1.TWO SUMS

class Solution:

def twoSum(self, nums, target):

seen = {}

for i, num in enumerate(nums):

diff = target - num

if diff in seen:

return [seen[diff], i]

seen[num] = i

2.ADD TWO NUMBERS

class Solution:

def addTwoNumbers(self, l1: Optional[ListNode], l2: Optional[ListNode]) -> Optional[ListNode]:

dummy = ListNode()

res = dummy

total = carry = 0

while l1 or l2 or carry:

total = carry

if l1:

total += l1.val

l1 = l1.next

if l2:

total += l2.val

l2 = l2.next

num = total % 10

carry = total // 10

dummy.next = ListNode(num)

dummy = dummy.next

return res.next

[**3. Longest Substring Without Repeating Characters**](https://leetcode.com/problems/longest-substring-without-repeating-characters/)

class Solution:

def lengthOfLongestSubstring(self, s: str) -> int:

left = max\_length = 0

char\_set = set()

for right in range(len(s)):

while s[right] in char\_set:

char\_set.remove(s[left])

left += 1

char\_set.add(s[right])

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

return max\_length

[**4. Median of Two Sorted Arrays**](https://leetcode.com/problems/median-of-two-sorted-arrays/)

class Solution(object):

def findMedianSortedArrays(self, nums1, nums2):

"""

:type nums1: List[int]

:type nums2: List[int]

:rtype: float

"""

nums = sorted(nums1 + nums2)

length = len(nums)

mid = length // 2

return float(nums[mid] if length % 2 else (nums[mid-1] + nums[mid]) / 2.0)

[**5. Longest Palindromic Substring**](https://leetcode.com/problems/longest-palindromic-substring/)

class Solution(object):

def longestPalindrome(self, s):

"""

:type s: str

:rtype: str

"""

n = len(s)

if n == 0:

return ""

start = 0

max\_length = 1

dp = [[False] \* n for \_ in range(n)]

for i in range(n):

dp[i][i] = True # Single characters are palindromes

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

for i in range(n - length + 1):

j = i + length - 1

if s[i] == s[j]:

if length == 2:

dp[i][j] = True

else:

dp[i][j] = dp[i + 1][j - 1]

if dp[i][j] and length > max\_length:

start = i

max\_length = length

return s[start:start + max\_length]

[**6. Zigzag Conversion**](https://leetcode.com/problems/zigzag-conversion/)

class Solution:

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

if numRows == 1: return s

a=""

for i in range(numRows):

for j in range(i,len(s),2\*(numRows-1)):

a+=s[j]

if(i>0 and i<numRows-1 and j+2\*(numRows-1)-2\*i < len(s)):

a+=s[j+2\*(numRows-1)-2\*i]

return a

[**7. Reverse Integer**](https://leetcode.com/problems/reverse-integer/)

class Solution:

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

res = 0

if x < 0:

res = int(str(x)[1:][::-1]) \* -1

else:

res = int(str(x)[::-1])

if res > 2 \*\* 31 - 1 or res < -2 \*\* 31:

return 0

return res

[**8. String to Integer (atoi)**](https://leetcode.com/problems/string-to-integer-atoi/)

class Solution:

def myAtoi(self, s: str) -> int:

if not s:

return 0

# Constants for 32-bit signed integer range

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

INT\_MIN = -2\*\*31

i = 0

n = len(s)

# Step 1: Skip leading whitespace

while i < n and s[i] == ' ':

i += 1

# Check if we've reached the end

if i == n:

return 0

# Step 2: Check for sign

sign = 1

if s[i] == '+':

i += 1

elif s[i] == '-':

sign = -1

i += 1

# Step 3: Read digits and convert

res = 0

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

digit = int(s[i])

res = res \* 10 + digit

if sign \* res <= INT\_MIN:

return INT\_MIN

if sign \* res >= INT\_MAX:

return INT\_MAX

i += 1

# Step 4: Apply sign and return

return res \* sign

[**9. Palindrome Number**](https://leetcode.com/problems/palindrome-number/)

class Solution:

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

if x < 0:

return False

rev = 0

num = x

while num != 0:

rev = rev \* 10 + num % 10

num = num // 10

return rev == x

[**10. Regular Expression Matching**](https://leetcode.com/problems/regular-expression-matching/)

class Solution:

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

i, j = len(s) - 1, len(p) - 1

return self.backtrack({}, s, p, i, j)

def backtrack(self, cache, s, p, i, j):

key = (i, j)

if key in cache:

return cache[key]

if i == -1 and j == -1:

cache[key] = True

return True

if i != -1 and j == -1:

cache[key] = False

return cache[key]

if i == -1 and p[j] == '\*':

k = j

while k != -1 and p[k] == '\*':

k -= 2

if k == -1:

cache[key] = True

return cache[key]

cache[key] = False

return cache[key]

if i == -1 and p[j] != '\*':

cache[key] = False

return cache[key]

if p[j] == '\*':

if self.backtrack(cache, s, p, i, j - 2):

cache[key] = True

return cache[key]

if p[j - 1] == s[i] or p[j - 1] == '.':

if self.backtrack(cache, s, p, i - 1, j):

cache[key] = True

return cache[key]

if p[j] == '.' or s[i] == p[j]:

if self.backtrack(cache, s, p, i - 1, j - 1):

cache[key] = True

return cache[key]

cache[key] = False

return cache[key]