# Assignment 5

1.Two sum:

def two\_sum(nums, target):

num\_dict = {}

for i, num in enumerate(nums):

complement = target - num

if complement in num\_dict:

return [num\_dict[complement], i]

num\_dict[num] = i

2.add two numbers:

class ListNode:

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

self.val = val

self.next = next

def addTwoNumbers(l1, l2):

dummy = ListNode(0)

current = dummy

carry = 0

while l1 or l2 or carry:

sum\_val = (l1.val if l1 else 0) + (l2.val if l2 else 0) + carry

carry = sum\_val // 10

current.next = ListNode(sum\_val % 10)

current = current.next

l1 = l1.next if l1 else None

l2 = l2.next if l2 else None

return dummy.next

3. Longest Substring without Repeating Characters:

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

start = maxLength = 0

usedChars = {}

for i in range(len(s)):

if s[i] in usedChars and start <= usedChars[s[i]]:

start = usedChars[s[i]] + 1

else:

maxLength = max(maxLength, i - start + 1)

usedChars[s[i]] = i

return maxLength

s1 = "abcabcbb"

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

4.Median of two sorted arrays:

def findMedianSortedArrays(nums1, nums2):

nums = sorted(nums1 + nums2)

n = len(nums)

if n % 2 == 0:

return (nums[n // 2 - 1] + nums[n // 2]) / 2

else:

return nums[n // 2]

nums1 = [1, 3]

nums2 = [2]

print(findMedianSortedArrays(nums1, nums2))

5.Longest palindrome string:

class Solution:

def longestPalindrome(self, s: str) -> str:

def expandAroundCenter(left, right):

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

left -= 1

right += 1

return s[left + 1:right]

if len(s) == 0:

return ""

longest = ""

for i in range(len(s)):

odd\_palindrome = expandAroundCenter(i, i)

even\_palindrome = expandAroundCenter(i, i + 1)

longest = max(longest, odd\_palindrome, even\_palindrome, key=len)

return longest

s = "babad"

solution = Solution()

print(solution.longestPalindrome(s))

6.Zig zag conversation:

def convert(s, numRows):

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

return s

rows = [''] \* numRows

index, step = 0, 1

for char in s:

rows[index] += char

if index == 0:

step = 1

elif index == numRows - 1:

step = -1

index += step

return ''.join(rows)

s = "PAYPALISHIRING"

numRows = 3

output = convert(s, numRows)

print(output)

7.Reverse integer:

class Solution:

def reverse(self, x: int) -> int:

if x < 0:

sign = -1

else:

sign = 1

x = abs(x)

reverse\_x = int(str(x)[::-1])

if reverse\_x > 2\*\*31 - 1:

return 0

return sign \* reverse\_x

8.String to integer:

def myAtoi(s):

s = s.strip()

if not s:

return 0

sign = 1

if s[0] == '-':

sign = -1

s = s[1:]

elif s[0] == '+':

s = s[1:]

num = 0

for char in s:

if not char.isdigit():

break

num = num \* 10 + int(char)

num = max(-2\*\*31, min(sign \* num, 2\*\*31 - 1))

return num

9.Palindrome number:

class Solution:

def isPalindrome(self, x: int) -> bool:

if x < 0:

return False

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

x = 121

sol = Solution()

print(sol.isPalindrome(x))

10.Regular expression making

class Solution:

def isMatch(self, s: str, p: str) -> bool:

if not p:

return not s

first\_match = bool(s) and p[0] in {s[0], '.'}

if len(p) >= 2 and p[1] == '\*':

return (self.isMatch(s, p[2:]) or

first\_match and self.isMatch(s[1:], p))

else:

return first\_match and self.isMatch(s[1:], p[1:])

solution = Solution()

s = "aa"

p = "a"

print(solution.isMatch(s, p))

11.Containing with most water:

def max\_area(height):

max\_area = 0

left = 0

right = len(height) - 1

while left < right:

width = right - left

h = min(height[left], height[right])

max\_area = max(max\_area, width \* h)

if height[left] < height[right]:

left += 1

else:

right -= 1

return max\_area

height1 = [1, 8, 6, 2, 5, 4, 8, 3, 7]

print(max\_area(height1))

12.Integer to roman:

class Solution:

def intToRoman(self, 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

num = 3

solution = Solution()

output = solution.intToRoman(num)

print(output)

13.Roman to integer:

def romanToInt(s):

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

result = 0

prev\_value = 0

for char in s:

value = roman\_dict[char]

if value > prev\_value:

result += value - 2 \* prev\_value

else:

result += value

prev\_value = value

return result

print(romanToInt("III"))

print(romanToInt("LVIII"))

14.Longest common prefix:

def longestCommonPrefix(strs):

if not strs:

return ""

strs.sort()

prefix = ""

for i in range(len(strs[0])):

if strs[0][i] == strs[-1][i]:

prefix += strs[0][i]

else:

break

return prefix

strs1 = ["flower", "flow", "flight"]

print(longestCommonPrefix(strs1))

strs2 = ["dog", "racecar", "car"]

print(longestCommonPrefix(strs2))

15.3 sum:

def threeSum(nums):

nums.sort()

res = []

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

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

continue

l, r = i+1, len(nums)-1

while l < r:

total = nums[i] + nums[l] + nums[r]

if total < 0:

l += 1

elif total > 0:

r -= 1

else:

res.append([nums[i], nums[l], nums[r]])

while l < r and nums[l] == nums[l+1]:

l += 1

while l < r and nums[r] == nums[r-1]:

r -= 1

l += 1

r -= 1

return res

nums1 = [-1, 0, 1, 2, -1, -4]

print(threeSum(nums1))

nums2 = [0, 1, 1]

print(threeSum(nums2))

16.3 sum closest:

def threeSumClosest(nums, target):

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

closest\_sum = current\_sum

if current\_sum < target:

left += 1

else:

right -= 1

return closest\_sum

nums = [-1, 2, 1, -4]

target = 1

output = threeSumClosest(nums, target)

print(output)

17.Letter combination of a number:

from itertools import product

def letterCombinations(digits):

if not digits:

return []

phone = {

'2': 'abc',

'3': 'def',

'4': 'ghi',

'5': 'jkl',

'6': 'mno',

'7': 'pqrs',

'8': 'tuv',

'9': 'wxyz'

}

return [''.join(p) for p in product(\*(phone[d] for d in digits))]

digits = "23"

output = letterCombinations(digits)

print(output)

18.4 sum:

def fourSum(nums, target):

nums.sort()

results = []

findNsum(nums, target, 4, [], results)

return results

def findNsum(nums, target, N, result, results):

if len(nums) < N or N < 2 or target < nums[0] \* N or target > nums[-1] \* N:

return

if N == 2:

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

while left < right:

s = nums[left] + nums[right]

if s == target:

results.append(result + [nums[left], nums[right]])

left += 1

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

left += 1

elif s < target:

left += 1

else:

right -= 1

else:

for i in range(len(nums) - N + 1):

if i == 0 or (i > 0 and nums[i] != nums[i - 1]):

findNsum(nums[i + 1:], target - nums[i], N - 1, result + [nums[i]], results)

nums = [1, 0, -1, 0, -2, 2]

target = 0

print(fourSum(nums, target))

19.Remove nth node from a list:

class ListNode:

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

self.val = val

self.next = next

def removeNthFromEnd(head, n):

dummy = ListNode(0)

dummy.next = head

first = dummy

second = dummy

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

first = first.next

while first is not None:

first = first.next

second = second.next

second.next = second.next.next

return dummy.next

20.valid parenthes:

def isValid(s):

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

s = "()"

print(isValid(s))