**1.Two Sum**

def two\_sum(nums, target):

num\_to\_index = {}

for i, num in enumerate(nums):

complement = target - num

if complement in num\_to\_index:

return [num\_to\_index[complement], i]

num\_to\_index[num] = i

return []

nums1 = [2, 7, 11, 15]

target1 = 9

print(two\_sum(nums1, target1))

output: [0, 1]

**2. Add Two Numbers (Linked List)**

class ListNode:

def init(self, val=0, next=None):

self.val = val

self.next = next

def add\_two\_numbers(l1, l2):

dummy = ListNode()

current, carry = dummy, 0

while l1 or l2 or carry:

val1 = l1.val if l1 else 0

val2 = l2.val if l2 else 0

carry, out = divmod(val1 + val2 + carry, 10)

current.next = ListNode(out)

current = current.next

l1 = l1.next if l1 else None

l2 = l2.next if l2 else None

return dummy.next

def create\_linked\_list(lst):

dummy = ListNode()

current = dummy

for num in lst:

current.next = ListNode(num)

current = current.next

return dummy.next

l1 = create\_linked\_list([2, 4, 3])

l2 = create\_linked\_list([5, 6, 4])

result = add\_two\_numbers(l1, l2)

output = []

while result:

output.append(result.val)

result = result.next

print(output)

output :

[7, 0, 8]

**3. Longest Substring Without Repeating Characters**

def length\_of\_longest\_substring(s):

char\_index = {}

left = max\_length = 0

for right, char in enumerate(s):

if char in char\_index and char\_index[char] >= left:

left = char\_index[char] + 1

char\_index[char] = right

max\_length = max(max\_length, right - left + 1)

return max\_length

s1 = "abcabcbb"

print(length\_of\_longest\_substring(s1))

output :

3

**4. Median of Two Sorted Arrays**

def find\_median\_sorted\_arrays(nums1, nums2):

def find\_kth\_smallest(a, b, k):

if len(a) > len(b):

a, b = b, a

if not a:

return b[k]

if k == len(a) + len(b) - 1:

return max(a[-1], b[-1])

i = len(a) // 2

j = k - i

if a[i] > b[j]:

return find\_kth\_smallest(a[:i], b[j:], i)

else:

return find\_kth\_smallest(a[i:], b[:j], j)

l = len(nums1) + len(nums2)

if l % 2 == 1:

return find\_kth\_smallest(nums1, nums2, l // 2)

else:

return (find\_kth\_smallest(nums1, nums2, l // 2 - 1) + find\_kth\_smallest(nums1, nums2, l // 2)) / 2

nums1\_1 = [1, 3]

nums2\_1 = [2]

print(find\_median\_sorted\_arrays(nums1\_1, nums2\_1))

output :

2

**5. Longest Palindromic Substring**

def longest\_pali ndromic\_substring(s):

n = len(s)

if n == 0:

return ""

longest = s[0]

for i in range(n):

odd = expand\_around\_center(s, i, i)

even = expand\_around\_center(s, i, i + 1)

longer = odd if len(odd) > len(even) else even

if len(longer) > len(longest):

longest = longer

return longest

def expand\_around\_center(s, left, right):

while left >= 0 and right < len(s) and s[left] == s[right]:

left -= 1

right += 1

return s[left + 1:right]

s1 = "babad"

print(longest\_palindromic\_substring(s1))

output:

bab

**6. Zigzag Conversion**

def parse\_int(s: str) -> int:

i = 0

while i < len(s) and s[i] == ' ':

i += 1

sign = 1

if i < len(s) and s[i] == '-':

sign = -1

i += 1

elif i < len(s) and s[i] == '+':

i += 1

num = 0

while i < len(s) and s[i].isdigit():

num = num \* 10 + int(s[i])

i += 1

num \*= sign

INT\_MIN, INT\_MAX = -2\*31, 2\*31 - 1

if num < INT\_MIN:

return INT\_MIN

if num > INT\_MAX:

return INT\_MAX

return num

examples = ["42", " -42", "4193 with words"]

outputs = [parse\_int(s) for s in examples]

return outputs

output :

[42, -42, 4193]

