1. Two Sum

```
# Duplicates in input are allowed

# search complement before inserting into hash table
# if this is done before checking for complement, case [3,3] and target = 6 f
ails
# (duplicate) or case [3,4,2] and target = 6 fails (complement must not be nu
mber itself)

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

        result = []
        d ={}

        for index,element in enumerate(nums):
            complement = target - element
            if d.get(complement) != None:
                result.append(d[complement])
                result.append(index)
                break   # question says only one unique solution hence break
when found

            d[element] = index
        return result

# Time Complexity = O(n) - we traverse the input once
# Space Complexity = O(n) - space required to create a hash table
```

2. Add Two Numbers

```
# Cases: When one list is longer than the other, extra carry at end
class Solution:
    def addTwoNumbers(self, l1: ListNode, l2: ListNode) -> ListNode:
        dummy = ListNode(0)
        carry = 0
        curr = dummy

        while l1 or l2:
            # One List can be shorter than other
            if l1:
                x = l1.val
                l1 = l1.next # advance l1 only when l1 is not null
            else:
                x = 0

            if l2:
                y = l2.val
                l2 = l2.next # advance l2 only when l2 is not null
            else:
                y = 0

            sum = x + y + carry
            carry = int(sum/10) # this carry will be used in next iteration h
ence after sum

            # dummy node was created to facilitate below
            curr.next = ListNode(sum % 10) # create link to next node, then a
dvance pointer
```

```
            curr = curr.next


        # The sum could have an extra carry of one at the end,
        if carry != 0:
            curr.next = ListNode(carry)

        return dummy.next

# Time and Space Complexity = O(max(m,n)) where m,n = length of list l1 and l
2
# Assume that m and n represents the length of l1 and l2 respectively, the al
gorithm above iterates at most max(m, n) times.
```

3. Longest Substring Without Repeating Characters

```
# define a hashmap of the characters: index
# fast and slow ptr
# At any time duplicate is seen by fast ptr, slow ptr moves +1 to fast ptr's
prev index where this char was
# seen and fast ptr overwrites the prev index with latest index where this ch
ar was seen

# All along the way keep track of length of longest substring seen

# e.g. a -> 1, dvdf -> 3, abba -> 2

class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:
        d = {}
        max_len = 0

        j = 0
        i = 0

        while j < len(s) and i < len(s):
            if d.get(s[j]) != None:
                i = max(d[s[j]]+1,i) # if i resets, it cannot go back e.g. ab
ba -> at the last 'a' i should 3 and should not go back to 0

            max_len = max(max_len, j-i+1) # O based index hence + 1, needs to
be done before inserting into dict e.g. "a" -> 1

            d[s[j]] = j # Overwrite with latest index, if occur multiple time
```

```
s
            j = j+1
        return max_len

# Time Complexity = O(n)
# Space Complexity = O(n)
```

---

4. Median of Two Sorted Arrays

https://www.youtube.com/watch?v=LPFhl65R7ww (https://www.youtube.com/watch?v=LPFhl65R7ww)

https://medium.com/@dimko1/median-of-two-sorted-arrays-b7f0c4284159 (https://medium.com/@dimko1/median-of-two-sorted-arrays-b7f0c4284159)

```
# Brute force: Merge Sort ( m+n log m+n) , where m,n = length of arrays
# Min Heap Method to merge sorted arrays: O ( m+n log 2)
# Below method: O(log min(m,n)) even when two arrays are of different size

# Binary search on smaller array
# partition the two arrays in such a way so that maxLeftX <= minRightY and mi
nRightX >= maxLeftY
class Solution:
    def findMedianSortedArrays(self, nums1: List[int], nums2: List[int]) -> f
loat:
        if len(nums1) > len(nums2):
            # we want first part to be smaller then second
             nums1, nums2 = nums2, nums1


        x = len(nums1)
        y = len(nums2)

        low = 0
        high = x

        while low <= high:
            # basic selection of the partitioner
            partitionX = (low + high) // 2
            # partitioning of the second array
            partitionY = (x + y + 1) // 2- partitionX

            # getting values. sometimes maxLeftX index can be 0
```

```
            maxLeftX = nums1[partitionX - 1] if partitionX > 0 else float('-i
nf')
            minRightX = float('inf') if partitionX == x else nums1[partition
X]

            maxLeftY = nums2[partitionY - 1] if partitionY > 0 else float('-i
nf')
            minRightY = float('inf') if partitionY == y else nums2[partition
Y]

            # Core of this algo: let's check if partition was selected correc
tly
            if maxLeftX <= minRightY and minRightX >= maxLeftY:
                if (x + y) % 2 == 0:
                    return (max(maxLeftX, maxLeftY) + min(minRightX, minRight
Y)) / 2.0
                else:
                    return max(maxLeftX, maxLeftY)
            # if not - move partitioning region
            elif maxLeftX > minRightY:
                high = partitionX -1
            else:
                low = partitionX + 1

# Time Complexity = O(log(min(len(nums1), len(nums2))))
# Space Complexity = O(1)
```

5. Longest Palindromic Substring

https://www.techiedelight.com/longest-palindromic-substring-non-dp-space-optimized-solution/ (https://www.techiedelight.com/longest-palindromic-substring-non-dp-space-optimized-solution/)

https://leetcode.com/problems/longest-palindromic-substring/discuss/2954/Python-easy-to-understand-solution-with-comments-(from-middle-to-two-ends) (https://leetcode.com/problems/longest-palindromic-substring/discuss/2954/Python-easy-to-understand-solution-with-comments-(from-middle-to-two-ends)).

```
class Solution:
    def longestPalindrome(self, s: str) -> str:
        def expand(s, low, high):
            while(low >= 0 and high<len(s) and s[low]==s[high]):
                low = low -1
                high = high + 1

            return s[low+1:high]

        max_pal =""
        max_len = 0
        for i in range(len(s)):
            curr_odd_pal = expand(s,i,i)
            curr_even_pal = expand(s,i, i+1)
            max_len = max(len(curr_odd_pal), len(curr_even_pal), max_len)

            if max_len == len(curr_odd_pal):
                max_pal = curr_odd_pal
            elif max_len == len(curr_even_pal):
                max_pal = curr_even_pal



        return max_pal

# Time Complexity = O(n^2), at each position in n, expand till ends of string
of len n
# Space Complexity = O(1)
```

7. Reverse Integer

```
# input is always in range but output could be out of range

class Solution:
    def reverse(self, x: int) -> int:
        if x > 0:
            sign = 1
        elif x < 0:
            sign = -1
        else:
            return 0

        rev = 0
        x = abs(x)
        while x:
            x, remainder = divmod(x,10)
            rev = rev*10 + remainder

        if -pow(2,31) <= sign*rev <= pow(2,31)-1 :
            return sign*rev
        else:
            return 0
# Time Complexity = O(log (number of digits)). There are roughly log(x) to th
e base 10 digits in x.
# Space Complexity = O(1)
```

8. String to Integer (atoi)

```
# invalid inputs -> only'-', only'+', only'-+' or if start is not digit -> re
turn 0
class Solution:
    def myAtoi(self, str: str) -> int:
        # case 1: empty string
        if len(str) == 0 :
            return 0

        # case 2: after stripping whitespace , empty string
        ls = list(str.strip(' '))
        if len(ls) == 0:
            return 0

        # case 3: collect sign and if only '-' -> return 0
        sign = 1 # see how sign is used
        two_strike = 0
        if ls[0] == '-' :
            sign = -1
            two_strike = two_strike + 1
            ls = ls[1:]
        if len(ls) == 0:
            return 0

        # case 4: collect sign and if only '+' -> return 0
        if ls[0] == '+':
            two_strike = two_strike + 1
            ls = ls[1:]
        if len(ls) == 0:
```

```
        return 0

    # case 5 : '-+' -> return 0
    if two_strike == 2:
        return 0

    # case 6: doesn't start with digit
    if ls[0].isdigit() == False:
        return 0

    # Main case : extract number before words begin
    ret = 0 # Important logic
    for index,element in enumerate(ls):
        if element.isdigit() == True:
            ret = ret*10 + ord(element) - ord('0')
        else:
            break

    ret = ret * sign  # Imp

    # Boundary conditions
    max_value = (2 ** 31) - 1
    if ret > max_value:
        return max_value
    if ret < - (max_value) - 1:
        return -(max_value) - 1
    return ret
```

```
# Time Complexity = O(n) for slicing and iterating thru each element of strin
g
# Space Complexity = O(n) for slicing
```

---

11. Container With Most Water

https://leetcode.com/problems/container-with-most-water/solution/ (https://leetcode.com/problems/container-with-most-water/solution/)

2 pointer approach

```
# Two pointers at opposite ends => maximizing base
# height of whichever pointer is smaller moves forward, in case of tie move b
egin ptr => max ht


class Solution:
    def maxArea(self, height: List[int]) -> int:
        begin = 0
        end = len(height) - 1

        maxarea = 0

        while begin < end:
            if height[begin] <= height[end]:
                area = height[begin] * (end-begin)
                begin = begin +1
            else:
                area = height[end] * (end-begin)
                end = end -1

            maxarea = max(area, maxarea)

        return maxarea

# Time Complexity = O(n)
# Space Complexity = O(1)
```

12. Integer to Roman

Look at solution animation:

Go from left to right collecting quotient

```
# Create a list of tuple of value and symbol in descending order of value
# Go from left to right collecting digits

class Solution:
    def intToRoman(self, num: int) -> str:
        digits = [(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")]

        roman = []
        for value,symbol in digits:
            if num == 0: # Stopping condition
                break
            count, num = divmod(num,value)
            roman.append(symbol*count)
        return "".join(roman)

# Time and Space Complexity = O(1)
```

13. Roman to Integer

```
# create a dict of char: val
# left to right pass -> if value[i] < value[i+1], take two chars and use subs
traction
# take example III and IV

class Solution:
    def romanToInt(self, s: str) -> int:
        values = {"I": 1,"V": 5,"X": 10,"L": 50,"C": 100,"D": 500,"M": 1000}

        i = 0
        result = 0
        while i < len(s): # if i+1 check done here then III -> gives wrong an
swer
            if i+1 < len(s) and values[s[i]] < values[s[i+1]]: # See how i+1
-> avoids overflow
                result = result + (values[s[i+1]] - values[s[i]])
                i = i+2 # increment by 2
            else:
                result = result + values[s[i]]
                i = i+1
        return result

# Time and Space Complexity = O(1)
```

14. Longest Common Prefix

```
# l = ['abc', 'xab', 'hello', 'prat']
# l.sort()
# O/P -> ['abc', 'hello', 'prat', 'xab']

class Solution:
    def longestCommonPrefix(self, strs: List[str]) -> str:
        if not strs: return ""
        if len(strs) == 1: return strs[0]

        strs.sort() # this will sort all strings (by prefix)
        r = ""
        for x,y in zip(strs[0], strs[-1]): # check first and last string char
by char
            if x ==y:
                r = r + x
            else: #only prefix matching
                break
        return r

# Time Complexity = O(nk log nk) where n = no. of words and k = max length of
any word for sorting. There is O(nk) comparison when comparing strs[0] and st
rs[-1] char by char but sorting complexity dominates

# Space Complexity = O(nk)
```

15. 3Sum

Comparing with 2 sum question, this question does not require index in answer hence sort can be done

https://fizzbuzzed.com/top-interview-questions-1/ (https://fizzbuzzed.com/top-interview-questions-1/)

https://www.youtube.com/watch?v=6qgC-dqKElA (https://www.youtube.com/watch?v=6qgC-dqKElA)

