```python
# You are given two integer arrays nums1 and nums2, sorted in non-decreasing order, and two integers m and n, representing the number of elements in nums1 and nums2 respectively.

# Merge nums1 and nums2 into a single array sorted in non-decreasing order.

# The final sorted array should not be returned by the function, but instead be stored inside the array nums1.
# To accommodate this, nums1 has a length of m + n, where the first m elements denote the elements that should be merged,
# and the last n elements are set to 0 and should be ignored. nums2 has a length of n.



# Example 1:

# Input: nums1 = [1,2,3,0,0,0], m = 3, nums2 = [2,5,6], n = 3
# Output: [1,2,2,3,5,6]
# Explanation: The arrays we are merging are [1,2,3] and [2,5,6].
# The result of the merge is [1,2,2,3,5,6] with the underlined elements coming from nums1.
# Example 2:

# Input: nums1 = [1], m = 1, nums2 = [], n = 0
# Output: [1]
# Explanation: The arrays we are merging are [1] and [].
# The result of the merge is [1].
# Example 3:

# Input: nums1 = [0], m = 0, nums2 = [1], n = 1
# Output: [1]
# Explanation: The arrays we are merging are [] and [1].
# The result of the merge is [1].
# Note that because m = 0, there are no elements in nums1. The 0 is only there to ensure the merge result can fit in nums1.

#leetcode 88 : two pointers technique
class Solution:
    def merge(self, nums1: List[int], m: int, nums2: List[int], n: int) -> None:
        """
        Do not return anything, modify nums1 in-place instead.
        """


        last = (m + n) - 1

        while m > 0 and n > 0:
```

```
            if nums1[m-1] > nums2[n-1]:
                nums1[last] = nums1[m-1]
                m-=1
            else:
                nums1[last] = nums2[n-1]
                n-=1
            last-=1

        while n > 0:
            nums1[last] = nums2[n-1]

            last, n = last -1, n-1


        # for ele in range(m, len(nums1)):
        #     nums1.pop()
        # nums1 += nums2
        # nums1.sort()

        # for i in range(m, len(nums1)):

        #     nums1.remove(0)
        # nums1 += nums2
        # nums1.sort()


# 929. Unique Email Addresses
# Solved
# Easy
# Topics
# Companies
# Every valid email consists of a local name and a domain name,
separated by the '@' sign.
# Besides lowercase letters, the email may contain one or more '.' or
'+'.

# For example, in "alice@leetcode.com", "alice" is the local name, and
"leetcode.com" is the domain name.
# If you add periods '.' between some characters in the local name
part of an email address,
# mail sent there will be forwarded to the same address without dots
in the local name. Note that this rule does not apply to domain names.

# For example, "alice.z@leetcode.com" and "alicez@leetcode.com"
forward to the same email address.
# If you add a plus '+' in the local name, everything after the first
plus sign will be ignored.
# This allows certain emails to be filtered. Note that this rule does
not apply to domain names.
```

```
# For example, "m.y+name@email.com" will be forwarded to
"my@email.com".
# It is possible to use both of these rules at the same time.

# Given an array of strings emails where we send one email to each
emails[i],
# return the number of different addresses that actually receive
mails.



# Example 1:

# Input: emails =
["test.email+alex@leetcode.com","test.e.mail+bob.cathy@leetcode.com","
testemail+david@lee.tcode.com"]
# Output: 2
# Explanation: "testemail@leetcode.com" and "testemail@lee.tcode.com"
actually receive mails.
# Example 2:

# Input: emails = ["a@leetcode.com","b@leetcode.com","c@leetcode.com"]
# Output: 3
class Solution:
    def numUniqueEmails(self, emails: List[str]) -> int:

        mails = set()

        for email in emails:
            addr, domain = email.split('@')
            actual_addr = ""
            for a in addr:
                if a == ".":
                    continue
                elif a =="+":
                    break
                else:
                    actual_addr +=a
            actual_email = actual_addr + "@"+ domain
            mails.add(actual_email)

        return len(mails)

# Input: nums = [3,2,2,3], val = 3
# Output: 2, nums = [2,2,_,_]
# Explanation: Your function should return k = 2, with the first two
elements of nums being 2.
# It does not matter what you leave beyond the returned k (hence they
are underscores).
# Example 2:
```

```python
# Input: nums = [0,1,2,2,3,0,4,2], val = 2
# Output: 5, nums = [0,1,4,0,3,_,_,_]
# Explanation: Your function should return k = 5, with the first five elements of nums containing 0, 0, 1, 3, and 4.
# Note that the five elements can be returned in any order.
# It does not matter what you leave beyond the returned k (hence they are underscores).

#leetcode 27: two pointer technique
class Solution:
    def removeElement(self, nums: List[int], val: int) -> int:

        k = 0

        for i in range(len(nums)):
            if nums[i] == val:
                continue
            else:
                nums[k] = nums[i]
                k+=1


        return k


# Input: nums = [0,0,1,1,1,2,2,3,3,4]
# Output: 5, nums = [0,1,2,3,4,_,_,_,_,_]
# Explanation: Your function should return k = 5, with the first five elements of nums being 0, 1, 2, 3, and 4 respectively.
# It does not matter what you leave beyond the returned k (hence they are underscores).

#leetcode 26: two pointer technique
class Solution:
    def removeDuplicates(self, nums: List[int]) -> int:


        k = 0
        for i in range(len(nums)):
            if nums[k] == nums[i]:
                continue
            else:
                k = k +1
                nums[k] = nums[i]
        return k+1

# 849. Maximize Distance to Closest Person
# Solved
# Medium
# Topics
```

```python
# Companies
# You are given an array representing a row of seats where seats[i] =
1
# represents a person sitting in the ith seat, and seats[i] = 0
represents that the ith seat is empty (0-indexed).

# There is at least one empty seat, and at least one person sitting.

# Alex wants to sit in the seat such that the distance between him and
the closest person to him is maximized.

# Return that maximum distance to the closest person.



# Example 1:


# Input: seats = [1,0,0,0,1,0,1]
# Output: 2
# Explanation:
# If Alex sits in the second open seat (i.e. seats[2]), then the
closest person has distance 2.
# If Alex sits in any other open seat, the closest person has distance
1.
# Thus, the maximum distance to the closest person is 2.
# Example 2:

# Input: seats = [1,0,0,0]
# Output: 3
# Explanation:
# If Alex sits in the last seat (i.e. seats[3]), the closest person is
3 seats away.
