LC 21

class Solution:

def mergeTwoLists(self, l1: ListNode, l2: ListNode) -> ListNode:

if not l1:

return l2

elif not l2:

return l1

if l1.val <= l2.val:

return ListNode(val=l1.val, next=self.mergeTwoLists(l1.next,l2))

else:

return ListNode(val=l2.val, next=self.mergeTwoLists(l1,l2.next))

LC 1545

class Solution:

def findKthBit(self, n: int, k: int) -> str:

if n == 1 or k == 1:

return '0'

threshold = 2 \*\* (n-1)

if k < threshold:

return self.findKthBit(n-1,k)

elif k == threshold:

return '1'

else:

return str(1-int(self.findKthBit(n-1,2\*\*n-k)))

LC 894

class Solution:

def allPossibleFBT(self, n: int) -> List[TreeNode]:

if n % 2 == 0:

return []

if n == 1:

return [TreeNode()]

res = []

for i in range(1,n,2):

j = n-1-i

lst1 = self.allPossibleFBT(i)

lst2 = self.allPossibleFBT(j)

for l1 in lst1:

for l2 in lst2:

res.append(TreeNode(left=l1,right=l2))

return res

LC 698

class Solution:

def canPartitionKSubsets(self, nums: List[int], k: int) -> bool:

if k == 1:

return True

if len(nums) < k:

return False

s = sum(nums)

if s % k:

return False

nums.sort()

target = s // k - nums[-1]

nums.pop()

if target == 0:

return self.canPartitionKSubsets(nums,k-1)

elif target < nums[0]:

return False

elif target in nums:

nums.remove(target)

return self.canPartitionKSubsets(nums,k-1)

l = len(nums)

ind = l-1

while nums[ind]>target:

ind -= 1

ind += 1

r = 2 \*\* ind

lst = []

for n in range(1,r):

j = format(n,'0'+str(ind)+'b')

temp = [i for i in range(ind) if j[i] == '1']

if sum([nums[i] for i in temp]) == target:

lst.append([nums[i] for i in range(l) if i not in temp])

for l in lst:

if self.canPartitionKSubsets(l,k-1):

return True

return False