Best solution: https://leetcode.com/problems/3sum/discuss/533577/Simple-python-3-AC-solution-(sort-%2B-2-pointers) (https://leetcode.com/problems/3sum/discuss/533577/Simple-python-3-AC-solution-(sort-%2B-2-pointers))

```
# Two pointer approach
# Using hash table approach is trickier here to handle duplicates in input he
nce not used

class Solution:
    def threeSum(self, nums: List[int]) -> List[List[int]]:
        nums.sort()  # sort
        result = set()  # takes care of duplicate triplets

        for i in range(len(nums)-2): # take care, upto second last index

            l = i+1
            r = len(nums) - 1 # last index

            while l<r:  # < and not <= as i,l,r all have to be different
                total = nums[i] + nums[l] + nums[r]
                if total < 0:
                    l = l+1
                elif total > 0:
                    r = r -1
                elif total == 0:
                    result.add((nums[i], nums[l], nums[r])) # tuple is added
hence extra ()
                    # can find multiple l and r for the same i
                    l = l+1
                    r = r-1
        return list(map(list,result))

# Time Complexity: O(n^2)
```

```
# Time Complexity = O(n^2)
# Space Complexity = O(n)  due to sort in the beginning, if input is already

sorted then O(1)

# Python's built in sort method is a spin off of merge sort called Timsort wi
th time complexity O(n log n) and space complexity O(n) (uses temp array)
```

---

17. Letter Combinations of a Phone Number

```
class Solution:
    def letterCombinations(self, digits: str) -> List[str]:
        d = {'2': ['a', 'b', 'c'],
             '3': ['d', 'e', 'f'],
             '4': ['g', 'h', 'i'],
             '5': ['j', 'k', 'l'],
             '6': ['m', 'n', 'o'],
             '7': ['p', 'q', 'r', 's'],
             '8': ['t', 'u', 'v'],
             '9': ['w', 'x', 'y', 'z']}

        def dfs(combo, next_digits):
            if len(next_digits) == 0:
                result.append(combo[:])
                return # without return, below for loop will give index out o
f range

            for letter in d[next_digits[0]]:
                combo = combo + letter
                dfs(combo, next_digits[1:])
                combo = combo[:-1]


        result = []
        if digits:
            dfs("",digits)
        return result
```

```
#Time and Space Complexity: O(3^N * 4^M) where N is the number of digits in t
he input that maps to 3 letters and M is the number of digits in the input th
at maps to 4 letters
```

---

19. Remove Nth Node From End of List

Adding a dummy node at the head and /or tail might help to handle many edge cases where operations have to be performed at the head or the tail. The presence of dummy nodes ensures that operations will never have be executed on the head or the tail. Dummy nodes remove the headache of writing conditional checks to deal with null pointers. Be sure to remove them at the end of the operation.

```
# Edge cases: [1,2] 2 -> [2] and [1,2] 1 -> [1] and only one node case
# dummy node simplifies edge cases when removing head and cases where only on
e node is present
# Note that before deletion, slow points to one node before

class Solution:
    def removeNthFromEnd(self, head: ListNode, n: int) -> ListNode:
        dummy = ListNode(0)
        dummy.next = head

        fast = dummy
        slow = dummy

        # Advances fast pointer so that the gap between fast and slow is n no
des apart
        i = 0
        while i <= n:
            fast = fast.next
            i = i+1

        # Move fast to the end, maintaining the gap
        while fast:
            fast = fast.next
            slow = slow.next

        # Adjust pointers to drop nth node from last
        slow.next = slow.next.next
```

```
        return dummy.next
```

---

20. Valid Parentheses

```
# Put all opening parenthesis in stack, and when closing parenthesis comes, p
op + check whether it # is corresponding opening parenthesis
# Edge cases: 1) extra openeing parenthesis 2) extra closing parenthesis

class Solution:
    def isValid(self, s: str) -> bool:
        d = {')':'(', '}':'{', ']':'['}
        stack = []

        for c in s:
            if c == '[' or c == '{' or c == '(':
                stack.append(c)

            if c == ']' or c == '}' or c == ')':
                # Edge case: closing parenthesis should have a complement in
stack e.g. ()]
                if len(stack) == 0:
                    return False


                # if stack is not empty, popped element should be the complem
ent parenthesis
                if d[c] != stack.pop():
                    return False

        if len(stack) != 0: # Edge case: Stack should be empty at last e.g.
(((
            return False
```

```
        else:
            return True

# Time Complexity = O(n) traverse through the string and push(), pop() on sta
ck is O(1)
# Space Complexity = O(n)
```

---

21. Merge Two Sorted Lists

Approach 2 : Iteration

```
class Solution:
    def mergeTwoLists(self, l1: ListNode, l2: ListNode) -> ListNode:
        dummy = ListNode(0)
        prev = dummy

        while l1 and l2:
            if l1.val <= l2.val:
                prev.next = l1
                l1 = l1.next
            else:
                prev.next = l2
                l2 = l2.next

            prev = prev.next

        if l1:
            prev.next = l1

        if l2:
            prev.next = l2

        return dummy.next

# Time Complexity = O(n)
# Space Complexity = O(1)
```

22. Generate Parentheses

```
# We can start an opening bracket if we still have one (of n) left to place.
# And we can start a closing bracket if it would not exceed the number of ope
ning brackets.

# Understand with n = 2

class Solution(object):
    def generateParenthesis(self, N):
        ans = []
        def backtrack(S = '', left = 0, right = 0):
            #print(S)
            if len(S) == 2 * N:
                ans.append(S)
                return
            if left < N:
                backtrack(S+'(', left+1, right)
            if right < left:
                backtrack(S+')', left, right+1)
            #print(S)
        backtrack()
        return ans

# Time and Space Complexity = nth catalan number bound by O(4^n / n^1/2)
```

23. Merge k Sorted Lists

https://www.youtube.com/watch?v=ptYUCjfNhJY (https://www.youtube.com/watch?v=ptYUCjfNhJY) (Read the description of above video which is below):

When building our final output array that is the sorted set of all the items we will be placing an item one by one from the original k arrays What is are the items in each of those k arrays that interest us? The smallest items.

How can we keep track of the smallest item of the k smallest items respective to each array?

A min-heap is optimal.

Since we will hold at max k items in the min-heap we know what we will have at least O(k) time complexity where k is # of sorted arrays we need to maintain the smallest item across.

We will take the smallest item from the min-heap and remember the array it came from so that we can add the next item in that array to the min heap (if it exists).

Example Walkthrough:

Input [[3, 5, 7], [0, 6], [0, 6, 28]]

Our output array will have size 8 [_, _, _, _, _, _, _, _]

Let us step through this:

Initialize: Min Heap: [3, 0, 0]

[0, _, _, _, _, _, _, _] Min Heap: [3, 0, 6] (0 came from array 2, we add the next item 6)

[0, 0, _, _, _, _, _, _] Min Heap: [3, 6, 6] (0 came from array 3, we add the next item 6)

[0, 0, 3, _, _, _, _, _] Min Heap: [5, 6, 6] (3 came from array 1, we add the next item 5)

[0, 0, 3, 5, _, _, _, _] Min Heap: [7, 6, 6] (5 came from array 1, we add the next item 7)

[0, 0, 3, 5, 6, _, _, _] Min Heap: [7, 6] (6 came from array 2, array 2 is finished)

[0, 0, 3, 5, 6, 6, _, _] Min Heap: [7, 28] (6 came from array 3, we add the next item 28)

[0, 0, 3, 5, 6, 6, 7, _] Min Heap: [28] (7 came from array 1, array 1 is finished)

[0, 0, 3, 5, 6, 6, 7, 28] Min Heap: [] (28 came from array 3, array 3 is finished)

Output [0, 0, 3, 5, 6, 6, 7, 28]

Complexities

Time: O(2(n * log(k))) = O(n * log(k))

Let n be the total elements across the k sorted arrays we are given.

Extracting and adding to the min heap will both take log(k) time.

For each of the n items, we will do an addition to the heap and removal from the heap (log(k) expense per heap operation).

Hence the times 2. It is dropped. We get O(n * log(k)).

Space: O(k)

Our heap will need to hold k elements for most of this process and cannot worsen past this.

We are not counting the size of the output array which uses O(n) space since that isn't core to our algorithm.

```
class Solution:
    def mergeKLists(self, lists: List[ListNode]) -> ListNode:
        curr = head = ListNode(0)
        count = 0
        queue = []

        for l in lists: # lists contain the head ptr to all lists
            if l:
                count = count + 1
                heapq.heappush(queue, (l.val, count, l)) # This pushes only t
he first elements of each list

        while len(queue) > 0:
            _,_, curr.next = heapq.heappop(queue)
            curr = curr.next
            if curr.next is not None:
                count = count + 1
                heapq.heappush(queue, (curr.next.val, count, curr.next))

        return head.next

# count var - it handles the case of a "tie" when two list nodes have the sam
e value. When that
# happens, Python will look at the next value in the tuple (in this case, cou
nt), and sort based
# on that value. Without count, a "tie" would error out if the next value in
the tuple were a
# ListNode (which can't be compared).
```

```
# Space Complexity: O ( k ), Our heap will need to hold k elements for most o
f this process and cannot worsen past this.

# Time Complexity: O (n * 2 * log k), Extracting and adding to the min heap w
ill both take log(k) time  as max number of items in heap at a time is k (2 -
> add + delete min), we do it for all elements in all lists which we  denote
by n
```

---

31. Next Permutation

https://www.youtube.com/watch?v=hPd4MFdg8VU (https://www.youtube.com/watch?v=hPd4MFdg8VU)

Approach 2 : Single Pass https://leetcode.com/problems/next-permutation/solution/ (https://leetcode.com/problems/next-permutation/solution/)

```
# The replacement must be in-place and use only constant extra memory.
# if digits are in descending order -> no higher permutation possible, revers
e the digits

# Find the first peak from right => peak - 1 is the first valley from right
# swap this digit with the next higher digit to right
# reverse the digits which are right of this swapped digit  (excluding the sw
ap digit)

# e.g. 1243 -> 1342 -> 1324

class Solution:
    def nextPermutation(self, nums: List[int]) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """

        i = len(nums)-1
        while i > 0 and nums[i-1] >= nums[i]:
            i -= 1

        if i == 0:   # nums are in descending order
            nums.reverse()
            return

        print(i)
        # find the first element from right which is greater than first valle
y from right
```

```
        j = len(nums)-1
        while nums[j] <= nums[i-1]: # <= and j decremented below => next high
er digit to nums[i-1]
            j -= 1
        print(j)
        # swap
        nums[i-1], nums[j] = nums[j], nums[i-1]

        print(nums)

        # reverse the second part
        l, r = i, len(nums)-1
        while l <= r:
            nums[l], nums[r] = nums[r], nums[l]
            l +=1 ; r -= 1

# Time Complexity = O(n)
# Space Complexity = O(1)
```

33. Search in Rotated Sorted Array

Look at approach 2, One -pass binary search

```
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        begin = 0
        end = len(nums) - 1

        while begin <= end:
            mid = (begin + end) // 2
            if nums[mid] == target:
                return  mid
            if nums[mid] > nums[end]:
                if nums[mid] > target and nums[begin] <= target: # <=
                    end = mid -1
                else:
                    begin = mid +1
            else: # else and not another if
                if nums[mid] < target and nums[end] >= target: # >=
                    begin = mid + 1
                else:
                    end = mid - 1
        return -1

# Time Complexity = O(log n)
# Space Complexity = O(1)
```