# This is the maximum distance possible, so the answer is 3.
# Example 3:

# Input: seats = [0,1]
# Output: 1


class Solution:
    def maxDistToClosest(self, seats: List[int]) -> int:

        #[1,0,0,0,1]--- case 1
        #the formula for calculating max empty spaces
        #is the number of empty spaces between divided by 2
        #ceil (empty spaces / 2)

        #[0,0,0,1] special case
        #lots of zero at the beginning
        #formula = (2 * empty space ) / 2
```

```
        res = 0
        dist = seats.index(1)

        for seat in seats:
            if seat == 1:
                res, dist = max(res, math.ceil(dist/2)), 0
            else:
                dist +=1
        return max(res, dist)



class Solution:
    def maxDistToClosest(self, seats: List[int]) -> int:
        res, last = 0, -1
        for i, seat in enumerate(seats):
            if seat:
                    # `else i` takes care of the edge case when we
do not have a seat at first index
                res = max(res, (i - last) // 2 if last >= 0 else i)
                last = i
           # `len(seats) - 1 - last` takes care of the edge case when
we do not have a seat at last index
        return max(res, len(seats) - 1 - last if not seats[-1] else 0)

# Example 1:

# Input: nums = [3,2,3]
# Output: 3
# Example 2:

# Input: nums = [2,2,1,1,1,2,2]
# Output: 2

#leetcode 169
class Solution:
    #used divide and conquer
    # space : O(lg n)
    #time : O(nlgn)
    def majorityElement(self, nums: List[int]) -> int:

        def majority_2(n: List[int], start: int, end: int) -> int:

            if start == end:
                return n[start]

            mid = (start + end) // 2

            left = majority(n, start, mid)
            right = majority(n, mid+1, end)
```

```
            count_of_left = sum(1 for i in range(start, end+1) if n[i]
== left)
            count_of_right = sum(1 for i in range(start, end+1) if
n[i] == right)

            return left if count_of_left > count_of_right else right

        return majority(nums, 0, len(nums)-1)

        majority = nums[0]
        count = 1

  #space O(1)
  #time O(N)
  def majorityElement_1(self, nums: List[int]) -> int:
        majority = nums[0]
        count = 1

        for i in range(1, len(nums)):

            if nums[i] == majority:
                count +=1
            elif count == 0:
                majority = nums[i]
                count = 1
            elif nums[i] != majority:
                count -=1
            else:
                continue

        return majority

  #used an hashtable
  space(O(n))
  def majorityElement_3(self, nums: List[int]) -> int:

        nums_count = {}


        for i in range(len(nums)):

            if nums[i] in nums_count:
                nums_count[nums[i]] +=1
            else:
                nums_count[nums[i]] = 1

        majority = nums[0]
        for num, freq in nums_count.items():
            if freq > nums_count[majority]:
                majority = num
```

```
        return majority


# 121. Best Time to Buy and Sell Stock
# Solved
# Easy
# Topics
# Companies
# You are given an array prices where prices[i] is the price of a
given stock on the ith day.

# You want to maximize your profit by choosing a single day to buy one
stock and choosing a different day in the future to sell that stock.

# Return the maximum profit you can achieve from this transaction. If
you cannot achieve any profit, return 0.



# Example 1:

# Input: prices = [7,1,5,3,6,4]
# Output: 5
# Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6),
profit = 6-1 = 5.
# Note that buying on day 2 and selling on day 1 is not allowed
because you must buy before you sell.
# Example 2:

# Input: prices = [7,6,4,3,1]
# Output: 0
# Explanation: In this case, no transactions are done and the max
profit = 0.

class Solution:
    def maxProfit(self, prices: List[int]) -> int:

        #find the minimum prices and keep it
        min_price = float('inf')
        profit = 0

        for i in range(len(prices)):

            min_price = min(min_price, prices[i])
            profit = max(profit, prices[i] - min_price)

        return profit


#this is O(N^2)
```

```python
  def maxProfit_2(self, prices: List[int]) -> int:
        profit = 0

        for i in range(len(prices)):
            for j in range(i+1, len(prices)):
                profit = max(profit, prices[j] - prices[i])

        return profit if profit > 0 else 0


# 13. Roman to Integer
# Solved
# Easy
# Topics
# Companies
# Hint
# Roman numerals are represented by seven different symbols: I, V, X,
L, C, D and M.

# Symbol       Value
# I             1
# V             5
# X             10
# L             50
# C             100
# D             500
# M             1000
# For example, 2 is written as II in Roman numeral, just two ones
added together. 12 is written as XII, which is simply X + II. The
number 27 is written as XXVII, which is XX + V + II.

# Roman numerals are usually written largest to smallest from left to
right. However, t
# he numeral for four is not IIII. Instead, the number four is written
as IV. Because the one is before the five we subtract it making four.
# The same principle applies to the number nine, which is written as
IX. There are six instances where subtraction is used:

# I can be placed before V (5) and X (10) to make 4 and 9.
# X can be placed before L (50) and C (100) to make 40 and 90.
# C can be placed before D (500) and M (1000) to make 400 and 900.
# Given a roman numeral, convert it to an integer.



# Example 1:

# Input: s = "III"
# Output: 3
# Explanation: III = 3.
```

```python
# Example 2:

# Input: s = "LVIII"
# Output: 58
# Explanation: L = 50, V= 5, III = 3.
# Example 3:

# Input: s = "MCMXCIV"
# Output: 1994
# Explanation: M = 1000, CM = 900, XC = 90 and IV = 4.

class Solution:
    def romanToInt(self, s: str) -> int:
        rom = s.lower()
        symbol_value = {
            "i": 1,
            "v": 5,
            "x": 10,
            "l": 50,
            "c": 100,
            "d": 500,
            "m": 1000,
            "iv": 4,
            "ix": 9,
            "xl": 40,
            "xc": 90,
            "cd": 400,
            "cm": 900
        }

        numeral = 0
        i = 0
        while i < len(rom):

            if i+1 < len(rom) and rom[i] + rom[i+1] in symbol_value:
                numeral += symbol_value[rom[i] + rom[i+1]]
                i+=2
            else:
                numeral += symbol_value[rom[i]]
                i+=1
        return numeral

# 657. Robot Return to Origin
# Solved
# Easy
# Topics
# Companies
# There is a robot starting at the position (0, 0), the origin, on a
2D plane. Given a sequence of its moves,
# judge if this robot ends up at (0, 0) after it completes its moves.
```

```python
# You are given a string moves that represents the move sequence of the robot where moves[i]
# represents its ith move. Valid moves are 'R' (right), 'L' (left), 'U' (up), and 'D' (down).

# Return true if the robot returns to the origin after it finishes all of its moves, or false otherwise.

# Note: The way that the robot is "facing" is irrelevant. 'R' will always make the robot move to the right once,
# 'L' will always make it move left, etc. Also, assume that the magnitude of the robot's movement is the same for each move.



# Example 1:

# Input: moves = "UD"
# Output: true
# Explanation: The robot moves up once, and then down once. All moves have the same magnitude,
# so it ended up at the origin where it started. Therefore, we return true.
# Example 2:

# Input: moves = "LL"
# Output: false
# Explanation: The robot moves left twice. It ends up two "moves" to the left of the
# origin. We return false because it is not at the origin at the end of its moves.

class Solution:
    def judgeCircle(self, moves: str) -> bool:

        origin = [0,0]

        for move in moves:
            if move == "U":
                origin[1]+=1
            elif move == "D":
                origin[1]-=1
            elif move == "R":
                origin[0]+=1
            elif move == "L":
                origin[0]-=1

        return origin[0] == 0 and origin[1] == 0
```

```python
class Solution:
    def judgeCircle(self, moves: str) -> bool:
        return moves.count('L') == moves.count('R') and
moves.count('U') == moves.count('D')


# 58. Length of Last Word
# Solved
# Easy
# Topics
# Companies
# Given a string s consisting of words and spaces, return the length
of the last word in the string.

