

# Grokking the Coding Interview: Patterns for Coding Questions



### Pattern: Merge Intervals



## Pattern: Cyclic Sort

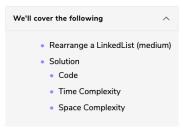


# Pattern: In-place Reversal of a LinkedList

Challenge 2

LinkeaList	
9	Introduction
0	Reverse a LinkedList (easy)
0	Reverse a Sub-list (medium)
	Reverse every K-element Sub-list (medium)
0	Problem Challenge 1
0	Solution Review: Problem Challenge 1
0	Problem Challenge 2
0	Solution Review: Problem

# Solution Review: Problem Challenge 2



# Rearrange a LinkedList (medium)

Given the head of a Singly LinkedList, write a method to modify the LinkedList such that the **nodes from the second half of the LinkedList are inserted alternately to the nodes from the first half in reverse order.** So if the LinkedList has nodes  $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 100$  null, your method should return  $1 \rightarrow 6 \rightarrow 2 \rightarrow 5 \rightarrow 3 \rightarrow 4 \rightarrow 100$ 

Your algorithm should not use any extra space and the input LinkedList should be modified in-place.

# Example 1:

```
Input: 2 -> 4 -> 6 -> 8 -> 10 -> 12 -> null
Output: 2 -> 12 -> 4 -> 10 -> 6 -> 8 -> null
```

#### Example 2:

```
Input: 2 -> 4 -> 6 -> 8 -> 10 -> null
Output: 2 -> 10 -> 4 -> 8 -> 6 -> null
```

### Solution

This problem shares similarities with Palindrome LinkedList. To rearrange the given LinkedList we will follow the following steps:

- We can use the Fast & Slow pointers method similar to Middle of the LinkedList to find the middle node
  of the LinkedList.
- $2. \ Once we have the middle of the LinkedList, we will reverse the second half of the LinkedList.\\$
- Finally, we'll iterate through the first half and the reversed second half to produce a LinkedList in the required order.

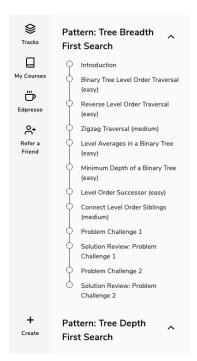
# Code

Here is what our algorithm will look like:

```
Python3
🐇 Java
                          @ C++
                                        ıs JS
         this.value = value;
this.next = next;
         temp = this;
          while (temp !== null) {
           process.stdout.write(`${temp.value} `);
            temp = temp.next;
      }
     function reorder(head) {
      if (head === null || head.next === null) {
       let slow = head,
         fast = head;
       while (fast !== null && fast.next !== null) {
         slow = slow.next;
fast = fast.next.n
                                                                                              SAVE
                                                                                                                    03
                                                                                                               Close
Output
 2 12 4 10 6 8
```







Time Complexity

The above algorithm will have a time complexity of O(N) where 'N' is the number of nodes in the LinkedList.

Space Complexity

The algorithm runs in constant space O(1).

Interviewing soon? We've partnered with Hired so that companies apply to you instead of you  $\quad\times$  applying to them. See how  $\odot$ 



