LINKED LIST

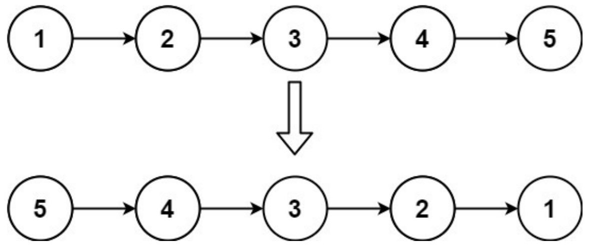
Problem – 206. Reverse Linked List

Easy

LeetCode leetcode.com/problems/reverse-linked-list

Problem

- This is a classic problem
- Given a singly linked list, reverse its order



Solution – 206. Reverse Linked List

Easy

LeetCode leetcode.com/problems/reverse-linked-list

Solution

- Use recursive approach
- Looking at the pseudo-code, this recursion will return the last node:

```
reverseList(head) {
    if (!head->next) return head
    node = reverseList(head->next);
    return node
}
```

- From end to beginning, each head will be a node in the list
- Therefore, you can change this node by setting a new head:

```
head->next->next = head;
head->next = nullptr;
```

Code – 206. Reverse Linked List

Easy

LeetCode leetcode.com/problems/reverse-linked-list

Code

Time: **O(n)** Space: **O(1)**

```
ListNode* reverseList(ListNode* head) {
    if (!head->next) return head;
    ListNode* node = reverseList(head->next);
    head->next->next = head;
    head->next = nullptr;
    return node;
}
```

Problem – 141. Linked List Cycle

Easy

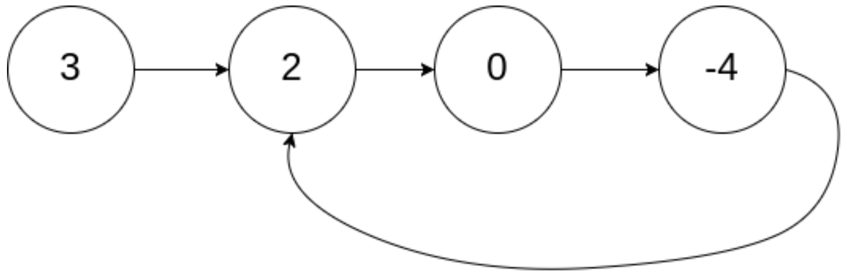
LeetCode leetcode.com/problems/linked-list-cycle

Problem

- You are given the head of a linked list
- Return **true** if there is a cycle, false otherwise
- **Example:**

  In the image below, there is a cycle (-4 to 2)

  Output: true



Solution – 141. Linked List Cycle

Easy

LeetCode leetcode.com/problems/linked-list-cycle

Solution

- Have two pointers: fast and slow
- Slow will go over each item in the linked list
- Fast will go twice as fast as slow (`fast = fast->next->next`)
- If fast reach at the end, there is no cycle
- If fast encounter slow, there is a cycle, return true

Code – 141. Linked List Cycle

Easy

LeetCode leetcode.com/problems/linked-list-cycle

Code

Time: **O(n)** Space: **O(1)**

```
bool hasCycle(ListNode *head) {
    if (!head || !head->next) return false;
    ListNode* slow = head;
    ListNode* fast = head;
    while (fast && fast->next) {
        slow = slow->next;
        fast = fast->next->next;
        if (slow == fast) return true;
    }
    return false;
}
```

Problem – 21. Merge Two Sorted Lists

Easy

LeetCode leetcode.com/problems/merge-two-sorted-lists

Problem Statement / Solution / Code

Time: **O(n)** Space: **O(n)**

- …

Problem – 23. Merge k Sorted Lists

Hard

LeetCode leetcode.com/problems/merge-k-sorted-lists

Problem Statement / Solution / Code

Time: **O(n)** Space: **O(n)**

- …

Problem – 19. Remove Nth Node From End of List

Medium

LeetCode leetcode.com/problems/remove-nth-node-from-end-of-list

Problem Statement / Solution / Code

Time: **O(n)** Space: **O(n)**

- …

Problem – 143. Reorder List

Medium

LeetCode leetcode.com/problems/reorder-list

Problem Statement / Solution / Code

Time: **O(n)** Space: **O(n)**

- …

Problem – 24. Swap Nodes in Pair

Medium

https://leetcode.com/problems/swap-nodes-in-pairs

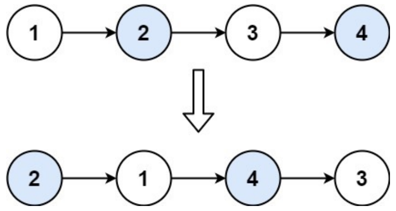
Problem

Given a linked list, swap every two adjacent nodes and return its head. You must solve the problem without modifying the values in the list's nodes (i.e., only nodes themselves may be changed.)

Example 1

Input: head = [1,2,3,4]

Output: [2,1,4,3]



Example 2

Input: head = []

Output: []

Example 3:

Example 3

Input: head = [1]

Output: [1]

Solution – Swap Nodes in Pair

Medium

https://leetcode.com/problems/swap-nodes-in-pairs

```
ListNode* swapPairs(ListNode* head) {
    if (head == NULL || head->next == NULL) {
        return head;
    }
    ListNode *node = head;
    ListNode *prev = NULL;
    head = head->next;

    while (node && node->next) {
        ListNode *second = node->next;
        ListNode *next_pair = second->next;
        second->next = node;
        node->next = next_pair;
        if (prev) {
            prev->next = second;
        }
        prev = node;
        node = next_pair;
    }
    return head;
}
```

Solution (recursive) – Swap Nodes in Pair

Medium

https://leetcode.com/problems/swap-nodes-in-pairs

```
ListNode* swapPairs(ListNode* head) {
    if(!head || !head->next)
        return head;
    ListNode* newHead = head->next;
    head->next = swapPairs(head->next->next);
    newHead->next = head;
    return newHead;
}
```