**Assignment 2**

1. Container With Most Water (Two Pointers):

def max\_area(heights):

left, right = 0, len(heights) - 1

max\_water = 0

while left < right:

area = min(heights[left], heights[right]) \* (right - left)

max\_water = max(max\_water, area)

if heights[left] < heights[right]:

left += 1

else:

right -= 1

return max\_water

2. Integer to Roman:

def int\_to\_roman(num):

roman\_dict = {

1000: "M",

900: "CM",

500: "D",

400: "CD",

100: "C",

90: "XC",

50: "L",

40: "XL",

10: "X",

9: "IX",

5: "V",

4: "IV",

1: "I"

}

result = ""

for value, symbol in roman\_dict.items():

while num >= value:

result += symbol

num -= value

return result

3. Roman to Integer:

def roman\_to\_int(s):

roman\_dict = {

"M": 1000,

"CM": 900,

"D": 500,

"CD": 400,

"C": 100,

"XC": 90,

"L": 50,

"XL": 40,

"X": 10,

"IX": 9,

"V": 5,

"IV": 4,

"I": 1

}

result = 0

i = 0

while i < len(s):

if i + 1 < len(s) and s[i] in roman\_dict and s[i+1] in roman\_dict and

roman\_dict[s[i]] < roman\_dict[s[i+1]]:

result += roman\_dict[s[i] + s[i+1]]

i += 2

else:

result += roman\_dict[s[i]]

i += 1

return result

4. Longest Common Prefix (Horizontal Scanning):

def longest\_common\_prefix(strs):

if not strs:

return ""

prefix = strs[0]

for string in strs[1:]:

while prefix and string and prefix[0] != string[0]:

prefix = prefix[1:]

return prefix

5. 3Sum (Two Pointers):

Python

def three\_sum(nums):

nums.sort()

result = []

for i in range(len(nums) - 2):

if i > 0 and nums[i] == nums[i-1]: # skip duplicates

continue

left, right = i + 1, len(nums) - 1

while left < right:

sum = nums[i] + nums[left] + nums[right]

if sum == 0:

result.append([nums[i], nums[left], nums[right]])

left += 1

while left < right and nums[left] == nums[left - 1]: # skip

duplicates

left += 1

elif sum < 0:

left += 1

else:

right -= 1

return result

6. 3Sum Closest (Two Pointers):

def three\_sum\_closest(nums, target):

nums.sort()

closest\_sum = float('inf') # Use infinity to track the closest sum

for i in range(len(nums) - 2):

if i > 0 and nums[i] == nums[i-1]: # skip duplicates

continue

left, right = i + 1, len(nums) - 1

while left < right:

current\_sum = nums[i] + nums[left] + nums[right]

diff = abs(target - current\_sum) # absolute difference to find

closeness

if diff == 0:

return current\_sum # return the exact target sum if found

elif diff < closest\_sum:

closest\_sum = diff

if current\_sum < target:

left += 1

else:

right -= 1

return closest\_sum

7. Letter Combinations of a Phone Number (Recursion):

def letter\_combinations(digits):

if not digits:

return []

phone\_dict = {

'2': 'abc',

'3': 'def',

'4': 'ghi',

'5': 'jkl',

'6': 'mno',

'7': 'pqrs',

'8': 'tuv',

'9': 'wxyz'

}

def backtrack(index, combination, result):

if index == len(digits):

result.append(combination)

return

current\_digit = digits[index]

for letter in phone\_dict[current\_digit]:

backtrack(index + 1, combination + letter, result)

result = []

backtrack(0, "", result)

return result

8. 4Sum (Nested Loops):

def four\_sum(nums, target):

nums.sort()

result = []

for i in range(len(nums) - 3):

if i > 0 and nums[i] == nums[i-1]: # skip duplicate quadruplets with

the same first element

continue

for j in range(i + 1, len(nums) - 2):

if j > i + 1 and nums[j] == nums[j-1]: # skip duplicate quadruplets

with the same second element

continue

left, right = j + 1, len(nums) - 1

while left < right:

current\_sum = nums[i] + nums[j] + nums[left] + nums[right]

if current\_sum == target:

result.append([nums[i], nums[j], nums[left], nums[right]])

left += 1

while left < right and nums[left] == nums[left - 1]: # skip

duplicates as third element

left += 1

elif current\_sum < target:

left += 1

else:

right -= 1

return result

9. Remove Nth Node From End of List (Two Pointers):

def remove\_nth\_from\_end(head, n):

dummy = ListNode(0) # create a dummy node to handle edge cases

dummy.next = head

fast, slow = dummy, dummy

for \_ in range(n):

fast = fast.next

while fast and fast.next:

slow = slow.next

fast = fast.next

slow.next = slow.next.next

return dummy.next

Use code with caution.

content\_copy

10. Valid Parentheses (Stack):

def is\_valid(s):

opening\_parens = {

'(': ')',

'{': '}',

'[': ']'

}

stack = []

for char in s:

if char in opening\_parens:

stack.append(char)

else:

if not stack or opening\_parens[stack.pop()] != char:

return False

return not stack