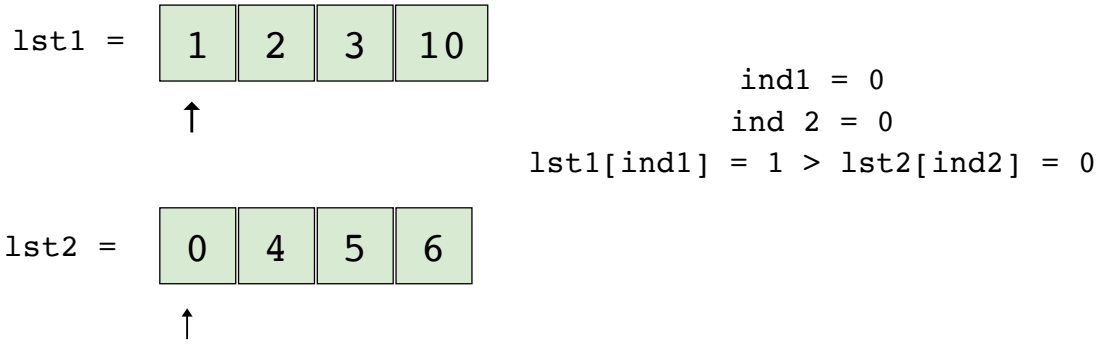
```
            ind1 += 1  # increment indices
            ind2 += 1
        else:
            ind1 += 1

    if ind2 < len(lst2):  # Append whatever is left of list2 to list1
        lst1.extend(lst2[ind2:])
    return lst1


print(merge_arrays([4, 5, 6], [-2, -1, 0, 7]))
```



This solution merges the two lists in place, i.e., no new list is created. First, initialize two new variables to track the ‘current index’ of both the lists to zero. Then, compare the current elements of both. If the current element of the first list is greater than the current element of the second list, insert the current element of the second list in place of the current element of the first list and increment both index variables by 1. Note that the insert operation is done using the built-in `insert` function. However, if the current element of the first list is *smaller* than the current element of the second list, then only increment the index variable of the first list by 1. Continue this until the end of one of the lists is reached, i.e., until one of the index variables is greater than or equal to the length of its respective list. After that, if the index of the second list is smaller than the length of the list, extend the first list by the second one from that index until the end.



1 of 8

Time Complexity#

Since both lists are traversed in this solution as well, the time complexity is $O(m(n+m))$ where n and m are the lengths of the lists. Both lists are not traversed separately so we cannot say that complexity is $(m+n)$. The shorter of the two lengths is traversed in the while loop. Also, the insert function gets called when the if-condition is true. In the worst-case, the second list has all the elements that are smaller than the elements of the first list. In this case, the complexity will be $O(mn)$.

> In order to further simplify the analysis, we could note that if m > n, we have $O(m^2)$, otherwise the complexity is O(n^2).

However, the extra space used in solution#1 is reduced to $O(m)$ in solution#2. Thus, it makes this a tradeoff between space and time.

Back

Challenge 2: Merge Two Sorted Lists

Next

Challenge 3: Find Two Numbers that ...

Completed

Report an Issue