# A word is a maximal
# substring
#  consisting of non-space characters only.



# Example 1:

# Input: s = "Hello World"
# Output: 5
# Explanation: The last word is "World" with length 5.
# Example 2:

# Input: s = "   fly me   to   the moon  "
# Output: 4
# Explanation: The last word is "moon" with length 4.
# Example 3:

# Input: s = "luffy is still joyboy"
# Output: 6
# Explanation: The last word is "joyboy" with length 6.

class Solution:
    def lengthOfLastWord(self, s: str) -> int:

        ln = 0
        found_space = False
        for i in range(len(s)):

            if s[i] == " " and i < len(s):
                found_space = True
            elif found_space == True and s[i] != " ":
                ln =1
                found_space = False
            else:
                ln+=1
        return ln
```

```python
    def lengthOfLastWord_2(self, s: str) -> int:

        words = s.split()
        last_word = words[-1]
        return len(last_word)

# 14. Longest Common Prefix
# Solved
# Easy
# Topics
# Companies
# Write a function to find the longest common prefix string amongst an
array of strings.

# If there is no common prefix, return an empty string "".



# Example 1:

# Input: strs = ["flower","flow","flight"]
# Output: "fl"
# Example 2:

# Input: strs = ["dog","racecar","car"]
# Output: ""
# Explanation: There is no common prefix among the input strings.

class Solution:
    def longestCommonPrefix(self, strs: List[str]) -> str:
        #pick a string in the list
        word = strs[0]

        prefix = ""

        for i in range(len(word)):

            for j in range(len(strs)):

                if i == len(strs[j]) or word[i] != strs[j][i]:
                    return prefix
            prefix += word[i]

        return prefix

   def longestCommonPrefix(self, strs: List[str]) -> str:

        #sort the array
        prefix = ""
```

```
        sortedStrings = sorted(strs)

        shortest_string = sortedStrings[0]
        longest_string = sortedStrings[-1]

        len_shortest = len(shortest_string)
        len_longest = len(longest_string)

        for i in range( len_shortest ):

            if shortest_string[i] != longest_string[i]:
                return prefix

            prefix += shortest_string[i]

        return prefix



# 28. Find the Index of the First Occurrence in a String
# Solved
# Easy
# Topics
# Companies
# Given two strings needle and haystack, return the index of the first
occurrence of needle in haystack, or -1 if needle is not part of
haystack.



# Example 1:

# Input: haystack = "sadbutsad", needle = "sad"
# Output: 0
# Explanation: "sad" occurs at index 0 and 6.
# The first occurrence is at index 0, so we return 0.
# Example 2:

# Input: haystack = "leetcode", needle = "leeto"
# Output: -1
# Explanation: "leeto" did not occur in "leetcode", so we return -1.

class Solution:
    def checkSubHaystack(self, subStack: str, needle: str, start: int,
end: int) -> bool:
        needle_index = 0
        for i in range(start, end):

            if needle[needle_index] != subStack[i]:
                return False
            needle_index +=1
```

```
        return True


    def strStr(self, haystack: str, needle: str) -> int:

        if len(needle) > len(haystack):
            return -1

        for i in range(len(haystack) - len(needle)+1):
            if self.checkSubHaystack(haystack, needle, i, i+
len(needle) ):
                return i

        return -1


# 125. Valid Palindrome
# Solved
# Easy
# Topics
# Companies
# A phrase is a palindrome if, after converting all uppercase letters
into lowercase letters and
# removing all non-alphanumeric characters, it reads the same forward
and backward. Alphanumeric characters include letters and numbers.

# Given a string s, return true if it is a palindrome, or false
otherwise.



# Example 1:

# Input: s = "A man, a plan, a canal: Panama"
# Output: true
# Explanation: "amanaplanacanalpanama" is a palindrome.
# Example 2:

# Input: s = "race a car"
# Output: false
# Explanation: "raceacar" is not a palindrome.
# Example 3:

# Input: s = " "
# Output: true
# Explanation: s is an empty string "" after removing non-alphanumeric
characters.
# Since an empty string reads the same forward and backward, it is a
palindrome.

class Solution:
```

```
    def checkIfPalindrome(self, words: str, start: int, end: int)->
bool:

        while start < end:

            if words[start] != words[end]:
                return False
            start+=1
            end-=1
        return True



    def isPalindrome(self, s: str) -> bool:

        if s == " ":
            return True
        s = s.lower()

        alpha_num_char = ""

        for i in range(len(s)):
            if s[i] != " " and s[i].isalnum():
                alpha_num_char += s[i]


        if self.checkIfPalindrome(alpha_num_char, 0,
len(alpha_num_char)-1):
            return True
        return False

class Solution:
    def isPalindrome(self, s: str) -> bool:
        l, r = 0, len(s) - 1

        while l < r:
            while l < r and not self.alphaNum(s[l]):
                l += 1
            while r > l and not self.alphaNum(s[r]):
                r -= 1
            if s[l].lower() != s[r].lower():
                return False
            l, r = l + 1, r - 1
        return True

    def alphaNum(self, c):
        return (ord('A') <= ord(c) <= ord('Z') or
                ord('a') <= ord(c) <= ord('z') or
                ord('0') <= ord(c) <= ord('9'))
```

```python
# 392. Is Subsequence
# Solved
# Easy
# Topics
# Companies
# Given two strings s and t, return true if s is a subsequence of t,
or false otherwise.

# A subsequence of a string is a new string that is formed from the
original string by deleting some (can be none) of the characters
without disturbing
# the relative positions of the remaining characters. (i.e., "ace" is
a subsequence of "abcde" while "aec" is not).



# Example 1:

# Input: s = "abc", t = "ahbgdc"
# Output: true
# Example 2:

# Input: s = "axc", t = "ahbgdc"
# Output: false


# Constraints:

# 0 <= s.length <= 100
# 0 <= t.length <= 104
# s and t consist only of lowercase English letters.

class Solution:
    def isSubsequence(self, s: str, t: str) -> bool:

        s_len = 0
        t_len = 0

        while s_len < len(s) and t_len < len(t):

            if s[s_len] == t[t_len]:

                s_len +=1
                t_len +=1
            else:
                t_len +=1


        return s_len == len(s)
```

```python
# 383. Ransom Note
# Solved
# Easy
# Topics
# Companies
# Given two strings ransomNote and magazine, return true if ransomNote
can be constructed by using the letters from magazine and false
otherwise.

# Each letter in magazine can only be used once in ransomNote.



# Example 1:

# Input: ransomNote = "a", magazine = "b"
# Output: false
# Example 2:

# Input: ransomNote = "aa", magazine = "ab"
# Output: false
# Example 3:

# Input: ransomNote = "aa", magazine = "aab"
# Output: true

class Solution:
    def canConstruct(self, ransomNote: str, magazine: str) -> bool:

        look_up = {}
        for mag in magazine:
            if mag not in look_up:
                look_up[mag] = 1
            else:
                look_up[mag] += 1

        for note in ransomNote:
            if note not in look_up or look_up[note]  == 0:
                return False

            look_up[note] -= 1
        return True


# 205. Isomorphic Strings
# Solved
# Easy
# Topics
# Companies
# Given two strings s and t, determine if they are isomorphic.
```

```python
# Two strings s and t are isomorphic if the characters in s can be replaced to get t.

# All occurrences of a character must be replaced with another character while preserving the order of characters. No two characters may map to the same character, but a character may map to itself.