**7. Reverse Integer**

def convert(s: str, numRows: int) -> str:

if numRows == 1 or numRows >= len(s):

return s

rows = [''] \* numRows

current\_row, step = 0, -1

for char in s:

rows[current\_row] += char

if current\_row == 0 or current\_row == numRows - 1:

step = -step

current\_row += step

return ''.join(rows)

examples = [("PAYPALISHIRING", 3), ("PAYPALISHIRING", 4), ("A", 1)]

outputs = [convert(s, numRows) for s, numRows in examples]

print(outputs)

output :

['PAHNAPLSIIGYIR', 'PINALSIGYAHRPI', 'A']

**8. String to Integer (atoi)**

def reverse(x: int) -> int:

sign = -1 if x < 0 else 1

x = abs(x)

reversed\_x = 0

while x != 0:

reversed\_x = reversed\_x \* 10 + x % 10

x //= 10

reversed\_x \*= sign

if reversed\_x < -2\*31 or reversed\_x > 2\*31 - 1:

return 0

return reversed\_x

examples = [123, -123, 120]

outputs = [reverse(x) for x in examples]

print(outputs)

output :

[321, -321, 21]

**9. Palindrome Number**

def is\_palindrome(x: int) -> bool:

if x < 0:

return False

return str(x) == str(x)[::-1]

examples = [121, -121, 10]

outputs = [is\_palindrome(x) for x in examples]

print(outputs)

output :

[True, False, False]

**10. Regular Expression Matching**

import re

def is\_match(s: str, p: str) -> bool:

pattern = re.compile(f"^{p}$")

return bool(pattern.match(s))

examples = [("aa", "a"), ("aa", "a\*"), ("ab", ".\*")]

outputs = [is\_match(s, p) for s, p in examples]

print(outputs)

output :

[False, True, True]

**11. Container With Most Water**

def max\_area(height: list) -> int:

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

max\_area = 0

while left < right:

width = right - left

max\_area = max(max\_area, min(height[left], height[right]) \* width)

if height[left] < height[right]:

left += 1

else:

right -= 1

return max\_area

examples = [[1,8,6,2,5,4,8,3,7], [1,1]]

outputs = [max\_area(height) for height in examples]

print(outputs)

output :

[49, 1]

**12. Integer to Roman**

def int\_to\_roman(num: int) -> str:

val = [

1000, 900, 500, 400,

100, 90, 50, 40,

10, 9, 5, 4,

1

]

syms = [

"M", "CM", "D", "CD",

"C", "XC", "L", "XL",

"X", "IX", "V", "IV",

"I"

]

roman\_num = ''

i = 0

while num > 0:

for \_ in range(num // val[i]):

roman\_num += syms[i]

num -= val[i]

i += 1

return roman\_num

examples = [3, 58, 1994]

outputs = [int\_to\_roman(num) for num in examples]

print(outputs)

output :

['III', 'LVIII', 'MCMXCIV']

**13. Roman to Integer**

def roman\_to\_int(s: str) -> int:

roman = {'I': 1, 'V': 5, 'X': 10, 'L': 50, 'C': 100, 'D': 500, 'M': 1000}

result = 0

prev\_value = 0

for char in reversed(s):

value = roman[char]

if value < prev\_value:

result -= value

else:

result += value

prev\_value = value

return result

examples = ["III", "LVIII", "MCMXCIV"]

outputs = [roman\_to\_int(s) for s in examples]

print(outputs)

output:

[3, 58, 1994]

**14. Longest Common Prefix**

def longest\_common\_prefix(strs: list) -> str:

if not strs:

return ""

prefix = strs[0]

for string in strs[1:]:

while not string.startswith(prefix):

prefix = prefix[:-1]

if not prefix:

return ""

return prefix

examples = [["flower", "flow", "flight"], ["dog", "racecar", "car"]]

outputs = [longest\_common\_prefix(strs) for strs in examples]

print(outputs)

output:

['fl', '']

**15. find all unique triplets that sum up to zero**

def three\_sum(nums: list) -> list:

nums.sort()

result = []

n = len(nums)

for i in range(n):

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

continue

left, right = i + 1, n - 1

while left < right:

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

if total < 0:

left += 1

elif total > 0:

right -= 1

else:

