

## Grokking the Coding Interview: Patterns for Coding Questions

28% completed

Challenge 1

Problem Challenge 2

Solution Review: Problem Challenge 2

Problem Challenge 3

Solution Review: Problem Challenge 3

### Pattern: In-place Reversal of a LinkedList

Introduction

Reverse a LinkedList (easy)

Reverse a Sub-list (medium)

Reverse every K-element Sub-list (medium)

Problem Challenge 1

Solution Review: Problem Challenge 1

Problem Challenge 2

Solution Review: Problem Challenge 2

### Pattern: Tree Breadth First Search

Introduction

Binary Tree Level Order Traversal (easy)

Reverse Level Order Traversal (easy)

Zigzag Traversal (medium)

Level Averages in a Binary Tree (easy)

Minimum Depth of a Binary Tree (easy)

Level Order Successor (easy)

Connect Level Order Siblings (medium)

Problem Challenge 1

Solution Review: Problem Challenge 1

Problem Challenge 2

Solution Review: Problem Challenge 2

### Pattern: Tree Depth First Search

Introduction

Binary Tree Path Sum (easy)

All Paths for a Sum (medium)

Sum of Path Numbers (medium)

Path With Given Sequence (medium)

Count Paths for a Sum (medium)

Problem Challenge 1

Solution Review: Problem Challenge 1

Problem Challenge 2

Solution Review: Problem Challenge 2

### Pattern: Two Heaps

## Reverse a LinkedList (easy)

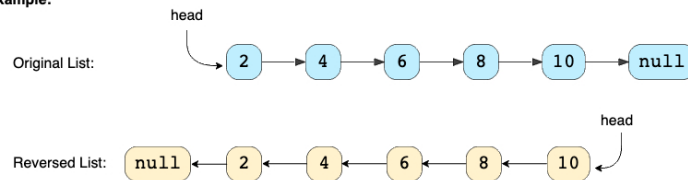
We'll cover the following

- Problem Statement
- Try it yourself
- Solution
  - Code
  - Time complexity
  - Space complexity

### Problem Statement

Given the head of a Singly LinkedList, reverse the LinkedList. Write a function to return the new head of the reversed LinkedList.

Example:



### Try it yourself

Try solving this question here:

JavaPython3JS C++

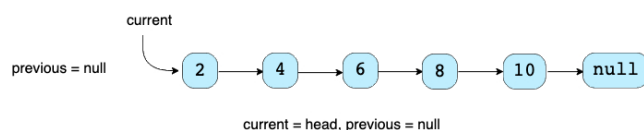
```
1 class Node {
2   constructor(value, next=null){
3     this.value = value;
4     this.next = next;
5   }
6
7   get_list() {
8     result = "";
9     temp = this;
10    while (temp !== null) {
11      result += temp.value + " ";
12      temp = temp.next;
13    }
14    return result;
15  }
16 };
17
18 const reverse = function(head) {
19   // TODO: Write your code here
20   return head;
21 };
22
23 head = new Node(2);
24 head.next = new Node(4);
25 head.next.next = new Node(6);
26 head.next.next.next = new Node(8);
27 head.next.next.next.next = new Node(10);
28 head.next.next.next.next.next = new Node(10);
```

RUNSAVERESET

### Solution

To reverse a LinkedList, we need to reverse one node at a time. We will start with a variable **current** which will initially point to the head of the LinkedList and a variable **previous** which will point to the previous node that we have processed; initially **previous** will point to **null**.

In a stepwise manner, we will reverse the **current** node by pointing it to the **previous** before moving on to the next node. Also, we will update the **previous** to always point to the previous node that we have processed. Here is the visual representation of our algorithm:



- Introduction
- Find the Median of a Number Stream (medium)
- Sliding Window Median (hard)
- Maximize Capital (hard)
- Problem Challenge 1
- Solution Review: Problem Challenge 1

## Pattern: Subsets

- Introduction
- Subsets (easy)
- Subsets With Duplicates (easy)
- Permutations (medium)
- String Permutations by changing case (medium)
- Balanced Parentheses (hard)
- Unique Generalized Abbreviations (hard)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2
- Problem Challenge 3
- Solution Review: Problem Challenge 3