34. Find First and Last Position of Element in Sorted Array

```
class Solution:
    def searchRange(self, nums: List[int], target: int) -> List[int]:
        if not nums: # edge case
            return [-1, -1]

        def bisect_left(nums, target):
            l, r = 0, len(nums) - 1

            while l < r:
                m = l + (r -l) // 2
                if nums[m] < target:
                    l = m + 1
                else:
                    r = m

            return l if nums[l] == target else -1

        def bisect_right(nums, target):
            l, r = 0, len(nums) - 1

            while l < r:
                m = l + (r -l) // 2 + 1   # Imp: + 1
                if nums[m] > target:
                    r = m - 1
                else:
                    l = m

            return l if nums[l] == target else -1
```

```
        return [bisect_left(nums, target), bisect_right(nums, target)]

# Time Complexity = O(log n) -> binary search 2 times
# Space complexity = O(1), in-place
```

---

39. Combination Sum

https://leetcode.com/problems/combination-sum/discuss/16502/A-general-approach-to-backtracking-questions-in-Java-(Subsets-Permutations-Combination-Sum-Palindrome-Partitioning) (https://leetcode.com/problems/combination-sum/discuss/16502/A-general-approach-to-backtracking-questions-in-Java-(Subsets-Permutations-Combination-Sum-Palindrome-Partitioning))

complexity : https://leetcode.com/problems/combination-sum-iii/discuss/427713 (https://leetcode.com/problems/combination-sum-iii/discuss/427713)
https://algorithmsandme.com/tag/leetcode-combination-sum/ (https://algorithmsandme.com/tag/leetcode-combination-sum/)

```
# Input has no duplicates
# number can be chosen from candidates unlimited number of times
class Solution:
    def combinationSum(self, candidates: List[int], target: int) -> List[List
[int]]:
        def dfs(temp_list, remain, start):
            if remain == 0: # base cond
                result.append(temp_list[:])
                                return
            if remain < 0: # base cond
                return
            for i in range(start,len(candidates)):  # Solution set should not
contain duplicates
                temp_list.append(candidates[i])
                dfs(temp_list, remain - candidates[i], i) # not i + 1 because
we can reuse
                temp_list.pop()

        result = []
        dfs([], target, 0)
        return result

# Time Complexity if numbers cannot be reused =  C(n,1) + C(n,2) + … + C(n,n)
= 2^n - C(n,0) = O(2^n)
# Now every number can be reused, the max number of times a number can be use
d = how many times the smallest number goes into target e.g. if target was 10
and smallest number was 2 then max number of times 2 can be used 10/2 = 5. he
nce time complexity = O((target/smallest number) * 2^n)
```

```
# Space Complexity if numbers cannot be reused = O(n) ie. all numbers togethe
r sum to target (worst case)
# However numbers can be repeated so again max number of times a number can b
e used = how many times the smallest number goes into target => hence space c
omplexity = O((target/smallest number) * n)
```

---

40. Combination Sum II

```
# The input has duplicates -> combinations can be duplicate but we want uniqu
e
# But we cannot reuse same element hence backtrack/dfs from next position
class Solution:
    def combinationSum2(self, candidates: List[int], target: int) -> List[Lis
t[int]]:
        def dfs(candidates, start, remain, combo):
            if remain == 0:
                result.append(combo[:])

            if remain < 0:
                return

            for i in range(start, len(candidates)):
                if i > start and candidates[i] == candidates[i-1]: # skip i i
f i and i-1 are same
                    continue

                combo.append(candidates[i])
                dfs(candidates, i+1, remain - candidates[i], combo) # i+1: ca
nnot reuse same element
                combo.pop()




        result = []
        candidates.sort() # Imp: if sorted then if i and i-1 are same element
```

```
s, skip i
        dfs(candidates, 0, target, [])
        return result

# Time Complexity if numbers cannot be reused =  C(n,1) + C(n,2) + … + C(n,n)
= 2^n – C(n,0) = O(2^n)
# Space Complexity if numbers cannot be reused = O(n) ie. all numbers togethe
r sum to target (worst case)
```

---

42. Trapping Rain Water

See Approach 4 in solutions

```
# Understand below brute force approach
#  For each element in the array, find minimum of maximum height of bars on b
oth the sides minus its own height.

# Also see dynamic programming approach in solution

class Solution:
    def trap(self, height: List[int]) -> int:
        left = 0
        right = len(height) - 1

        left_max = 0
        right_max = 0

        result = 0

        while left < right:
            if height[left] <= height[right]:
                if height[left] > left_max:
                    left_max = height[left]
                else:
                    result = result + (left_max - height[left])

                left = left + 1
            else:
                if height[right] > right_max:
                    right_max = height[right]
                else:
```

```
                    result = result + (right_max - height[right])
                right = right - 1

        return result

# Time Complexity = O(n)
# Space Complexity = O(1)
```

---

46. Permutations

[1,2,3]

nums = [1, 2, 3] permutation = [] i = 0 nums = [2, 3] permutation = [1] i = 0 nums = [3] permutation = [1, 2] i = 0 nums = [] permutation = [1, 2, 3] unwinding unwinding i = 1 nums = [2] permutation = [1, 3] i = 0 nums = [] permutation = [1, 3, 2] unwinding unwinding unwinding i = 1 nums = [1, 3] permutation = [2] i = 0 nums = [3] permutation = [2, 1] i = 0 nums = [] permutation = [2, 1, 3] unwinding unwinding i = 1 nums = [1] permutation = [2, 3] i = 0 nums = [] permutation = [2, 3, 1] unwinding unwinding unwinding i = 2 nums = [1, 2] permutation = [3] i = 0 nums = [2] permutation = [3, 1] i = 0 nums = [] permutation = [3, 1, 2] unwinding unwinding i = 1 nums = [1] permutation = [3, 2] i = 0 nums = [] permutation = [3, 2, 1] unwinding unwinding unwinding

https://www.youtube.com/watch?v=KukNnoN-SoY (https://www.youtube.com/watch?v=KukNnoN-SoY)

https://leetcode.com/problems/combination-sum/discuss/16502/A-general-approach-to-backtracking-questions-in-Java-(Subsets-Permutations-Combination-Sum-Palindrome-Partitioning) (https://leetcode.com/problems/combination-sum/discuss/16502/A-general-approach-to-backtracking-questions-in-Java-(Subsets-Permutations-Combination-Sum-Palindrome-Partitioning))

https://leetcode.com/problems/permutations/discuss/360280/Python3-backtracking (https://leetcode.com/problems/permutations/discuss/360280/Python3-backtracking)

https://leetcode.com/problems/permutations-ii/discuss/309479/Simple-Python-DFS-solutions-for-8-backtrack-problems (https://leetcode.com/problems/permutations-ii/discuss/309479/Simple-Python-DFS-solutions-for-8-backtrack-problems)

https://www.geeksforgeeks.org/time-complexity-permutations-string/ (https://www.geeksforgeeks.org/time-complexity-permutations-string/)

```
class Solution:
    def permute(self, nums: List[int]) -> List[List[int]]:
        result = []
        def dfs(nums, permutation):
            #print("nums = {}".format(nums))
            #print("permutation = {}".format(permutation))
            if nums == []:
                result.append(permutation)
                return

            for i,x in enumerate(nums):
                #print("i = {}".format(i))
                dfs(nums[:i] + nums[i+1:], permutation + [nums[i]])
                #print("unwinding")


        dfs(nums, [])
        return result

# Time Complexity = O(N * N!) (Upper bound) + Slicing operation = O(N)
# Space complexity : O(N * N!) since one has to keep N! solutions.
```

47. Permutations II

https://leetcode.com/problems/combination-sum/discuss/16502/A-general-approach-to-backtracking-questions-in-Java-(Subsets-Permutations-Combination-Sum-Palindrome-Partitioning) (https://leetcode.com/problems/combination-sum/discuss/16502/A-general-approach-to-backtracking-questions-in-Java-(Subsets-Permutations-Combination-Sum-Palindrome-Partitioning))

https://leetcode.com/problems/permutations-ii/discuss/309479/Simple-Python-DFS-solutions-for-8-backtrack-problems (https://leetcode.com/problems/permutations-ii/discuss/309479/Simple-Python-DFS-solutions-for-8-backtrack-problems)

https://www.youtube.com/watch?v=KukNnoN-SoY (https://www.youtube.com/watch?v=KukNnoN-SoY)

```
class Solution:
    def permuteUnique(self, nums: List[int]) -> List[List[int]]:
        def dfs(nums, permutation, result):
            if nums == []:
                result.append(permutation)

            for i,x in enumerate(nums):
                dfs(nums[:i] + nums[i+1:], permutation + [nums[i]], result)

        result = []
        dfs(nums, [], result)

        # to find uniques in list of list, simply doing set() won't cut it!
        # also set => hash, list is mutable hence cannot be the key, need to
convert to tuple
        s = set()
        for l in result:
            s.add(tuple(l))
        return list(s)

# Note instead of deduping at last using set, we can use result as set instea
d of list

# Time Complexity = O(N * N!) (Upper bound) + Slicing operation = O(N)
# Space complexity : O(N!) since one has to keep N! solutions.
```

48. Rotate Image

```
# Transpose + reverse rows
class Solution:
    def rotate(self, matrix: List[List[int]]) -> None:
        """
        Do not return anything, modify matrix in-place instead.
        """
        # rememeber this is square matrix so: # of rows = # of cols
        # also, that's why transpose can be done in below way
        for i in range(len(matrix)):
            for j in range(i,len(matrix[0])):
                matrix[j][i], matrix[i][j] = matrix[i][j], matrix[j][i]

        for i in range(len(matrix)):
            matrix[i].reverse()

# Time Complexity = O(n^2)
# Space Complexity = O(1)
```

49. Group Anagrams

```
# key - sorted string, value - input string
# dict key = tuple, value = list
# One -pass solution
from collections import defaultdict
class Solution:
    def groupAnagrams(self, strs: List[str]) -> List[List[str]]:
        result = defaultdict(list)

        for s in strs:
            result[tuple(sorted(s))].append(s) # append function
        return result.values()

# The builtin list type should not be used as a dictionary key. Note that sin
ce tuples are
# immutable, they do not run into the troubles of lists - they can be hashed
by their contents
# without worries about modification.

# Time Complexity: O(n k log k), where n is the length of strs, and k is the
maximum length of a string in strs. Outer loop = O(n) and sort each string O
(k log k)

# Space Complexity = O(n k) # max hash table size with n keys and max k value
s to each key
```

50. Pow(x, n)

```
# n is an int (-ve, 0 and +ve all possible)

class Solution:
    def myPow(self, x: float, n: int) -> float:
        if n < 0:
            x = 1/x
            n = -n

        if n == 0:
            return 1

        answer = 1
        current_product = x

        i = n
        while i > 0:
            if i % 2 == 1: # this will be hit twice, once when i gets odd and
at last when i reaches 1, hence return variable is 'answer'
                answer = answer * current_product
                i = i-1

            current_product = current_product * current_product
            i = i/2

        return answer
```

```
# Time Complexity = O(log n)
# Space Complexity = O(1)
```

---

53. Maximum Subarray

```
# Initialize first element as max sum
# for every element from 1 to n, either it is part of running continuous sum
or start of new continuous sum
# Also keep track of global max sum i.e. max continuous sum seen till now

class Solution:
    def maxSubArray(self, nums: List[int]) -> int:
        global_max_sum = nums[0]
        local_max_sum = nums[0]

        for i in range(1, len(nums)):
            local_max_sum = max(nums[i], local_max_sum + nums[i])
            global_max_sum = max(global_max_sum, local_max_sum)

        return global_max_sum

# Time Complexity = O(n)
# Space Complexity = O(1)
```