# Example 1:

# Input: s = "egg", t = "add"
# Output: true
# Example 2:

# Input: s = "foo", t = "bar"
# Output: false
# Example 3:

# Input: s = "paper", t = "title"
# Output: true

class Solution:
    def isIsomorphic(self, s: str, t: str) -> bool:

        if len(s) != len(t):
            return False

        # Dictionary to store the mapping of characters from s to t
        iso_dict_s_to_t = {}
        # Dictionary to store the mapping of characters from t to s
        iso_dict_t_to_s = {}

        for char_s, char_t in zip(s, t):
            if char_s in iso_dict_s_to_t:
                if iso_dict_s_to_t[char_s] != char_t:
                    return False
            else:
                iso_dict_s_to_t[char_s] = char_t

            if char_t in iso_dict_t_to_s:
                if iso_dict_t_to_s[char_t] != char_s:
                    return False
            else:
                iso_dict_t_to_s[char_t] = char_s

        return True

# 290. Word Pattern
# Solved
# Easy
```

```python
# Topics
# Companies
# Given a pattern and a string s, find if s follows the same pattern.

# Here follow means a full match, such that there is a bijection
between a letter in pattern and a non-empty word in s.



# Example 1:

# Input: pattern = "abba", s = "dog cat cat dog"
# Output: true
# Example 2:

# Input: pattern = "abba", s = "dog cat cat fish"
# Output: false
# Example 3:

# Input: pattern = "aaaa", s = "dog cat cat dog"
# Output: false

class Solution:
    def wordPattern(self, pattern: str, s: str) -> bool:

        pat_dict = {}
        s_dict = {}
        stringify = s.split()
        p = []



        for char in pattern:
            p.append(char)

        if len(p) != len(stringify):
            return False

        for p, wd in zip(pattern, stringify):

            if p in pat_dict:
                if pat_dict[p] != wd:
                    return False
            else:
                pat_dict[p] = wd

            if wd in s_dict:
                if s_dict[wd] != p:
                    return False
            else:
                s_dict[wd] = p
```

```
        return True


# 242. Valid Anagram
# Solved
# Easy
# Topics
# Companies
# Given two strings s and t, return true if t is an anagram of s, and
false otherwise.

# An Anagram is a word or phrase formed by rearranging the letters of
a different word or phrase, typically using all the original letters
exactly once.



# Example 1:

# Input: s = "anagram", t = "nagaram"
# Output: true
# Example 2:

# Input: s = "rat", t = "car"
# Output: false


class Solution:
    def isAnagram(self, s: str, t: str) -> bool:

        if len(s) != len(t):
            return False

        anagram = [0] * 256

        for char in s:

            anagram[ord(char)] +=1


        for char in t:

            if anagram[ord(char)] == 0:
                return False

            anagram[ord(char)] -=1

        return True
```

```python
# Code
# Testcase
# Testcase
# Test Result
# 1. Two Sum
# Solved
# Easy
# Topics
# Companies
# Hint
# Given an array of integers nums and an integer target, return
indices of the two numbers such that they add up to target.

# You may assume that each input would have exactly one solution, and
you may not use the same element twice.

# You can return the answer in any order.



# Example 1:

# Input: nums = [2,7,11,15], target = 9
# Output: [0,1]
# Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].
# Example 2:

# Input: nums = [3,2,4], target = 6
# Output: [1,2]
# Example 3:

# Input: nums = [3,3], target = 6
# Output: [0,1]

class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:

        numbers_and_index = {}
        ans = []
        for i in range(len(nums)):

            if target - nums[i] in numbers_and_index:

                return [numbers_and_index[target - nums[i]], i]
            else:
                numbers_and_index[nums[i]] = i


        return ans
```

```python
# 202. Happy Number
# Solved
# Easy
# Topics
# Companies
# Write an algorithm to determine if a number n is happy.

# A happy number is a number defined by the following process:

# Starting with any positive integer, replace the number by the sum of the squares of its digits.
# Repeat the process until the number equals 1 (where it will stay), or it loops endlessly in a cycle which does not include 1.
# Those numbers for which this process ends in 1 are happy.
# Return true if n is a happy number, and false if not.



# Example 1:

# Input: n = 19
# Output: true
# Explanation:
# 12 + 92 = 82
# 82 + 22 = 68
# 62 + 82 = 100
# 12 + 02 + 02 = 1
# Example 2:

# Input: n = 2
# Output: false

class Solution:
    def isHappy(self, n: int) -> bool:

        seen_numbers = set()

        current = n

        while current not in seen_numbers:

            seen_numbers.add(current)
            cummulative = 0

            while current > 0:
                cummulative += (current % 10)**2

                current = current // 10
```

```
            current = cummulative

            if current == 1:
                return True

        return False


# 219. Contains Duplicate II
# Solved
# Easy
# Topics
# Companies
# Given an integer array nums and an integer k, return true if there
are two distinct indices i and j in the array such that nums[i] ==
nums[j] and abs(i - j) <= k.



# Example 1:

# Input: nums = [1,2,3,1], k = 3
# Output: true
# Example 2:

# Input: nums = [1,0,1,1], k = 1
# Output: true
# Example 3:

# Input: nums = [1,2,3,1,2,3], k = 2
# Output: false

class Solution:
    def containsNearbyDuplicate(self, nums: List[int], k: int) ->
bool:


        seen_numbers_and_index = {}
        for i in range(len(nums)):

            val = nums[i]

            if val in seen_numbers_and_index:

                index_of_seen = seen_numbers_and_index[val]

                if abs(index_of_seen - i) <= k:
                    return True

            seen_numbers_and_index[val] = i
```

```
        return False

# 217. Contains Duplicate
# Solved
# Easy
# Topics
# Companies
# Given an integer array nums, return true if any value appears at
least twice in the array, and return false if every element is
distinct.