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

while left < right and nums[left] == nums[left + 1]:

left += 1

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

right -= 1

left += 1

right -= 1

return result

examples = [[-1, 0, 1, 2, -1, -4], [0, 1, 1], [0, 0, 0]]

outputs = [three\_sum(nums) for nums in examples]

print(outputs)

output :

[[[-1, -1, 2], [-1, 0, 1]], [], [[0, 0, 0]]]

**16. 3Sum Closest**

def three\_sum\_closest(nums: list, target: int) -> int:

nums.sort()

closest\_sum = float('inf')

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

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

while left < right:

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

if abs(current\_sum - target) < abs(closest\_sum - target):

closest\_sum = current\_sum

if current\_sum < target:

left += 1

elif current\_sum > target:

right -= 1

else:

return current\_sum

return closest\_sum

examples = [([-1, 2, 1, -4], 1), ([0, 0, 0], 1)]

outputs = [three\_sum\_closest(nums, target) for nums, target in examples]

print(outputs)

output :

[2, 0]

**17. Letter Combinations of a Phone Number**

def letter\_combinations(digits: str) -> list:

if not digits:

return []

phone\_map = {

'2': 'abc', '3': 'def', '4': 'ghi', '5': 'jkl',

'6': 'mno', '7': 'pqrs', '8': 'tuv', '9': 'wxyz'

}

def backtrack(index, path):

if len(path) == len(digits):

combinations.append("".join(path))

return

possible\_letters = phone\_map[digits[index]]

for letter in possible\_letters:

path.append(letter)

backtrack(index + 1, path)

path.pop()

combinations = []

backtrack(0, [])

return combinations

examples = ["23", "", "2"]

outputs = [letter\_combinations(digits) for digits in examples]

print(outputs)

output :

[['ad', 'ae', 'af', 'bd', 'be', 'bf', 'cd', 'ce', 'cf'], [], ['a', 'b', 'c']]

**18. Find all unique quadruplets that sum up to the target**

def four\_sum(nums: list, target: int) -> list:

nums.sort()

result = []

n = len(nums)

for i in range(n - 3):

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

continue

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

if j > i + 1 and nums[j] == nums[j - 1]:

continue

left, right = j + 1, n - 1

while left < right:

total = nums[i] + nums[j] + nums[left] + nums[right]

if total < target:

left += 1

elif total > target:

right -= 1

else:

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

while left < right and nums[left] == nums[left + 1]:

left += 1

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

right -= 1

left += 1

right -= 1

return result

examples = [([1, 0, -1, 0, -2, 2], 0), ([2, 2, 2, 2, 2], 8)]

outputs = [four\_sum(nums, target) for nums, target in examples]

print(outputs)

output : [[[-2, -1, 1, 2], [-2, 0, 0, 2], [-1, 0, 0, 1]], [[2, 2, 2, 2]]]

**19. Remove Nth Node From End of List**

class ListNode:

def init(self, val=0, next=None):

self.val = val

self.next = next

def remove\_nth\_from\_end(head: ListNode, n: int) -> ListNode:

dummy = ListNode(0, head)

first = second = dummy

for \_ in range(n + 1):

first = first.next

while first:

first = first.next

second = second.next

second.next = second.next.next

return dummy.next

# Helper function to create linked list from list and convert linked list to list

def create\_linked\_list(lst):

dummy = ListNode()

current = dummy

for val in lst:

current.next = ListNode(val)

current = current.next

return dummy.next

def linked\_list\_to\_list(head):

lst = []

current = head

while current:

lst.append(current.val)

current = current.next

return lst

examples = [([1, 2, 3, 4, 5], 2), ([1], 1), ([1, 2], 1)]

outputs = [linked\_list\_to\_list(remove\_nth\_from\_end(create\_linked\_list(lst), n)) for lst, n in examples]

print(outputs)

output :

[[1, 2, 3, 5], [], [1]]

**20. Valid Parentheses**

def is\_valid(s: str) -> bool:

stack = []

mapping = {')': '(', '}': '{', ']': '['}

for char in s:

if char in mapping:

top\_element = stack.pop() if stack else '#'

if mapping[char] != top\_element:

return False

else:

stack.append(char)

return not stack

examples = ["()", "()[]{}", "(]"]

outputs = [is\_valid(s) for s in examples]

print(outputs)

output :

[True, True, False]