**Justin\_LinkedList\_DepthfirstSearch\_0109.**  **Convert Sorted List to Binary Search Tree**

**Concept:**

先用兩個快慢 Pointer，把中心點找出來

再建一棵樹，Root 為中心點，左右子樹分別為 Linked List 中心點的兩邊

**Code:**

# Definition for singly-linked list.

# class ListNode:

# def \_\_init\_\_(self, val=0, next=None):

# self.val = val

# self.next = next

# Definition for a binary tree node.

# class TreeNode:

# def \_\_init\_\_(self, val=0, left=None, right=None):

# self.val = val

# self.left = left

# self.right = right

class Solution:

def findMiddle(self, head):

# The pointer used to disconnect the left half from the mid node.

prevPtr = None

slowPtr = head

fastPtr = head

# Iterate until fastPr doesn't reach the end of the linked list.

while fastPtr and fastPtr.next:

prevPtr = slowPtr

slowPtr = slowPtr.next

fastPtr = fastPtr.next.next

# Handling the case when slowPtr was equal to head.

if prevPtr:

prevPtr.next = None

return slowPtr

def sortedListToBST(self, head: ListNode) -> TreeNode:

# If the head doesn't exist, then the linked list is empty

if not head:

return None

# Find the middle element for the list.

mid = self.findMiddle(head)

# The mid becomes the root of the BST.

node = TreeNode(mid.val)

# Base case when there is just one element in the linked list

if head == mid:

return node

# Recursively form balanced BSTs using the left and right halves of the original list.

node.left = self.sortedListToBST(head)

node.right = self.sortedListToBST(mid.next)

return node