# Example 1:

# Input: nums = [1,2,3,1]
# Output: true
# Example 2:

# Input: nums = [1,2,3,4]
# Output: false
# Example 3:

# Input: nums = [1,1,1,3,3,4,3,2,4,2]
# Output: true
class Solution:
    def containsDuplicate(self, nums: List[int]) -> bool:

        dup = set()

        for num in nums:

            if num in dup:
                return True
            else:
                dup.add(num)

        return False
class Solution:
    def containsDuplicate(self, nums: List[int]) -> bool:

       s = set(nums)

       return True if len(s) < len(nums) else False

class Solution:
    def containsDuplicate(self, nums: List[int]) -> bool:

        def quickSort(start, end):
```

```
            if start < end:
                p, pivot = start, nums[end]

                for i in range(start, end):
                    if nums[i] < pivot:
                        nums[p], nums[i] = nums[i], nums[p]
                        p+=1

                nums[p], nums[end], = nums[end], nums[p]

                quickSort(start, p-1)
                quickSort(p+1, end)

        s, end = 0, len(nums)-1

        quickSort(s, end)

        for i in range(1, len(nums)):

            if nums[i-1] == nums[i]:
                return True


        return False




class Solution:
    def containsDuplicate(self, nums: List[int]) -> bool:

        for i in range(len(nums)-1):
            for j in range(i+1, len(nums)):
                if nums[i] == nums[j]:
                    return True


        return False

# 228. Summary Ranges
# Solved
# Easy
# Topics
# Companies
# You are given a sorted unique integer array nums.

# A range [a,b] is the set of all integers from a to b (inclusive).

# Return the smallest sorted list of ranges that cover all the numbers
in the array exactly. That is,
```

```
# each element of nums is covered by exactly one of the ranges, and
there is no integer x such that x is in one of the ranges but not in
nums.

# Each range [a,b] in the list should be output as:

# "a->b" if a != b
# "a" if a == b


# Example 1:

# Input: nums = [0,1,2,4,5,7]
# Output: ["0->2","4->5","7"]
# Explanation: The ranges are:
# [0,2] --> "0->2"
# [4,5] --> "4->5"
# [7,7] --> "7"
# Example 2:

# Input: nums = [0,2,3,4,6,8,9]
# Output: ["0","2->4","6","8->9"]
# Explanation: The ranges are:
# [0,0] --> "0"
# [2,4] --> "2->4"
# [6,6] --> "6"
# [8,9] --> "8->9"

class Solution:
    def summaryRanges(self, nums: List[int]) -> List[str]:

        #handle empty buffer
        if not nums:
            return []
        #handle  buffer with one element
        if len(nums) == 1:
            return [str(nums[0])]


        ans = []
        left_pointer = 0
        right_pointer = 1

        while right_pointer < len(nums):

            #if the range is no longer continous
            #i.e that the difference is more than one
            #found a demarcation
            if nums[right_pointer] - nums[right_pointer - 1] > 1:
```

```
                range_1 = nums[left_pointer]
                range_2 = nums[right_pointer - 1]

                #first check if range will be same number
                #then one element
                if range_1 == range_2:
                    sub_range = str(range_1)
                    ans.append(sub_range)
                #otherwise
                #two element
                else:
                    sub_range = str(range_1) + "->" + str(range_2)
                    ans.append(sub_range)

                #advance left to current right pointerpointer
                left_pointer = right_pointer


            # advance right pointer
            right_pointer+=1

        #check if left pointer never left the last element
        #we include it as it owns range
        if left_pointer == len(nums) - 1:
            last_number = nums[len(nums) - 1]
            sub_range = str(last_number)
            ans.append(sub_range)

        #if last pointer is still far behind,
        #Then we include it and the last element
        #because the range is still continous
        if left_pointer < len(nums) - 1:
            left_pointer_number = nums[left_pointer]
            last_number = nums[len(nums) - 1]
            sub_range = str(left_pointer_number)+ "->"
+str(last_number)
            ans.append(sub_range)


        return ans


# 20. Valid Parentheses
# Solved
# Easy
# Topics
# Companies
# Hint
# Given a string s containing just the characters '(', ')', '{', '}',
'[' and ']', determine if the input string is valid.
```

```python
# An input string is valid if:

# Open brackets must be closed by the same type of brackets.
# Open brackets must be closed in the correct order.
# Every close bracket has a corresponding open bracket of the same
type.


# Example 1:

# Input: s = "()"
# Output: true
# Example 2:

# Input: s = "()[]{}"
# Output: true
# Example 3:

# Input: s = "(]"
# Output: false

class Solution:
    def isValid(self, s: str) -> bool:


        stk = []
        for ch in s:
            if ch == "(" or ch == "{" or ch == "[":
                stk.append(ch)
            elif ch == ")":
                if not stk or stk.pop() != "(":
                    return False
            elif ch == "}":
                if not stk or stk.pop() != "{":
                    return False
            elif ch == "]":
                if not stk or stk.pop() != "[":
                    return False
            else:
                return False

        #return true if stack is empty
        return not stk

# 141. Linked List Cycle
# Solved
# Easy
# Topics
# Companies
# Given head, the head of a linked list, determine if the linked list
```

```
has a cycle in it.

# There is a cycle in a linked list if there is some node in the list
that can be reached again
# by continuously following the next pointer. Internally, pos is used
to denote the index of the node that t
# ail's next pointer is connected to. Note that pos is not passed as a
parameter.

# Return true if there is a cycle in the linked list. Otherwise,
return false.



# Example 1:


# Input: head = [3,2,0,-4], pos = 1
# Output: true
# Explanation: There is a cycle in the linked list, where the tail
connects to the 1st node (0-indexed).
# Example 2:


# Input: head = [1,2], pos = 0
# Output: true
# Explanation: There is a cycle in the linked list, where the tail
connects to the 0th node.
# Example 3:


# Input: head = [1], pos = -1
# Output: false
# Explanation: There is no cycle in the linked list.

# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    #use a set
    def hasCycle(self, head: Optional[ListNode]) -> bool:

        seen = set()

        curr = head
        while curr:

            if curr in seen:
```

```
                return True
            else:
                seen.add(curr)

            curr = curr.next

        return False

    #use a two pointers: slow and fast
    def hasCycle(self, head: Optional[ListNode]) -> bool:


        slow = head
        fast = head

        while fast and fast.next:

            slow = slow.next
            fast = fast.next.next

            if slow == fast:
                return True

        return False

# 21. Merge Two Sorted Lists
# Solved
# Easy
# Topics
# Companies
# You are given the heads of two sorted linked lists list1 and list2.

# Merge the two lists into one sorted list. The list should be made by
splicing together the nodes of the first two lists.

# Return the head of the merged linked list.



# Example 1:


# Input: list1 = [1,2,4], list2 = [1,3,4]
# Output: [1,1,2,3,4,4]
# Example 2:

# Input: list1 = [], list2 = []
# Output: []
# Example 3:

# Input: list1 = [], list2 = [0]
```

```
# Output: [0]


# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:

    def mergeTwoLists(self, list1: Optional[ListNode], list2:
Optional[ListNode]) -> Optional[ListNode]:

        #use while loop
        head = ListNode()
        current = head

        while list1 and list2:

            if list1.val <= list2.val:
                current.next = list1
                current = current.next
                list1 = list1.next
            else:
                current.next = list2
                current = current.next
                list2 = list2.next


        #This is the case where one list has finished iterating and
the other is still there
        if list1 is not None:
            current.next = list1
            current = current.next
        if list2 is not None:
            current.next = list2
            current = current.next

        return head.next


    def mergeTwoLists(self, list1: Optional[ListNode], list2:
Optional[ListNode]) -> Optional[ListNode]:

      #use recursion
        head : ListNode = None

        if list1 is None:
            return list2
        if list2 is None:
```

```
            return list1

        # Choose the smaller value as the head and merge the remaining
lists
        if list1.val <= list2.val:
            head = list1
            head.next = self.mergeTwoLists(list1.next, list2)
        else:
            head = list2
            head.next = self.mergeTwoLists(list1, list2.next)

        return head


# 104. Maximum Depth of Binary Tree
# Solved
# Easy
# Topics
# Companies
# Given the root of a binary tree, return its maximum depth.

# A binary tree's maximum depth is the number of nodes along the
longest path from the root node down to the farthest leaf node.