54. Spiral Matrix

http://theoryofprogramming.com/2017/12/31/print-matrix-in-spiral-order/
(http://theoryofprogramming.com/2017/12/31/print-matrix-in-spiral-order/)

```
class Solution:
    def spiralOrder(self, matrix: List[List[int]]) -> List[int]:
        if len(matrix) == 0 or len(matrix[0]) == 0:
            return []

        top = 0
        bottom = len(matrix) - 1
        left = 0
        right = len(matrix[0]) - 1


        dir = 1
        result = []
        while (top<=bottom) and (left<=right):
            if dir == 1:
                for i in range(left,right+1):
                    result.append(matrix[top][i])
                top=top+1
                dir = dir + 1

            elif dir == 2:
                for i in range(top,bottom+1):
                    result.append(matrix[i][right])
                right = right - 1
                dir = dir + 1

            elif dir == 3:
                for i in range(right,left-1,-1):
```

```
                    result.append(matrix[bottom][i])
                bottom = bottom -1
                dir = dir + 1

            elif dir == 4:
                for i in range(bottom,top-1,-1):
                    result.append(matrix[i][left])
                left = left + 1
                dir = 1

        return result

# Time Complexity = O(n)
# Space Complexity = O(1), if result array space is not included
```

56. Merge Intervals

```
class Solution:
    def merge(self, intervals: List[List[int]]) -> List[List[int]]:

        intervals.sort(key=lambda x: x[0]) # sort by begin time

        merged = []

        begin = 0
        end = 1
        for interval in intervals:
            # if merged is empty (in case input is empty list) or intervals d
o not overlap
                        # no merging required
            if not merged or merged[-1][end] < interval[begin]:
                merged.append(interval)
            else:
                # intervals overlap, update the ending time
                merged[-1][end] = max(merged[-1][end], interval[end])

        return merged

# Time Complexity = O(n log n) for sorting
# Space Complxity = O(n) for sorting
```

57. Insert Interval

```
# Intervals are already sorted and non-overlaping

class Solution:
    def insert(self, intervals: List[List[int]], newInterval: List[int]) -> L
ist[List[int]]:
        begin = 0
        end = 1

        index = 0
        result = []

        # add all intervals starting before newInterval
        while index < len(intervals) and intervals[index][begin] < newInterva
l[begin]:
            result.append(intervals[index])
            index = index + 1

        # add newInterval
        # if newInterval is before intervals (first element) or
        # there is no overlap with the last interval in result, just add the
newInterval
        # else merge with the last interval
        if not result or result[-1][end] < newInterval[begin]:
            result.append(newInterval)
        else:
            result[-1][end] = max(result[-1][end], newInterval[end])
```

```
        # add next intervals, merge if needed
        while index < len(intervals):
            if result[-1][end] < intervals[index][begin]:
                result.append(intervals[index])
            else:
                result[-1][end] = max(result[-1][end], intervals[index][end])

            index = index + 1

        return result

# Time Complexity = O(n), array is already sorted and one-pass
# Space Complexity = O(n) for result array
```

---

59. Spiral Matrix II

```
class Solution:
    def generateMatrix(self, n: int) -> List[List[int]]:
        matrix = [[0 for _ in range(n)] for _ in range(n)]

        top = 0
        bottom = len(matrix) - 1
        left = 0
        right = len(matrix[0]) - 1


        dir = 1
        result = []
        counter = 0
        while (top<=bottom) and (left<=right):
            if dir == 1:
                for i in range(left,right+1):
                    counter  = counter + 1
                    matrix[top][i] = counter
                top=top+1
                dir = dir + 1

            elif dir == 2:
                for i in range(top,bottom+1):
                    counter  = counter + 1
                    matrix[i][right] = counter
                right = right - 1
                dir = dir + 1
```

```
            elif dir == 3:
                for i in range(right,left-1,-1):
                    counter  = counter + 1
                    matrix[bottom][i] = counter
                bottom = bottom -1
                dir = dir + 1

            elif dir == 4:
                for i in range(bottom,top-1,-1):
                    counter  = counter + 1
                    matrix[i][left] = counter
                left = left + 1
                dir = 1

        return matrix

# Time Complexity = O(n)
# Space Complexity = O(n), matrix with n elements is populated
```

62. Unique Paths

https://www.youtube.com/watch?v=GO5QHC_BmvM (https://www.youtube.com/watch?v=GO5QHC_BmvM)

```
# Base case: 0th row and 0th column = 1 (# of unique paths to traverse 0th ro
w or 0th column )
class Solution:
    def uniquePaths(self, m: int, n: int):
        if m == 0 or n == 0: # edge case
            return 0

        path_sum =[[1 for _ in range(n)] for _ in range(m)] # Initialize matr
ix with all 1's

        for i in range(1,m):
            for j in range(1,n):
                path_sum[i][j] = path_sum[i-1][j] + path_sum[i][j-1]

        return path_sum[m-1][n-1]

# Time and Space complexity = 0(m*n)
```

---

63. Unique Paths II

```
class Solution:
    def uniquePathsWithObstacles(self, obstacleGrid: List[List[int]]) -> int:
        rows = len(obstacleGrid)
        cols = len(obstacleGrid[0])

        # Flip 0 and 1
        obstacleGrid = [[ obstacleGrid[row][col] ^ 1 for col in range(cols)]
for row in range(rows)]

        # Base Case - top row and first column
        # See how if obstacle is encountered in base case, remaining row or c
ol becomes obstacle
        for row in range(1,rows):
            obstacleGrid[row][0] = int(obstacleGrid[row][0] and obstacleGrid
[row-1][0])

        for col in range(1,cols):
            obstacleGrid[0][col] = int(obstacleGrid[0][col] and obstacleGrid
[0][col-1])

        for i in range(1, rows):
            for j in range(1, cols):
                if obstacleGrid[i][j] == 0:# if obstacle, move ahead
                    continue
                else:
                    obstacleGrid[i][j] = obstacleGrid[i-1][j] + obstacleGrid
[i][j-1]
```

```
        return obstacleGrid[rows-1][cols-1]

# Time Complexity = O(m*n)
# Space Complexity = O(1) , no extra space used
```

---

64. Minimum Path Sum

```
# Base case: 0th row and Oth column = cumulative sum
# min for ith row and jth column = Take min from left,top and add to current
value
# edge case: # rows = 0 or # cols = 0
class Solution:
    def minPathSum(self, grid: List[List[int]]) -> int:
        rows = len(grid)
        cols = len(grid[0])

        if rows == 0 or cols == 0: # edge case
            return 0

        # Base case : cumulative sum along Oth row and Oth col
        for row in range(1,rows): # Oth col
            grid[row][0] = grid[row-1][0] + grid[row][0]

        for col in range(1,cols): # Oth row
            grid[0][col] = grid[0][col-1] + grid[0][col]

        # Take min from left,top and add to current value
        for i in range(1,rows):
            for j in range(1,cols):
                grid[i][j] = grid[i][j] + min(grid[i-1][j], grid[i][j-1])

        return grid[rows-1][cols-1]

# Time Complexity = O(m*n)
# Space Complexity = O(1)
```

67. Add Binary

cannot use the modulus and quotient logic as add strings problem

69. Sqrt(x)

```
# Square root of non-negative int => returns floor of result
# for x >= 2, square root is always between 2 and floor(x/2) -> binary search

class Solution:
    def mySqrt(self, x: int) -> int:
        if x < 2: # edge case 0,1
            return x

        left = 2 # start from 2
        right = x // 2 # floor int division

        while left <= right:
            # in binary search use this formula to find middle, works when le
ft = 0 and when left = any other number (2 in this case)
            sqrt = left + (right - left) //2
            sqr = sqrt * sqrt

            if sqr == x:
                return sqrt

            if sqr > x:
                right = sqrt -1
            elif sqr < x:
                left = sqrt + 1


        return right  # for x = 3 etc. (if exact sqrt int is not found, retur
n the floor (sqrt))
```

```
# Time Complexity = O(log n)
# Space Complexity = O(1)
```

---

73. Set Matrix Zeroes

```
# Brute force: Iterate thru the entire matrix and if an element is zero, stor
e its row and column and then again iterate over the entire matrix and if its
row and column matches whatever was stored in first pass, make it all zeros.
Space Complexity = O(M+N) as need to store all rows and cols which are zero

# Need space complexity: O(1)

# Use the first row and first column to track rows and columns where 0 is see
n
# Now since first row and first col are used for tracking, first find out whe
ther any element in first row or  # col is zero, since their values will be o
ver written
# matrix[0][0] also needs extra var to keep track of whether it is zero or no
t


class Solution:
    def setZeroes(self, matrix: List[List[int]]) -> None:
        """
        Do not return anything, modify matrix in-place instead.
        """
        col0_make_zero = False
        row0_make_zero = False
        row0_col0_make_zero = False

        if matrix[0][0] == 0:
            row0_col0_make_zero = True
```

```
        for col in range(len(matrix[0])):
            if matrix[0][col] == 0:
                row0_make_zero = True

        for row in range(len(matrix)):
            if matrix[row][0] == 0:
                col0_make_zero = True

        for row in range(len(matrix)):
            for col in range(len(matrix[0])):
                if matrix[row][col] == 0:
                    matrix[row][0] = 0
                    matrix[0][col] = 0

        for row in range(1, len(matrix)):
            for col in range(1, len(matrix[0])):
                if matrix[row][0] == 0 or matrix[0][col] == 0:
                    matrix[row][col] = 0

        if row0_col0_make_zero:
            for col in range(len(matrix[0])):
                matrix[0][col] = 0
            for row in range(len(matrix)):
                matrix[row][0] = 0



        if col0_make_zero:
            for row in range(len(matrix)):
```

```
                matrix[row][0] = 0

        if row0_make_zero:
            for col in range(len(matrix[0])):
                matrix[0][col] = 0


# Time Complexity = O(M*N)
# Space Complexity = O(1)
```

---

74. Search a 2D Matrix

```
# Unroll the matrix into an array from index 0 to index m*n-1
# Now given an index, row = index // n (quotient) and col = index %n (remaind
er)
# Usual Binary search

class Solution:
    def searchMatrix(self, matrix: List[List[int]], target: int) -> bool:
        m = len(matrix)
        if m == 0:
            return False
        n = len(matrix[0])

        low = 0
        high = m*n-1

        while low<=high:
            middle = low + (high-low) // 2
            middle_element = matrix[middle //n][middle % n]

            if target == middle_element:
                return True

            if target > middle_element:
                low = middle+1
            elif target < middle_element:
                high = middle-1

        return False
```

```
# Time Complexity = O(log mn)
# Space Complexity = O(1)
```

