Contents

[1. Reverse Linked List in k groups 2](#_Toc463530523)

[2. Sort Linked list using Merge sort 2](#_Toc463530524)

[3. Find intersection of two LinkedLists 2](#_Toc463530525)

[4. Valid Parentheses 3](#_Toc463530526)

[5. WildCard 3](#_Toc463530527)

[6. Longest Valid Parentheses 3](#_Toc463530528)

[7. New problem 4](#_Toc463530529)

# Reverse Linked List in k groups

<todo>

# Sort Linked list using Merge sort

**Tags:** LinkedList, Sorting, Merge sort, LeetCode

Merge sort’s merge operation usually requires additional space for merging into new array. But, using LinkstList that need can be avoided.

public int getLen(ListNode h) { }

public ListNode merge(ListNode l, ListNode r) { }

public ListNode mergeRecursive(ListNode l, ListNode r)

{

// Stack overflow problem on LeetCode with around 31k size length

}

public ListNode sortList(ListNode head, int len)

{

int mid = len/2;

ListNode left = sortList(firstHalf, mid);

ListNode right = sortList(secondHalf, len - mid);

return merge(left, right);

}

# Find intersection of two LinkedLists

**Tags:** LinkedList, LeetCode

Given that m and n are lengths of two LLs -

1. Time O(m\*n); Space O(1)

For each node in m, search entire LinkedList n. If you find a match return that node.

1. **Time O(m+n); Space O(1)**

Find length of each list; Then for longest list traverse abs(m-n) nodes; last step - traverse both lists one at a time till they are same;

1. Time O(m+n); space O(m+n)

Using hashing (set can be used too)

Traverse first list and hash each node; for second list while traversing each node check in hash if it is present; if matched report that node as intersection

1. Time O(m+n); space O(m+n)

Use stack; Idea is use the fact that both list share same size from the end.

While traversing both the list put them in 2 separate stacks; now start poping each element from two stacks simultaneously, if they match report that node as intersection.

# Valid Parentheses

**Tags:** Stack, LeetCode

Given a string containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

The brackets must close in the correct order, "()" and "()[]{}" are all valid but "(]" and "([)]" are not

Solution:

Use a stack to push (, { and ]. Pop during closing brackets.

# WildCard

**Tags:** Recursion

Given a string 123?, replace ? with ‘0’ or ‘1’. So, output will be 1230, 1231

# Longest Valid Parentheses

**Tags:** Stack, LeetCode

**Problem statement:**

Given a string containing just the characters '(' and ')', find the length of the longest valid (well-formed) parentheses substring.

For "(()", the longest valid parentheses substring is "()", which has length = 2.

Another example is ")()())", where the longest valid parentheses substring is "()()", which has length = 4.

**Solution [1]**: Time O(n); Space O(n)

Keep pushing **index** of ‘(‘ in the stack. On finding ‘)’ pop the index and calculate the length (current\_index – index\_of\_top\_element\_from\_stack)

# New problem