class Solution:
    def maxDepth(self, root: Optional[TreeNode]) -> int:

        if root is None: return 0

        left = self.maxDepth(root.left) + 1
        right = self.maxDepth(root.right) + 1

        return left if left > right else right

# 100. Same Tree
# Solved
# Easy
# Topics
# Companies
# Given the roots of two binary trees p and q, write a function to
check if they are the same or not.

# Two binary trees are considered the same if they are structurally
identical, and the nodes have the same value.

class Solution:
    def isSameTree(self, p: Optional[TreeNode], q: Optional[TreeNode])
-> bool:

        if p is None and q is None:
```

```
            return True

        if (p is not None and q is None ) or (p is None and q is not
None ) or p.val != q.val:
            return False


        return self.isSameTree(p.left, q.left) and
self.isSameTree(p.right, q.right)


# Code
# Testcase
# Testcase
# Test Result
# 226. Invert Binary Tree
# Solved
# Easy
# Topics
# Companies
# Given the root of a binary tree, invert the tree, and return its
root.

# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def invertTree(self, root: Optional[TreeNode]) ->
Optional[TreeNode]:

        if root is None:
            return root


        queue = []

        queue.append(root)

        while len(queue) > 0:

            current = queue.pop(0)

            if current is None:
                continue

            left = current.left
            right = current.right
```

```
            current.right = left
            current.left = right

            if left is not None:
                queue.append(left)
            if right is not None:
                queue.append(right)

        return root

    def invertTree(self, root: Optional[TreeNode]) ->
Optional[TreeNode]:

        if root is None:
            return root

        left = self.invertTree(root.left)
        right = self.invertTree(root.right)

        root.left, root.right = right, left



        return root


# 101. Symmetric Tree
# Solved
# Easy
# Topics
# Companies
# Given the root of a binary tree, check whether it is a mirror of
itself (i.e., symmetric around its center).

# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def isSymmetric(self, root: Optional[TreeNode]) -> bool:

        if root is None:
            return True

        def dfs(left: Optional[TreeNode], right : Optional[TreeNode])
-> bool:

            #the case of a single node binary tree
```

```
            if not left and not right:
                return True
            #if either left or right is missing
            #cannot be symmetric

            if (not left and right) or (not right and left):
                return False

            return (left.val == right.val) and dfs(left.left,
right.right) and dfs(left.right, right.left)

        return dfs(root.left, root.right)



# 112. Path Sum
# Solved
# Easy
# Topics
# Companies
# Given the root of a binary tree and an integer targetSum, return
true if the tree has a root-to-leaf path such that adding up all the
values along the path equals targetSum.

# A leaf is a node with no children.

# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def hasPathSum(self, root: Optional[TreeNode], targetSum: int) ->
bool:

        if not root:
            return False

        #This makes sure we are the root
        if not root.left and not root.right:
            return targetSum == root.val

        #this does not ensure we are the root
        # if targetSum == root.val:
        #     return True

        return self.hasPathSum(root.left, targetSum - root.val) or
self.hasPathSum(root.right, targetSum - root.val)
```

```python
# Code
# Testcase
# Testcase
# Test Result
# 222. Count Complete Tree Nodes
# Solved
# Easy
# Topics
# Companies
# Given the root of a complete binary tree, return the number of the
nodes in the tree.

# According to Wikipedia, every level, except possibly the last, is
completely filled in a complete binary tree,
# and all nodes in the last level are as far left as possible. It can
have between 1 and 2h nodes inclusive at the last level h.

# Design an algorithm that runs in less than O(n) time complexity.

# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def countNodes(self, root: Optional[TreeNode]) -> int:
        if not root:
            return 0
        return self.countNodes(root.left) +
self.countNodes(root.right) + 1

    def countNodes(self, root: Optional[TreeNode]) -> int:

        ans = 0

        if not root:
            return ans

        queue = []

        current  = root
        queue.append(current)
        ans+=1

        while len(queue) > 0:

            left = None
            right = None
```

```
            current = queue.pop(0)

            left = current.left
            right = current.right

            if left:
                queue.append(left)
                ans+=1
            if right:
                queue.append(right)
                ans+=1
        return ans




# 637. Average of Levels in Binary Tree
# Solved
# Easy
# Topics
# Companies
# Given the root of a binary tree, return the average value of the
nodes on each level in the form of an array. Answers within 10-5 of
the actual answer will be accepted.

# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def averageOfLevels(self, root: Optional[TreeNode]) ->
List[float]:

        ans : List[float] = []

        if not root:
            return ans


        queue : TreeNode = []
        queue.append(root)

        while len(queue) > 0:

            average = 0.0

            size_of_queue = len(queue)
```

```
            for i in range(size_of_queue):

                current = queue.pop(0)

                average += current.val

                if current.left:
                    queue.append(current.left)
                if current.right:
                    queue.append(current.right)

            average = average / size_of_queue

            ans.append(average)



        return ans

# 530. Minimum Absolute Difference in BST
# Solved
# Easy
# Topics
# Companies
# Given the root of a Binary Search Tree (BST), return the minimum
absolute difference between the values of any two different nodes in
the tree.


# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def getMinimumDifference(self, root: Optional[TreeNode]) -> int:

        def getMinSortedDifference(nums: List[int]) -> int:

            min = 10**5

            for i in range(1, len(nums)):

                if nums[i] - nums[i-1] < min:
                    min = nums[i] - nums[i-1]

            return min
```

```
        def getNodesVal(root: TreeNode, vals: list[int]) -> List[int]:

            if not root:
                return

            getNodesVal(root.left, vals)
            vals.append(root.val)
            getNodesVal(root.right, vals)


        values = []
        getNodesVal(root, values)
        minDiff = getMinSortedDifference(values)

        return minDiff



    def getMinimumDifference(self, root: Optional[TreeNode]) -> int:

        #use inorder traversal

        def getMinDifferenceHelper(root: Optional[TreeNode], min:
int , seen: List[int]) -> int:

            if not root:
                return

            getMinDifferenceHelper(root.left, min, seen)

            #check if seen is not empty and compare latest value with
seen values to update minimum difference
            if len(seen) > 0:

                for i in range(len(seen)):

                    if abs( seen[i] - root.val ) < min[0]:
                        min.pop(0)
                        min.append( abs( seen[i] - root.val ) )
            seen.append(root.val)
            getMinDifferenceHelper(root.right, min, seen)


        min = [11**10]
        seen = []

        ans = getMinDifferenceHelper(root, min, seen)

        return min[0]
```

```python
# 67. Add Binary
# Solved
# Easy
# Topics
# Companies
# Given two binary strings a and b, return their sum as a binary
string.