---

75. Sort Colors

Check Solution

```
# p0 and p2 are ptrs to keep track of position where swapping "needs to happe
n"
# which means curr should traverse upto p2 (since p2 will be one index less t
han all 2s encountered)
# curr is used for traversal till 2's begin
class Solution:
    def sortColors(self, nums: List[int]) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """
        p0 = 0  # rightmost boundary 0s
        p2 = len(nums) - 1 # leftmost boundary of 2s

        curr = 0
        while curr <= p2: # [2,0,1] if not '<=' then ans [1,0,2] which is wro
ng
            if nums[curr] == 0:
                nums[curr], nums[p0] = nums[p0], nums[curr]
                curr = curr + 1
                p0 = p0 + 1

            elif nums[curr] == 2:
                nums[curr], nums[p2] = nums[p2], nums[curr]
                p2 = p2 -1
                # not needed curr = curr + 1

            else: # for 1s in the middle
                curr = curr + 1
```

```
# Time Complexity = O(n)
# Space Complexity = O(1)
```

https://www.interviewbit.com/tutorial/insertion-sort-algorithm/ (https://www.interviewbit.com/tutorial/insertion-sort-algorithm/) Time Complexity in worst case O(n^2) but Space Complexity is O(1) example - > [10,12,14,11]

```
def insertionSort(arr):
    # Traverse through 1 to len(arr)
    for i in range(1, len(arr)):
        key = arr[i]
        # Move elements of arr[0..i-1], that are
        # greater than key, to one position ahead
        # of their current position
        j = i-1
        while j >=0 and key < arr[j] :
            arr[j+1] = arr[j]
            j -= 1
        arr[j+1] = key
```

https://runestone.academy/runestone/books/published/pythonds/SortSearch/TheMergeSort.html (https://runestone.academy/runestone/books/published/pythonds/SortSearch/TheMergeSort.html)

```
def mergeSort(alist):
    print("Splitting ",alist)
    if len(alist)>1:
        mid = len(alist)//2
        lefthalf = alist[:mid]
        righthalf = alist[mid:]

        mergeSort(lefthalf)
        mergeSort(righthalf)

        i=0
        j=0
        k=0
        while i < len(lefthalf) and j < len(righthalf):
            if lefthalf[i] <= righthalf[j]:
                alist[k]=lefthalf[i]
                i=i+1
            else:
                alist[k]=righthalf[j]
                j=j+1
            k=k+1

        while i < len(lefthalf):
            alist[k]=lefthalf[i]
            i=i+1
            k=k+1

        while j < len(righthalf):
```

```
            alist[k]=righthalf[j]
            j=j+1
            k=k+1
    print("Merging ",alist)

alist = [54,26,93,17,77,31,44,55,20]
mergeSort(alist)
print(alist)
```

https://www.interviewbit.com/tutorial/quicksort-algorithm/ (https://www.interviewbit.com/tutorial/quicksort-algorithm/) https://www.youtube.com/watch?v=uXBnyYuwPe8 (https://www.youtube.com/watch?v=uXBnyYuwPe8)

```
# This function takes last element as pivot, places
# the pivot element at its correct position in sorted
# array, and places all smaller (smaller than pivot)
# to left of pivot and all greater elements to right
# of pivot

# i keeps track of the element just smller than pivot
# j keeps moving forward if element at j is greater than pivot
def partition(arr,low,high):
    i = ( low-1 )         # index of smaller element
    pivot = arr[high]     # pivot

    for j in range(low , high):

        # If current element is smaller than or
        # equal to pivot
        if arr[j] <= pivot:

            # increment index of smaller element
            i = i+1
            arr[i],arr[j] = arr[j],arr[i]

    arr[i+1],arr[high] = arr[high],arr[i+1]
    return ( i+1 )

# Function to do Quick sort
def quickSort(arr,low,high):
    if low < high:
```

```
        # pi is partitioning index, arr[p] is now
        # at right place
        pi = partition(arr,low,high)

        # Separately sort elements before
        # partition and after partition
        quickSort(arr, low, pi-1)
        quickSort(arr, pi+1, high)
```

---

76. Minimum Window Substring

Not really substring but subsequence (not continguous)

https://leetcode.com/problems/minimum-window-substring/discuss/26808/here-is-a-10-line-template-that-can-solve-most-substring-problems (https://leetcode.com/problems/minimum-window-substring/discuss/26808/here-is-a-10-line-template-that-can-solve-most-substring-problems)

https://www.youtube.com/watch?v=MK-NZ4hN7rs (https://www.youtube.com/watch?v=MK-NZ4hN7rs)

https://medium.com/outco/how-to-solve-sliding-window-problems-28d67601a66 (https://medium.com/outco/how-to-solve-sliding-window-problems-28d67601a66)

https://www.techiedelight.com/sliding-window-problems/ (https://www.techiedelight.com/sliding-window-problems/)

---

77. Combinations

https://www.youtube.com/watch?v=7IQHYbmuoVU (https://www.youtube.com/watch?v=7IQHYbmuoVU)

Another solution similar to permutations: https://leetcode.com/problems/combinations/discuss/26990/Easy-to-understand-Python-solution-with-comments (https://leetcode.com/problems/combinations/discuss/26990/Easy-to-understand-Python-solution-with-comments).

https://v4.software-carpentry.org/python/alias.html (https://v4.software-carpentry.org/python/alias.html)

What does [:] do in Python?

It's a slicing, and what it does depends on the type of population . If population is a list, this line will create a shallow copy of the list. For an object of type tuple or a str , it will do nothing (the line will do the same without [:])

Because the list nums is being modified during the function calls. If you just append it to the output you append a reference (pointer) to nums not the actual list which means that after nums is modified from some other recursive function it will be "changed" in the output list that stores the reference to

nums. In the end, output will contain pointers that will point to the same result (whatever was the last change in nums). So you need to make a deep copy of nums. I suggest you to look over list aliasing in Python.

```
class Solution:
    def combine(self, n: int, k: int) -> List[List[int]]:

        result = []
        def dfs(start, combination):
            if len(combination) == k:  # Base condition
                result.append(combination[:]) # Imp: List Aliasing in python
since list is mutable
                return


            for i in range(start, n+1):
                combination.append(i)
                dfs(i+1, combination) # recurse from i+1
                combination.pop() # After base condition is met - backtrack


        dfs(1, [])
        return result

# Time Complexity = O(k * nCk)
# Space Complexity = O( k * nCk)

# each time you run the backtracking method it will place 1 number in the cor
rect position.
# so how many numbers are there in the final result?
# that is = No. of numbers in one possible combination * No. of possible comb
inations. right?
```

```
# No. of numbers in one possible combination = k
# No. possible combinations. = nCk
```

---

78. Subsets

```
# Find all combinations of size 1,2...len(nums)

class Solution:
    def subsets(self, nums: List[int]) -> List[List[int]]:
        def dfs(start, subset,k):
            if len(subset) == k:
                result.append(subset[:])
                                return
            for i in range(start, len(nums)):
                subset.append(nums[i])
                dfs(i+1, subset,k)
                subset.pop()

        result = []
        for k in range(len(nums)+1): # Find all combinations of size 1,2...le
n(nums)
            dfs(0, [], k)
        return result

# Time complexity: O(N * 2^N) to generate all subsets and then copy them into
output list
# Space Complexity: O(N * 2^N) to generate all subsets since each of N elemen
ts could be present or absent.
```

79. Word Search

Basic idea is DFS with either a set to keep track of which characters were explored or directly mark expored character as * (invalid) and unmark after dfs backtracks.

https://cs.stackexchange.com/questions/96626/whats-the-big-o-runtime-of-a-dfs-word-search-through-a-matrix (https://cs.stackexchange.com/questions/96626/whats-the-big-o-runtime-of-a-dfs-word-search-through-a-matrix) Time Complexity: The complexity will be $O(m*n*4$ ^s) where m is the no. of rows and n is the no. of columns in the 2D matrix and s is the length of the input string.

When we start searching from a character we have 4 choices of neighbors for the first character and subsequent characters have only 3 or less than 3 choices but we can take it as 4 (permissible slopiness in upper bound). This slopiness would be fine in large matrices. So for each character we have 4 choices. Total no. of characters are s where s is the length of the input string. So one invocation of search function of your implementation would take O(4^s) time.

Also in worst case the search is invoked for $m*n$ times. So an upper bound would be $O(m*n*4^{\wedge}s)$.

Space Complexity: O(length of board) + O(length of word) -> size of visited matrix + maximum length of recursive call stack

```
class Solution:
    def exist(self, board: List[List[str]], word: str) -> bool:
        # Edge cases
        if len(word) == 0:
            return False
        if len(board) == 0:
            return False
        if len(word) > len(board) * len(board[0]):
            return False

        def dfs(i,j, visited, board,word,word_position, result):
            if result[0] == True: # Unwind recursion when found success
                return

            if word_position == len(word) - 1:
                result[0] = True  # Found full sequence - Stopping condition
                return

            if visited[i][j] == 1:  # Cannot repeat in one continuous search
                return False



            visited[i][j] = 1

            directions = [[0,1], [0,-1], [1,0], [-1,0]]

            word_position = word_position + 1
```

```
            for direction in directions:

                next_i, next_j = i + direction[0], j + direction[1]

                if 0 <= next_i < len(board) and 0 <= next_j < len(board[0]) a
nd (visited[next_i][next_j] == 0) and (word_position < len(word)) and (word[w
ord_position] == board[next_i][next_j]):
                    # word_position < len(word) should come before word[word_
position] == board[next_i][next_j] comparison (broader comparisons before nar
rower ones)
                    dfs(next_i, next_j, visited, board, word, word_position,
result)
                    visited[next_i][next_j] = 0 # Important: mark unvisited w
hen unwinding



        result = [False] # simple variable problematic in recursive stack
        for row in range(len(board)):
            for col in range(len(board[0])):
                if word[0] == board[row][col]:
                    # Important: every search should start with fresh visited
                    visited = [[0 for _ in range(len(board[0]))] for _ in ran
ge(len(board))]
                    dfs(row, col, visited, board, word, 0, result)
                    if result[0] == False: # Check whether success or not
                        continue
```

```
                    else:
                        break

        return result[0]

# Time Complexity : O(m*n*4^s) where m = # of rows, n = # of columns and s =
length of word
# Space Complexity: O(length of board) + O(length of word) -> size of visited
matrix + maximum length of recursive # call stack
```

---

88. Merge Sorted Array

look solution 2 : with O(1) space complexity

```
# Compare from last and fill in from last

class Solution:
    def merge(self, nums1: List[int], m: int, nums2: List[int], n: int) -> No
ne:
        """
        Do not return anything, modify nums1 in-place instead.
        """
        p1 = m-1
        p2 = n-1
        p = m+n-1

        # while there are still elements to compare
        while p1 >= 0 and p2 >= 0:

            if nums2[p2] > nums1[p1]:
                nums1[p] = nums2[p2]
                p = p-1
                p2 = p2-1

            elif nums2[p2] <= nums1[p1]:
                nums1[p] = nums1[p1]
                p = p-1
                p1 = p1 -1


        # Imp: add missing elements from nums2
        nums1[:p2 + 1] = nums2[:p2 + 1]
```

94. Binary Tree Inorder Traversal

https://www.techiedelight.com/inorder-tree-traversal-iterative-recursive/ (https://www.techiedelight.com/inorder-tree-traversal-iterative-recursive/)

https://www.youtube.com/watch?v=nzmtCFNae9k (https://www.youtube.com/watch?v=nzmtCFNae9k)

```
# Do not put anything in stack before while loop

class Solution:
    def inorderTraversal(self, root: TreeNode) -> List[int]:
        if root is None: # edge case
            return []

        stack = []
        cur = root

        result = []
        while stack or cur: # Imp: or
            if cur:
                stack.append(cur)
                cur = cur.left
            else:
                cur = stack.pop()
                result.append(cur.val)
                cur = cur.right

        return result
```

98. Validate Binary Search Tree

```
# Inorder traversal of binary tree is in ascending order => BST
# keep track of prev value and keep comparing
class Solution:
    def isValidBST(self, root: TreeNode) -> bool:
        if root is None:
            return True

        stack = []
        cur = root
        prev_val = float('-inf')
        while stack or cur:
            if cur:
                stack.append(cur)
                cur = cur.left

            else:
                cur = stack.pop()
                if cur.val <= prev_val: # question mentions < and >, for test
cases it is <= and >=
                    return False
                prev_val = cur.val

                cur = cur.right
        return True
```

