

← Back To Course Home

Grokking the Coding Interview: Patterns for Coding Questions

21% completed

🔍 Search Course

Solution Review: Problem Challenge 1

Problem Challenge 2

Solution Review: Problem Challenge 2

Problem Challenge 3

Solution Review: Problem Challenge 3

## Solution Review: Problem Challenge 2

We'll cover the following

- Rearrange a LinkedList (medium)
- Solution
  - Code
- Time Complexity
- Space Complexity

### 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 -> 2 -> 3 -> 4 -> 5 -> 6 -> null, your method should return 1 -> 6 -> 2 -> 5 -> 3 -> 4 -> null.

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.
- 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:

JavaPython3C++JS

```
1
2 class ListNode {
3     int value = 0;
4     ListNode next;
5
6     ListNode(int value) {
7         this.value = value;
8     }
9 }
10
11 class RearrangeList {
12
13     public static void reorder(ListNode head) {
14         if (head == null || head.next == null)
15             return;
16
17         // find the middle of the LinkedList
18         ListNode slow = head, fast = head;
19         while (fast != null && fast.next != null) {
20             slow = slow.next;
21             fast = fast.next.next;
22         }
23
24         // slow is now pointing to the middle node
25         ListNode headSecondHalf = reverse(slow); // reverse the second half
26         ListNode headFirstHalf = head;
27
28         // rearrange to produce the LinkedList in the required order
```

Run

SaveReset

### 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 the other way around. [See how](#)

×

← Back

Problem Challenge 2

Next →

Problem Challenge 3

✓

Mark as Completed

🚨

Report an Issue