# Example 1:

# Input: a = "11", b = "1"
# Output: "100"
# Example 2:

# Input: a = "1010", b = "1011"
# Output: "10101"

def addBinary(a: str, b: str) -> str:
    # Base case: if one of the strings is empty, return the other
    if not a:
        return b
    if not b:
        return a

    # If both strings have bits, process the least significant bit
(last bit)
    if a[-1] == '1' and b[-1] == '1':
        # Both bits are 1, so result is 0 with a carry
        return addBinary(addBinary(a[:-1], b[:-1]), '1') + '0'
    elif a[-1] == '0' and b[-1] == '0':
        # Both bits are 0, so result is 0 with no carry
        return addBinary(a[:-1], b[:-1]) + '0'
    else:
        # One bit is 1, and the other is 0, so result is 1 with no
carry
        return addBinary(a[:-1], b[:-1]) + '1'


def addBinary(a: str, b: str) -> str:
    result = []

    def helper(i, j, carry):
        # Base case: if both strings are fully processed and no carry,
return
        if i < 0 and j < 0 and carry == 0:
            return
```

```
        # Get the current bits or 0 if the index is out of bounds
        digit_a = int(a[i]) if i >= 0 else 0
        digit_b = int(b[j]) if j >= 0 else 0

        # Add the bits and carry
        total = digit_a + digit_b + carry
        result.append(str(total % 2))  # Append the result for the
current bit
        carry = total // 2  # Calculate new carry

        # Recurse for the next bits
        helper(i - 1, j - 1, carry)

    # Start the recursive helper function
    helper(len(a) - 1, len(b) - 1, 0)

    # Join the result in reverse order and return it as a string
    return ''.join(reversed(result))


class Solution:
    def addBinary(self, a: str, b: str) -> str:


        shortest = a if len(a) <= len(b) else b
        longest = b if len(b) >= len(a) else a


        modify_shortest = ""

        for i in range(len(longest) - len(shortest)):
            modify_shortest += "0"

        for i in range(len(shortest)):
            modify_shortest += str(shortest[i])


        l1 = len(modify_shortest) -1
        l2 = len(longest) - 1
        carry = 0

        result = []
        while l1 >=0 and l2 >= 0:
            sum = carry

            sum += int( modify_shortest[l1] )
            sum += int( longest[l2] )

            result.append(str(sum % 2))
            carry = sum //2
```

```
            l1 -=1
            l2-=1
        if carry == 1:
            result.append("1")
        result.reverse()

        return "".join(result)



# 190. Reverse Bits
# Solved
# Easy
# Topics
# Companies
# Reverse bits of a given 32 bits unsigned integer.

# Input: n = 00000010100101000001111010011100
# Output:    964176192 (00111001011110000010100101000000)
# Explanation: The input binary string
00000010100101000001111010011100 represents the unsigned integer
43261596,
# so return 964176192 which its binary representation is
00111001011110000010100101000000.

class Solution:
    def reverseBits(self, n: int) -> int:
        # Initialize the reversed number to 0
        reversed_num = 0

        # Iterate over all 32 bits of the given number
        for i in range(32):
            # Left shift the reversed number by 1 and add the last bit
of the given number to it
            reversed_num = (reversed_num << 1) | (n & 1)
            # To add the last bit of the given number to the reversed
number, perform an AND operation with the given number and 1
            n >>= 1

        # Return the reversed number
        return reversed_num

# 108. Convert Sorted Array to Binary Search Tree
# Solved
# Easy
# Topics
# Companies
# Given an integer array nums where the elements are sorted in
ascending order, convert it to a
# height-balanced
```

```python
#  binary search tree.

# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def sortedArrayToBSTHelper(self, nums: List[int], left: int,
right: int) -> Optional[TreeNode]:

        if left > right:
            return None
        mid = (left + right) // 2

        root = TreeNode(nums[mid])

        root.left = self.sortedArrayToBSTHelper(nums, left, mid-1)
        root.right = self.sortedArrayToBSTHelper(nums, mid+1, right)

        return root

    def sortedArrayToBST(self, nums: List[int]) -> Optional[TreeNode]:

        root = self.sortedArrayToBSTHelper(nums, 0, len(nums)-1)

        return root



# 35. Search Insert Position
# Solved
# Easy
# Topics
# Companies
# Given a sorted array of distinct integers and a target value, return
the index if the target is found. If not, return the index where it
would be if it were inserted in order.

# You must write an algorithm with O(log n) runtime complexity.



# Example 1:

# Input: nums = [1,3,5,6], target = 5
# Output: 2
# Example 2:

# Input: nums = [1,3,5,6], target = 2
```

```
# Output: 1
# Example 3:

# Input: nums = [1,3,5,6], target = 7
# Output: 4


#recursive solution

 class Solution:
    def searchInsert(self, nums: List[int], target: int) -> int:

        def insertPos(nums, left, right, target):

            if left > right:
                return left

            mid = (left + right) //2

            if nums[mid] == target:
                return mid
            elif target > nums[mid]:
                return insertPos(nums, mid+1, right, target)
            else:
                return insertPos(nums, left, mid-1, target)


            return left

        return insertPos(nums, 0, len(nums)-1, target)


#iterative solution
class Solution:
    def searchInsert(self, nums: List[int], target: int) -> int:

        start : int  = 0
        end : int = len(nums) - 1

        while start <= end:
            mid : int = (start + end) // 2

            if nums[mid] == target:
                return mid
            elif nums[mid] > target:
                end = mid - 1
            else:
                start = mid + 1


        return start
```

```python
# 1768. Merge Strings Alternately
# Solved
# Easy
# Topics
# Companies
# Hint
# You are given two strings word1 and word2. Merge the strings by
adding letters in alternating order, starting with word1.
# If a string is longer than the other, append the additional letters
onto the end of the merged string.

# Return the merged string.



# Example 1:

# Input: word1 = "abc", word2 = "pqr"
# Output: "apbqcr"
# Explanation: The merged string will be merged as so:
# word1:  a   b   c
# word2:    p   q   r
# merged: a p b q c r
# Example 2:

# Input: word1 = "ab", word2 = "pqrs"
# Output: "apbqrs"
# Explanation: Notice that as word2 is longer, "rs" is appended to the
end.
# word1:  a   b
# word2:    p   q   r   s
# merged: a p b q   r   s
# Example 3:

# Input: word1 = "abcd", word2 = "pq"
# Output: "apbqcd"
# Explanation: Notice that as word1 is longer, "cd" is appended to the
end.
# word1:  a   b   c   d
# word2:    p   q
# merged: a p b q c   d

class Solution:
    def mergeAlternately(self, word1: str, word2: str) -> str:

        l1 = 0
        l2 = 0
        ans = ""
        while l1 < len(word1) and l2 < len(word2):

            if l1 == l2:
```

```
                ans += word1[l1]
                l1 +=1
            else:
                ans += word2[l2]
                l2 += 1

        while l1 < len(word1):
            ans += word1[l1]
            l1 +=1
        while l2  < len(word2):
            ans += word2[l2]
            l2 += 1

        return ans




# 190. Reverse Bits
# Solved
# Easy
# Topics
# Companies
# Reverse bits of a given 32 bits unsigned integer.
# Input: n = 00000010100101000001111010011100
# Output:    964176192 (00111001011110000010100101000000)
# Explanation: The input binary string
00000010100101000001111010011100 represents the unsigned integer
43261596,
# so return 964176192 which its binary representation is
00111001011110000010100101000000.