100. Same Tree

https://leetcode.com/articles/same-tree/ (https://leetcode.com/articles/same-tree/)

```
# different binary trees can have the same inorder, preorder, or postorder tr
aversal
class Solution:
    def isSameTree(self, p: TreeNode, q: TreeNode) -> bool:

        # p and q are both None
        if p == None and q == None:
            return True
        # one of p and q is None
        if p == None or q == None:
            return False
        # p and q values do not match
        if p.val != q.val:
            return False
        return self.isSameTree(p.right, q.right) and self.isSameTree(p.left,
q.left)

# Time Complexity = O(n)
# Space Complexity = O(n) for recursion stack
```

101. Symmetric Tree

See BFS solution (iterative)

```
class Solution:
    def isSymmetric(self, root: TreeNode) -> bool:
        if root is None: # edge case
            return True

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

        while queue:
            t1 = queue.pop(0)
            t2 = queue.pop(0)

            if t1 is None and t2 is None:
                continue

            if t1 is None or t2 is None:
                return False

            if t1.val != t2.val:
                return False

            queue.append(t1.left)
            queue.append(t2.right)
            queue.append(t1.right)
            queue.append(t2.left)

        return True
```

102. Binary Tree Level Order Traversal

https://medium.com/@timpark0807/leetcode-is-easy-binary-tree-patterns-1-2-6e1793b76415 (https://medium.com/@timpark0807/leetcode-is-easy-binary-tree-patterns-1-2-6e1793b76415)

```
# Visit->Left-> Right
# take care of level, prev_level
# Add new list when encounter new level

class Solution:

    def levelOrder(self, root: TreeNode) -> List[List[int]]:
        if root is None: # edge case
            return []

        queue = []

        queue.append((root,0))

        result = []
        prev_level = -1
        while queue:
            cur, level = queue.pop(0)
            if level != prev_level: # if encounter new level, create new list
                result.append([])


            result[level].append(cur.val)
            if cur.left:
                queue.append((cur.left, level+1))

            if cur.right:
                queue.append((cur.right, level+1))
```

```
            prev_level = level

        return result



# Time Complexity = O(n)
# Space Complexity = O(n) for storing elements in queue
```

---

103. Binary Tree Zigzag Level Order Traversal

https://www.techiedelight.com/spiral-order-traversal-binary-tree/
(https://www.techiedelight.com/spiral-order-traversal-binary-tree/)

```
# [1,2,3,4,null,null,5]: if at a level only single child exists, usual approa
ch of appending in queue in
# opposite way (i.e. right first then left) at alternate levels will fail

# An easier approach is: perform level order traversal as usual and at last r
everse alternate list
from collections import deque
class Solution:
    def zigzagLevelOrder(self, root: TreeNode) -> List[List[int]]:
        if root is None:
            return

        result = []
        q = deque()

        q.append(root)
        flag = True

        while q:
            nodeCount = len(q)
            if flag:
                subresult = []
                while nodeCount > 0:
                    curr = q.popleft() # popleft
                    subresult.append(curr.val)

                    # left then right
                    if curr.left:
                        q.append(curr.left)
```

```
                    q.append(curr.left)
                if curr.right:
                    q.append(curr.right)

                nodeCount = nodeCount - 1
            else:
                subresult = []
                while nodeCount > 0:
                    curr = q.pop() # pop
                    subresult.append(curr.val)

                    # right then left
                    if curr.right:
                        q.appendleft(curr.right)


                    if curr.left:
                        q.appendleft(curr.left)

                    nodeCount = nodeCount - 1

                result.append(subresult)
                flag = not flag

        return result

# Time Complexity = O(n)
# Space Complexity = O(n) for storing elements in queue

# As one can see, at any given moment, the node queue would hold the nodes th
```

```
# As one can see, at any given moment, the node_queue would hold the nodes th
at are at most across two levels.Therefore, at most, the size of the queue wo
uld be no more than 2L, assuming L is the maximum number of nodes that might
reside on the same level. Since we have a binary tree, the level that contain
s the most nodes could occur to consist all the leave nodes in a full binary
tree, which is roughly L = N/2. Therefore, 2*N/2 = N
```

104. Maximum Depth of Binary Tree

https://medium.com/@timpark0807/leetcode-is-easy-binary-tree-patterns-1-2-6e1793b76415
(https://medium.com/@timpark0807/leetcode-is-easy-binary-tree-patterns-1-2-6e1793b76415)

```
class Solution:
    def maxDepth(self, root: TreeNode) -> int:
        if root is None: #edge case
            return 0

        queue = []
        queue.append((root,1))

        max_depth = 0
        while queue:
            cur, level = queue.pop(0)
            max_depth = max(max_depth, level)

            if cur.left:
                queue.append((cur.left, level+1))

            if cur.right:
                queue.append((cur.right, level+1))

        return max_depth

# Time Complexity = O(n)
# Space Complexity = O(n) for storing elements in queue
```

105. Construct Binary Tree from Preorder and Inorder Traversal

https://www.youtube.com/watch?v=PoBGyrIWisE (https://www.youtube.com/watch?v=PoBGyrIWisE)

see comments - https://leetcode.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal/discuss/34579/Python-short-recursive-solution (https://leetcode.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal/discuss/34579/Python-short-recursive-solution)

```
# inorder + postorder or inorder + preorder are both unique identifiers of wh
atever binary tree
# root comes from preorder (DLR)
# left and right subtrees come from inorder (LDR)

from collections import deque
class Solution:
    def buildTree(self, preorder: List[int], inorder: List[int]) -> TreeNode:
        preorder = deque(preorder)
        inorder_map = {num:index for index, num in enumerate(inorder)}

        def build_tree(start, end):
            if start > end: # stop condition for recursion
                return None

            root_index = inorder_map[preorder.popleft()]
            root = TreeNode(inorder[root_index])
            root.left = build_tree(start, root_index-1)
            root.right = build_tree(root_index+1, end)
            return root

        return build_tree(0, len(inorder)-1)

# Time Complexity: deque(preorder) = O(n), inorder_map = O(n), build tree = O
(n); total = O(n)
# Space Complexity: deque = O(n), inorder_map = O(n), build_tree = O(n) to st
ore entire tree; total = O(n)
```

108. Convert Sorted Array to Binary Search Tree

See solution Approach 1

```
# Inorder traversal is not a unique identifier of BST. Several BSTs can have
same inorder traversal(see
# example in solution)
# At the same time both preorder and postorder traversals are unique identifi
ers of BST
# Approach: Always choose Left Middle Node as a Root

class Solution:
    def sortedArrayToBST(self, nums: List[int]) -> TreeNode:
        def helper(left, right):
            if left > right: # exit condition for recursion
                return None

            mid = left + (right-left) //2

            root = TreeNode(nums[mid])
            root.left = helper(left, mid -1)
            root.right = helper(mid+1, right)
            return root

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

# Time Complexity = O(n) since each node is visited once
# Space Complexity = O(n) to construct the BST and O(log n) for recursion sta
ck (height-balanced)
```

110. Balanced Binary Tree

https://www.techiedelight.com/calculate-height-binary-tree-iterative-recursive/
(https://www.techiedelight.com/calculate-height-binary-tree-iterative-recursive/)

```
class Solution:
    def isBalanced(self, root: TreeNode) -> bool:
        if root is None: # edge case: empty tree is balanced by definition
            return True

        def get_height(node):
            if node is None:
                return 0

            left = get_height(node.left)
            right = get_height(node.right)

            if abs(left - right) > 1 or left == -1 or right == -1:  # -1 valu
e being bubbled up in recursion
                return -1
            return max(left, right) + 1



        return get_height(root) != -1


# Time Complexity = O(n)
# Space Complexity = O(n) for recursion stack, if the tree is completely skew
ed towards one side
```

112. Path Sum

```
class Solution:
    def hasPathSum(self, root: TreeNode, sum: int) -> bool:
        if root is None:
            return []

        stack = []
        stack.append((root, sum - root.val))

        while stack:
            node, remains = stack.pop()

            if remains == 0 and node.left is None and node.right is None: # r
emain = 0 only at leaf node
                return True

            if node.left:
                stack.append((node.left, remains - node.left.val))

            if node.right:
                stack.append((node.right, remains - node.right.val))

        return False

# Time Complexity = O(n)
# Space Complexity = O(n)
```

121. Best Time to Buy and Sell Stock

```
# We need to find the largest peak following the smallest valley.
# We can maintain two variables - minprice and maxprofit corresponding to the
smallest valley and
# maximum profit (maximum difference between selling price and minprice) obta
ined so far

# Input can never be negative int
# If the input is in descending order, max profit = 0

import sys
class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        min_price = sys.maxsize
        max_profit = 0

        for i in range(len(prices)):
            if prices[i] < min_price:
                min_price = prices[i]
            elif prices[i] - min_price > max_profit:
                max_profit = prices[i] - min_price

        return max_profit

# Time Complexity = O(n)
# Space Complexity = O(1)
```

122. Best Time to Buy and Sell Stock II

```
# We can simply go on crawling over the slope and keep on adding the profit o
btained from every
# consecutive transaction, if there is any profit i.e. if prices[i] > prices
[i-1]

class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        max_profit = 0

        for i in range(1,len(prices)):
            if prices[i] > prices[i-1]:
                max_profit = max_profit + (prices[i] - prices[i-1])

        return max_profit

# Time Complexity = O(n)
# Space Complexity = O(1)
```

125. Valid Palindrome

```
class Solution:
    def isPalindrome(self, s: str) -> bool:
        if len(s) == 0:
            return True

        begin = 0
        end = len(s) -1

        while begin < end:
            if s[begin].isalnum() and s[end].isalnum():
                if s[begin].lower() != s[end].lower():
                    return False
                else:
                    begin = begin +1
                    end = end - 1

            if s[begin].isalnum() == False:
                begin = begin + 1
            if s[end].isalnum() == False:
                end = end - 1

        return True

# Time Complexity = O(n)
# Space Complexity = O(1)
```

127. Word Ladder

read solution (Approach 1: BFS) https://leetcode.com/problems/word-ladder/solution/ (https://leetcode.com/problems/word-ladder/solution/)

```
# Shortest - BFS
from collections import defaultdict
class Solution:
    def ladderLength(self, beginWord: str, endWord: str, wordList: List[str])
-> int:
        if endWord not in wordList or len(beginWord) == 0 or len(endWord) ==
0 or len(wordList) == 0:
            return 0

        d = defaultdict(list)
        for word in wordList:
            for i in range(len(word)):
                d[word[:i] + "*" + word[i+1:]].append(word)

        visited = []
        q = []
        q.append([beginWord,1])

        while q:
            current_word, level = q.pop(0)

            for i in range(len(word)):
                for word in d[current_word[:i] + "*" + current_word[i+1:]]:
                    if word == endWord:
                        return level+1
                    if word not in visited:
                        visited.append(word)
```

```
                    q.append([word,level+1])
        return 0
```

---

128. Longest Consecutive Sequence

```
# l = ['abc', 'xab', 'hello', 'prat']
# l.sort()
# O/P -> ['abc', 'hello', 'prat', 'xab']

class Solution:
    def longestCommonPrefix(self, strs: List[str]) -> str:
        if not strs: return ""
        if len(strs) == 1: return strs[0]

        strs.sort() # this will sort all strings (by prefix)
        r = ""
        for x,y in zip(strs[0], strs[-1]): # check first and last string char
by char
            if x ==y:
                r = r + x
            else: #only prefix matching
                break
        return r

# Time Complexity = O(nk log nk) where n = no. of words and k = max length of
any word for sorting. There is O(nk) comparison when comparing strs[0] and st
rs[-1] char by char but sorting complexity dominates

# Space Complexity = O(nk)
```