## Pattern: Modified Binary Search

- Introduction
- Order-agnostic Binary Search (easy)
- Ceiling of a Number (medium)
- Next Letter (medium)
- Number Range (medium)
- Search in a Sorted Infinite Array (medium)
- Minimum Difference Element (medium)
- Bitonic Array Maximum (easy)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2
- Problem Challenge 3
- Solution Review: Problem Challenge 3

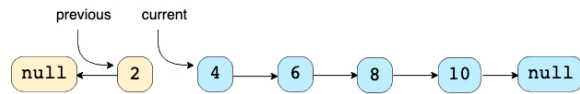
## Pattern: Bitwise XOR

- Introduction
- Single Number (easy)
- Two Single Numbers (medium)
- Complement of Base 10 Number (medium)
- Problem Challenge 1
- Solution Review: Problem Challenge 1

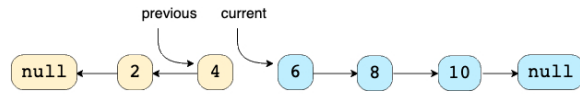
## Pattern: Top 'K' Elements

- Introduction
- Top 'K' Numbers (easy)
- Kth Smallest Number (easy)
- 'K' Closest Points to the Origin (easy)
- Connect Ropes (easy)
- Top 'K' Frequent Numbers (medium)

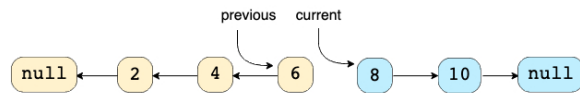
1 of 7



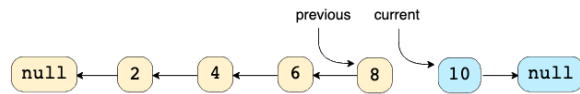
2 of 7



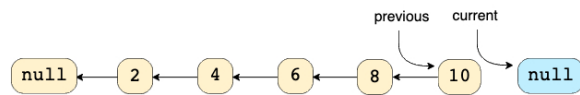
3 of 7



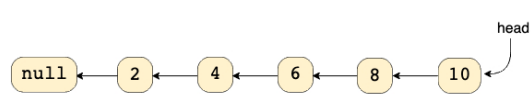
4 of 7



5 of 7



6 of 7



7 of 7

Code

Here is what our algorithm will look like:

Java Python3 C++ JS

```
1 class Node {
2   constructor(value, next = null) {
3     this.value = value;
4     this.next = next;
5   }
6
7   print_list() {
8     let temp = this;
9     while (temp !== null) {
10      process.stdout.write(`${temp.value} `);
11      temp = temp.next;
12    }
13    console.log();
14  }
15 }
16
17
18 function reverse(head) {
19   let current = head,
20     previous = null;
21   while (current !== null) {
22     next = current.next; // temporarily store the next node
23     current.next = previous; // reverse the current node
```

MW

Explore

Tracks

My Courses

Edpresso

Refer a Friend

Create

Frequency Sort (medium)

Kth Largest Number in a Stream (medium)

'K' Closest Numbers (medium)

Maximum Distinct Elements (medium)

Sum of Elements (medium)

Rearrange String (hard)

Problem Challenge 1

Solution Review: Problem Challenge 1

Problem Challenge 2

Solution Review: Problem Challenge 2

Problem Challenge 3

Solution Review: Problem Challenge 3

Pattern: K-way merge

Introduction

Merge K Sorted Lists (medium)

Kth Smallest Number in M Sorted Lists (Medium)

Kth Smallest Number in a Sorted Matrix (Hard)

Smallest Number Range (Hard)

Problem Challenge 1

Solution Review: Problem Challenge 1

```
24     previous = current; // before we move to the next node, point previous to the current node
25     current = next; // move on the next node
26 }
27 return previous;
28 }
```

RUN

SAVE

RESET

Time complexity

The time complexity of our algorithm will be  $O(N)$  where 'N' is the total number of nodes in the LinkedList.

Space complexity

We only used constant space, therefore, the space complexity of our algorithm is  $O(1)$ .

Interviewing soon? We've partnered with Hired so that companies apply to you, instead of the other way around. [See how](#)

← Back

Introduction

✓ MARK AS COMPLETED

Next →

Reverse a Sub-list (medium)

Report an Issue