class Solution:

    # def reverseBits(self, n: int) -> int:

    #     result = 0
    #     while n > 0:

    #         #get the least signifcant bit: lsb

    #         lsb = n & 1

    #         #shift result to the left to accomodate 1
    #         result = result << 1

    #         #append lsb to result
    #         result = result | lsb

    #         #update n by shifting it to the right
```

```
    #         n = n >> 1


    #     return result


    #only this solution will work because i need to reverse the whole
32 bits
    def reverseBits(self, n: int) -> int:

        result = 0

        for i in range(32):

            #get the least significant bit from the right: lsb

            lsb = n & 1

            #shift result by 1 to accomodate lsb
            result = result << 1

            #append bit to result from the right

            result = result | lsb

            #update n to get it to the next bit
            #shift it to the right so that the bit can fall off
            n = n >> 1


        return result



# 191. Number of 1 Bits
# Solved
# Easy
# Topics
# Companies
# Write a function that takes the binary representation of a positive
integer and returns the number of
# set bits
#  it has (also known as the Hamming weight).

# Example 1:

# Input: n = 11

# Output: 3
```

```
# Explanation:

# The input binary string 1011 has a total of three set bits.


class Solution:
    def hammingWeight(self, n: int) -> int:

        count = 0
        while n > 0:

            #get least significant bit from the right : lsb

            lsb = n & 1

            #check if lsb = 1

            if lsb == 1:
                count +=1

            #update n by shifting it to the right
            n = n >> 1

        return count

# 136. Single Number
# Solved
# Easy
# Topics
# Companies
# Hint
# Given a non-empty array of integers nums, every element appears
twice except for one. Find that single one.

# You must implement a solution with a linear runtime complexity and
use only constant extra space.



# Example 1:

# Input: nums = [2,2,1]
# Output: 1

class Solution:
    def singleNumber(self, nums: List[int]) -> int:

        #exclusive or guarantees the unique number is num
        num = 0
        for i in range(len(nums)):
            num ^= nums[i]
```

```
        return num

# 9. Palindrome Number
# Solved
# Easy
# Topics
# Companies
# Hint
# Given an integer x, return true if x is a
# palindrome
# , and false otherwise.



# Example 1:

# Input: x = 121
# Output: true
# Explanation: 121 reads as 121 from left to right and from right to
left.
# Example 2:

# Input: x = -121
# Output: false
# Explanation: From left to right, it reads -121. From right to left,
it becomes 121-. Therefore it is not a palindrome.

class Solution:

    def isPalindrome(self, x: int) -> bool:
        if x < 0:
            return False
        if x==0:
            return True

        reversed = 0
        temp = x
        while temp > 0:

            #get the last digit: last_digit
            last_digit = temp % 10

            #append it to reversed
            reversed = reversed * 10 + last_digit

            #update number
            temp = temp // 10

        #make sure reversed is same as x
```

```
        return reversed == x

    def isPalindrome(self, x: int) -> bool:
        # Handle negative numbers and zero
        if x < 0:
            return False
        if x == 0:
            return True

        num = x
        arr = []
        while num > 0:
            num_i = num % 10
            arr.append(num_i)
            num = num // 10

        start, end = 0, len(arr)-1

        while start <= end:
            if arr[start] != arr[end]:
                return False
            start +=1
            end -= 1
        return True

# 66. Plus One
# Solved
# Easy
# Topics
# Companies
# You are given a large integer represented as an integer array
digits, where each digits[i] is
# the ith digit of the integer. The digits are ordered from most
significant to least significant
# in left-to-right order. The large integer does not contain any
leading 0's.

# Increment the large integer by one and return the resulting array of
digits.



# Example 1:

# Input: digits = [1,2,3]
# Output: [1,2,4]
# Explanation: The array represents the integer 123.
# Incrementing by one gives 123 + 1 = 124.
# Thus, the result should be [1,2,4].
```

```python
# Input: digits = [9]
# Output: [1,0]
# Explanation: The array represents the integer 9.
# Incrementing by one gives 9 + 1 = 10.
# Thus, the result should be [1,0].


class Solution:
    def plusOne(self, digits: List[int]) -> List[int]:

        for i in reversed(range(len(digits))):

            if digits[i] < 9:
                digits[i] += 1
                return digits
            digits[i] = 0
        digits.insert(0,1)
        return digits


     def plusOne(self, digits: List[int]) -> List[int]:

        last = len(digits) -1

        while last >=0:
            #if less than 9
            if digits[last] < 9:
                digits[last] += 1
                return digits
            #if not less than 9
            digits[last] = 0
            last -=1
        #if it could not be completed inside loop
        #1 must be remaining
        digits.insert(0,1)

        return digits

# 69. Sqrt(x)
# Solved
# Easy
# Topics
# Companies
# Hint
# Given a non-negative integer x, return the square root of x rounded
down to the nearest integer. The returned integer should be non-
negative as well.

# You must not use any built-in exponent function or operator.

# For example, do not use pow(x, 0.5) in c++ or x ** 0.5 in python.
```

```
# Example 1:

# Input: x = 4
# Output: 2
# Explanation: The square root of 4 is 2, so we return 2.
# Example 2:

# Input: x = 8
# Output: 2
# Explanation: The square root of 8 is 2.82842..., and since we round
it down to the nearest integer, 2 is returned.


class Solution:
    def mySqrt(self, x: int) -> int:
        left = 0
        right = x
        while left <= right:
            mid = (left + right) // 2
            if mid * mid < x:
                left = mid + 1
            elif mid * mid > x:
                right = mid -1
            else:
                #execute if mid * mid == x
                return mid

        return right

# 70. Climbing Stairs
# Solved
# Easy
# Topics
# Companies
# Hint
# You are climbing a staircase. It takes n steps to reach the top.

# Each time you can either climb 1 or 2 steps. In how many distinct
ways can you climb to the top?



# Example 1:

# Input: n = 2
# Output: 2
# Explanation: There are two ways to climb to the top.
# 1. 1 step + 1 step
# 2. 2 steps
```

```python
# Example 2:

# Input: n = 3
# Output: 3
# Explanation: There are three ways to climb to the top.
# 1. 1 step + 1 step + 1 step
# 2. 1 step + 2 steps
# 3. 2 steps + 1 step


class Solution:
    def climbStairs(self, n: int) -> int:

        table : int = [0] * 46

        table[1] = 1
        table[2] = 2

        for i in range(3, n+1):

            table[i] = table[i-1] + table[i-2]

        return table[n]

    def climbStairs(self, n: int) -> int:

        lookup = {}
        def ways(num : int, table):
            if num == 0 or num == 1:
                table[num] = 1
            if num in table:
                return table[num]

            table[num] = ways(num-1, table) + ways(num-2, table)

            return table[num]

        return ways(n, lookup)
```