133. Clone Graph

Similar to https://leetcode.com/problems/copy-list-with-random-pointer/ (https://leetcode.com/problems/copy-list-with-random-pointer/)

```
class Solution:
    def cloneGraph(self, node: 'Node') -> 'Node':

        if not node:
            return node

        # Dictionary to save the visited node and it's respective clone
        # as key and value respectively. This helps to avoid cycles.
        visited = {}

        queue = []
        queue.append(node)
        # Clone the node and put it in the visited dictionary.
        visited[node] = Node(node.val, [])

        # Start BFS traversal
        while queue:

            n = queue.pop(0)

            for neighbor in n.neighbors:
                if neighbor not in visited:
                    # Clone the neighbor and put in the visited, if not prese
nt already
                    visited[neighbor] = Node(neighbor.val, [])

                    queue.append(neighbor)
                # Add the clone of the neighbor to the neighbors of the clone
```

```
node "n".
                visited[n].neighbors.append(visited[neighbor]) # Important

        # Return the clone of the node from visited.
        return visited[node]
```

136. Single Number

```
# Method 1: Hash but will have O(n) time and space complexity
# Method 2 : XoR => a xor 0 = a, a xor a = 0 and a xor b xor a = b, space com
plexity will be O(1)
class Solution:
    def singleNumber(self, nums: List[int]) -> int:
        a = 0
        for num in nums:
            a = a ^ num
        return a
```

138. Copy List with Random Pointer

Similar to Clone graph LC:133

https://www.youtube.com/watch?v=OvpKeraoxW0 (https://www.youtube.com/watch?v=OvpKeraoxW0)

```
# deep copy means new copy at new address
# now if the 'pointed to' node does not exist => it's corresponding address a
lso does not exist,
# then how to fill pointer values of current node ??

# Create a visited dictionary to hold old node reference as "key" and new nod
e reference as the
# "value"

class Solution:
    def __init__(self):
        self.visited = {}

    def getClonedNode(self, node):
        if node: # check for null
            if node in self.visited:
                return self.visited[node]
            else:
                self.visited[node] = Node(node.val, None, None) # create new
node and update dict
                return self.visited[node]
        return None

    def copyRandomList(self, head: 'Node') -> 'Node':
        if not head:
            return head

        old_node = head
```

```
        new_node = Node(old_node.val, None, None) # create first node
        self.visited[old_node] = new_node # update dict

        while old_node:
            new_node.random = self.getClonedNode(old_node.random)
            new_node.next = self.getClonedNode(old_node.next)

            old_node = old_node.next
            new_node = new_node.next

        return self.visited[head]

# Time Complexity = O(n) - one pass over original linked list
# Space Complexity = O(n) - create hash table for keeping track of mapping
```

---

139. Word Break

```
# Given a starting index, loop thru entire string till end to find all substr
ings that are part of # word dict, add the ending index (exclusive) to queue
and repeat from the ending index so obtained # till end of string
# At any point, this ending index (exclusive) becomes equal to length of stri
ng -> Success

# Keep track of index in string from which searching till end has already hap
pened to avoid
# duplication in search e.g. "aaaaaaaaaaaaaaa" ['a', 'aa', 'aaa'] else TLE (l
evel+1 contains
# starting index where level has already done searching till end)

class Solution:
    def wordBreak(self, s: str, wordDict: List[str]) -> bool:
        wordDict = set(wordDict)

        queue = []
        visited = []

        queue.append(0)
        while queue:
            start_index = queue.pop(0)
            if start_index not in visited:
                for end_index in range(start_index,len(s)+1):
                    if s[start_index:end_index] in wordDict:
                        queue.append(end_index)
                        if end_index == len(s):
                            return True
```

```
                visited.append(start_index)

        return False

# Time Complexity = O(n^2) as for every starting index, we loop thru entire s
tring to find all substrings that is contained in dict
# Space Complexity = O(n) queue to store ending indexes, O(n) visited array
```

---

141. Linked List Cycle

Another approach: Reverse a linked list, if you get the same head that means the linked List has a cycle otherwise it doesn't

```
# slow pointer moves 1 step, fast pointer moves 2 steps
# If they meet = cycle
# if fast pointer reaches end (both for odd length and even length list) = no
cycle

class Solution:
    def hasCycle(self, head: ListNode) -> bool:
        if head == None or head.next == None:  # Edge case -> no node or only
1 node
            return False

        slow = head
        fast = head.next

        while(slow != fast):
            if fast == None or fast.next == None: # length of list could be o
dd or even
                return False

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

        return True

# Time Complexity = O(n)
# Space Complexity = O(1)
```

144. Binary Tree Preorder Traversal

https://www.techiedelight.com/preorder-tree-traversal-iterative-recursive/ (https://www.techiedelight.com/preorder-tree-traversal-iterative-recursive/)
https://www.youtube.com/watch?v=elQcrJrfObg (https://www.youtube.com/watch?v=elQcrJrfObg)

```
class Solution:
    def preorderTraversal(self, root: TreeNode) -> List[int]:
        if root is None: #edge case
            return []

        stack = []
        stack.append(root)

        result = []
        while stack:
            cur = stack.pop()

            result.append(cur.val)

            # Stack is LIFO hence first right then left
            if cur.right:
                stack.append(cur.right)

            if cur.left:
                stack.append(cur.left)

        return result
```

145. Binary Tree Postorder Traversal

https://www.techiedelight.com/postorder-tree-traversal-iterative-recursive/ (https://www.techiedelight.com/postorder-tree-traversal-iterative-recursive/)
https://www.youtube.com/watch?v=qT65HltK2uE (https://www.youtube.com/watch?v=qT65HltK2uE)

```
# Iterative Preorder Traversal: Tweak the Order of the Output
# Two stacks (one to reverse at last)
# DLR in implementation (LRD in theory) + Reverse
class Solution:
    def postorderTraversal(self, root: TreeNode) -> List[int]:
        if root is None: # edge case
            return []

        stack = []
        stack.append(root)

        result = []
        while stack:
            cur = stack.pop()
            result.append(cur.val)

            if cur.left:
                stack.append(cur.left)

            if cur.right:
                stack.append(cur.right)

        return result[::-1]
```

146. LRU Cache

the put() func should check capacity first, if capacity is reached, it should delete least recently used first before adding the new item. However, if you do this, one test case fails (which I think is wrong in LC) TC: ["LRUCache","get","put","get","put","put","get","get"] [[2],[2],[2,6],[1],[1,5],[1,2],[1],[2]] LC ans: [null,-1,null,-1,null,null,2,6] -> has three items whereas capacity is just 2 Actual ans: [null,-1,null,-1,null,null,2,-1]

---

152. Maximum Product Subarray

https://www.youtube.com/watch?v=vtJvbRlHqTA (https://www.youtube.com/watch?v=vtJvbRlHqTA)

e.g. [4,-2,-3]

```
# E.g. -1,6,2,-2 => max product is 24, it can be seen why we want to keep tra
ck of prev_min_prod

class Solution:
    def maxProduct(self, nums: List[int]) -> int:
        curr_max_prod = nums[0]
        curr_min_prod = nums[0]
        result = nums[0]
        prev_max_prod = nums[0]
        prev_min_prod = nums[0]

        for i in range(1, len(nums)):
            curr_max_prod = max(prev_max_prod * nums[i], prev_min_prod * nums
[i], nums[i])
            curr_min_prod = min(prev_max_prod * nums[i], prev_min_prod * nums
[i], nums[i])
            result = max(result, curr_max_prod)
            prev_max_prod = curr_max_prod
            prev_min_prod = curr_min_prod

        return result

# Time Complexity = O(n)
# Space Complexity = O(1)
```

153. Find Minimum in Rotated Sorted Array

```
# No duplicates
# This question is equivalent to finding the index where rotation happens
# At any point if mid element is greater than mid+1 element -> mid+1 is the r
otation index
# Compare mid to end and adjust begin and end pointers accordingly

class Solution:
    def findMin(self, nums: List[int]) -> int:
        begin = 0
        end = len(nums) - 1

        if len(nums) == 1: # edge case - only one element
            return nums[0]

        if nums[begin] < nums[end]: # edge case - if no rotation
            return nums[begin]

        while begin <= end:
            mid = begin + (end-begin) // 2

            if nums[mid] > nums[mid+1]: # Found condition
                return nums[mid+1]

            if nums[mid] > nums[end]:
                begin = mid + 1
            elif nums[mid] < nums[end]:
                end = mid # Imp: not mid + 1 e.g. [4,5,1,2,3]
```

```
# Time Complexity = O(log n)
# Space Complexity = O(1)
```

---

157. Read N Characters Given Read4

```
# readn using read4 and return the number of chars read + fill buf with the c
haracters read
# two cases: len(file) >= 4 and len(file) < 4
# Also note that buf4 (and buf) are populated after passing them as arg in fn
calls

class Solution:
    def read(self, buf, n):
        buf4 = [''] * 4 # needs to be initialized
        copied_chars = 0

        chars_read = 0

        while copied_chars < n:
            chars_read = read4(buf4)
            if chars_read == 0: # e.g. file = "abc" and n = 4
                return copied_chars

            for i in range(chars_read):
                if copied_chars == n:
                    return copied_chars
                buf[copied_chars] = buf4[i]
                copied_chars = copied_chars + 1

        return copied_chars

# Time Complexity = O(n) to copy n chars
```

```
# Space Complexity = O(1), buf is given and buf4 is always of 4 chars indepen
dent of input
```

---

160. Intersection of Two Linked Lists

Core idea :

If tails of both list are

1) different = no intersection 2) same = intersection

To find the intersection node: a = length of list A b = length of list B k = difference of lengths = abs(a-b) if this difference is maintained and traversal happens one at a time for both lists: they meet = intersection

To maintain k difference between lists: When pA reaches the end of a list, then redirect it to the head of B (yes, B, that's right.); similarly when pB reaches the end of a list, redirect it the head of A.

https://leetcode.com/problems/intersection-of-two-linked-lists/discuss/49798/Concise-python-code-with-comments (https://leetcode.com/problems/intersection-of-two-linked-lists/discuss/49798/Concise-python-code-with-comments)

```
class Solution:
    def getIntersectionNode(self, headA: ListNode, headB: ListNode) -> ListNo
de:
        a = headA
        b = headB

        while a is not b:
            if a is not None:
                a = a.next
            else:
                a = headB

            if b is not None:
                b = b.next
            else:
                b = headA

        return a

# if they didn't meet, they will hit the end at the same iteration, a == b ==
None (since a and b would have traversed (len a + len b) nodes)
```

167. Two Sum II - Input array is sorted

We use two indexes, initially pointing to the first and last element respectively. Compare the sum of these two elements with target. If the sum is equal to target, we found the exactly only solution. If it is less than target, we increase the smaller index by one. If it is greater than target, we decrease the larger index by one. Move the indexes and repeat the comparison until the solution is found.

```
# Two pointer approach since input array is sorted
# Hash table can be used but will have time complexity = O(n), we can do bett
er than than using two  # pointer approach

class Solution:
    def twoSum(self, numbers: List[int], target: int) -> List[int]:
        low = 0
        high = len(numbers)-1

        result = []

        while low<high:
            total = numbers[low] + numbers[high]

            if total < target:
                low = low+1

            elif total > target:
                high = high-1

            else: # only one unique solution and not zero-indexed
                result.append(low+1)
                result.append(high+1)
                return result

# Time Complexity = O(n)
# Space Complexity = O(1)
```

169. Majority Element

See Approach 6

189. Rotate Array

```
# edge case [1, 2,3] k = 4 => k = 1 (as 3 rotations will bring array back to
its original position)

class Solution:
    def rotate(self, nums: List[int], k: int) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """
        def reverse(nums,start,end):
            while(start<=end):
                temp = nums[end]
                nums[end] = nums[start]
                nums[start] = temp
                start = start + 1
                end = end - 1

        k = k % len(nums) # edge case
        reverse(nums,0,len(nums)-1)
        reverse(nums,0,k-1)
        reverse(nums,k,len(nums)-1)

# Time Complexity = O(n)
# Space Complexity = O(1)
```

199. Binary Tree Right Side View

See Approach 3: BFS + level size

```
# Put all children of a node, left to right in queue at a level
# Now the last element of queue is the right side view

class Solution:
    def rightSideView(self, root: TreeNode) -> List[int]:
        if root is None: # edge case
            return []

        queue = []
        queue.append(root)

        right_view = []

        while queue:
            level_length = len(queue)

            for i in range(level_length):
                cur = queue.pop(0)

                if i == level_length - 1:
                    right_view.append(cur.val)

                if cur.left:
                    queue.append(cur.left)

                if cur.right:
                    queue.append(cur.right)
```

```
        return right_view

# Time Complexity = O(n)
# Space Complexity = O(n)
```

---

200. Number of Islands

```
class Solution:
    def numIslands(self, grid: List[List[str]]) -> int:
        row = len(grid)
        if len(grid) == 0:
            return 0
        col = len(grid[0])
        visited = [[0 for _ in range(col)] for _ in range(row)]



        def dfs(i,j):
            if visited[i][j] == 1:
                return
            visited[i][j] = 1
            directions = [[0,1], [1,0], [0,-1], [-1,0]]
            for direction in directions:
                next_i, next_j = i + direction[0], j + direction[1]
                if 0 <= next_i < len(grid) and 0 <= next_j <len(grid[0]) and
grid[next_i][next_j] == "1":
                    dfs(next_i, next_j)

        num_of_island = 0
        for i in range(len(grid)):
            for j in range(len(grid[0])):
                if grid[i][j] == "1" and visited[i][j] == 0:
                    num_of_island = num_of_island + 1
                    dfs(i,j)
        return num_of_island
```

```
# Time Complexity = O(m*n) where m = # of rows and n = # of cols
# Space Complexity = O(m*n) visited array
```

---

202. Happy Number

```
# Two parts:
# 1. Given a number n, what is its next number?
# 2. Follow a chain of numbers and detect if we've entered a cycle.

class Solution:
    def isHappy(self, n: int) -> bool:
        def get_next(n): # Time Complexity = O(log n)
            sum_sqr = 0
            while n > 0:
                n, digit = divmod(n,10) # extracting each digit
                sum_sqr = sum_sqr + digit ** 2

            return sum_sqr

        slow = n
        fast = get_next(n)

        while slow != fast and fast != 1: # break when cycle or fast becomes
1 -> any 1 cond true
            slow = get_next(slow)
            fast = get_next(get_next(fast))

        return fast == 1

# number of digits in a number N is floor(log10(N)) + 1 e.g. 100 has 3 digits
=> log 100 + 1 = 3
# Finding the next value for a given number has a cost of O(logn) because we
are processing each digit in the number, and the number of digits in a number
```

```
is given by log n.


# Time Complexity = O (log n), 2 cases: 1) fast becomes 1  2) slow catches up
with fast
# case 1: if there is no cycle, fast reaches 1 and slow will be halfway to 1
=> O(2 log n) = O(log n)
# case 2: if there is cycle, fast will get one step closer to slow at each cy
cle. Imagine there are k numbers in the cycle. If they started at k - 1 place
s apart (which is the furthest apart they can start), then it will take k - 1
steps for the fast runner to reach the slow runner, which again is constant f
or our purposes. Therefore, the dominating operation is still calculating the
next value for the starting n, which is O(log n).

# Space Complexity = O(1)
```

206. Reverse Linked List

```
# Three pointers are needed
# 4 step process

# Order of operations
class Solution:
    def reverseList(self, head: ListNode) -> ListNode:
        prev = None

        while head:
            curr = head
            head = head.next
            curr.next = prev
            prev = curr
        return prev

# Time Complexity = O(n)
# Space Complexity = O(1)
```

---

207. Course Schedule

https://www.geeksforgeeks.org/detect-cycle-direct-graph-using-colors/ (https://www.geeksforgeeks.org/detect-cycle-direct-graph-using-colors/)
https://github.com/harishvc/challenges/blob/master/graph-detect-cycles-DFS.py (https://github.com/harishvc/challenges/blob/master/graph-detect-cycles-DFS.py)

Use the following approach: consider we have three colors, and each vertex should be painted with one of these colors. "White color" means that the vertex hasn't been visited yet. "Gray" means that we've visited the vertex but haven't visited all vertices in its subtree. "Black" means we've visited all vertices in subtree and left the vertex. So, initially all vertices are white. When we visit the vertex, we should paint it gray. When we leave the vertex (that is we are at the end of the dfs() function, after going throung all edges from the vertex), we should paint it black. If you use that approach, you just need to change dfs() a little bit. Assume we are going to walk through the edge v->u. If u is white, go there. If u is black, don't do anything. If u is gray, you've found the cycle because you haven't left u yet (it's gray, not black), but you come there one more time after walking throung some path.

```
class Solution:
    def canFinish(self, numCourses: int, prerequisites: List[List[int]]) -> b
ool:
        color_dict = {x:"WHITE" for x in range(numCourses)}
        #instead of visited -> 2 states, color -> 3 states is used which cont
ains visited's 2 states
        adj_list = {x:[] for x in range(numCourses)}
        for x,y in prerequisites:
            adj_list[y].append(x)

        found_cycle = [False]  # issue with global when using recursive stac
k, use list



        def dfs(start_node, adj_list, color_dict, found_cycle):

            if found_cycle[0] == True:
                return # Unwrapping recursion when condition is met

            color_dict[start_node] = "GRAY"

            for neighbor in adj_list[start_node]:
                if color_dict[neighbor] == "GRAY":
                    found_cycle[0] = True
                    return # Unwrapping recursion when condition is met
                if color_dict[neighbor] == "WHITE" and dfs(neighbor, adj_lis
t, color_dict, found_cycle) == True:
                    return True  # above 'and' is important with return value
```

```
true


            color_dict[start_node] = "BLACK"



        for start_node in range(numCourses):     # need to iterate on all nod
es in case disconnected graph
            if color_dict[start_node] == "WHITE":
                dfs(start_node, adj_list, color_dict, found_cycle)
            if found_cycle[0] == True:
                break
        return not found_cycle[0] # Need to return complement value when cycl
e is found i.e. False
```

208. Implement Trie (Prefix Tree)

https://medium.com/basecs/trying-to-understand-tries-3ec6bede0014#:~:text=A%20trie%20is%20a%20tree,are%20a%20relatively%20new%20thing (https://medium.com/basecs/trying-to-understand-tries-3ec6bede0014#:~:text=A%20trie%20is%20a%20tree,are%20a%20relatively%20new%20thing).

https://leetcode.com/problems/implement-trie-prefix-tree/discuss/58834/AC-Python-Solution
(https://leetcode.com/problems/implement-trie-prefix-tree/discuss/58834/AC-Python-Solution)

```
class Trie:
    def __init__(self):
        self.root = TrieNode()

    def insert(self, word: str) -> None:
        current = self.root
        for letter in word:
            current = current.children[letter]
        current.is_word = True # set this flag to indicate that it is a valid
word

    def search(self, word: str) -> bool:
        current = self.root
        for letter in word:
            current = current.children.get(letter)
            if current is None: # if the word mismatches with traversal then
False
                return False
        return current.is_word # if the word is longer than what we traversed
upto current then False

    def startsWith(self, prefix: str) -> bool:
        current = self.root
        for letter in prefix:
            current = current.children.get(letter)
            if current is None:
                return False
        return True # here return True instead of is_word value
```

```
# Time Complexity of insert, search and delete for trie = O(m) wher m is leng
th of word
# Space Complexity of insert = O(m)
# Space Complexity of search and delete = O(1)
```

---

210. Course Schedule II

Topological sort cannot be applied if graph has cycles In Topological Sort, the idea is to visit the parent node followed by the child node. If the given graph contains a cycle, then there is at least one node which is a parent as well as a child so this will break Topological Order.

```
# Topological sort  - cannot be done if graph has cycles

# A topological sort or topological ordering of a directed graph is a linear
ordering of its vertices such that
# for every directed edge uv from vertex u to vertex v, u comes before v in t
he ordering. In a circle, through one # path u comes v and through another pa
th v comes before u

# if no cycles and not disconnected, topological sort covers all vertices (de
pending on what edge it chooses to
# go depth-wise -> can have multiple topologocal sort sequence)

class Solution:
    def canFinish(self, numCourses: int, prerequisites: List[List[int]]) -> b
ool:
        color_dict = {x:"WHITE" for x in range(numCourses)}
        #instead of visited -> 2 states, color -> 3 states is used which cont
ains visited's 2 states
        adj_list = {x:[] for x in range(numCourses)}
        for x,y in prerequisites:
            adj_list[y].append(x)

        found_cycle = [False]  # issue with global when using recursive stac
k, use list


        def dfs(start_node, adj_list, color_dict, found_cycle):
```

```
            if found_cycle[0] == True:
                return # Unwrapping recursion when condition is met

            color_dict[start_node] = "GRAY"

            for neighbor in adj_list[start_node]:
                if color_dict[neighbor] == "GRAY":
                    found_cycle[0] = True
                    return # Unwrapping recursion when condition is met
                if color_dict[neighbor] == "WHITE" and dfs(neighbor, adj_lis
t, color_dict, found_cycle) == True:
                    return True  # above 'and' is important with return value
true as when no cycle -> recursion/depth first search should continue till th
e end and backtrack safely


            color_dict[start_node] = "BLACK" # Fully explored this node, all
descendants visited

        for start_node in range(numCourses):     # need to iterate on all nod
es in case disconnected graph
            if color_dict[start_node] == "WHITE":
                dfs(start_node, adj_list, color_dict, found_cycle)
            if found_cycle[0] == True:
                break
        return not found_cycle[0] # Need to return complement value when cycl
e is found i.e. False
```

```
    def findOrder(self, numCourses: int, prerequisites: List[List[int]]) -> L
ist[int]:
        cycle_absent = self.canFinish(numCourses, prerequisites)
        if cycle_absent == False:
            return []

        visited = [0 for _ in range(numCourses)]
        adj_list = {x:[] for x in range(numCourses)}
        for x,y in prerequisites:
            adj_list[y].append(x)

        topsort = []
        def toposort(start_node, adj_list, visited, topsort):
            if visited[start_node] == 1:
                return
            visited[start_node] = 1
            for neighbor in adj_list[start_node]:
                toposort(neighbor, adj_list, visited, topsort)

            topsort.append(start_node) # only when you backtrack in dfs i.e.
when you have explored all neighbors of start_node

        for start_node in range(numCourses):
            if visited[start_node] == 0:
                toposort(start_node, adj_list, visited, topsort)
        return topsort[::-1]  # print in reverse order
```

211. Design Add and Search Words Data Structure

https://leetcode.com/problems/implement-trie-prefix-tree/ (https://leetcode.com/problems/implement-trie-prefix-tree/)

https://leetcode.com/problems/add-and-search-word-data-structure-design/discuss/59725/Python-easy-to-follow-solution-using-Trie (https://leetcode.com/problems/add-and-search-word-data-structure-design/discuss/59725/Python-easy-to-follow-solution-using-Trie).

```
from collections import defaultdict
class TrieNode:
    def __init__(self):
        self.children = defaultdict(TrieNode)
        self.is_word = False

class WordDictionary:

    def __init__(self):
        self.root = TrieNode()


    def addWord(self, word: str) -> None:
        curr = self.root
        for letter in word:
            curr = curr.children[letter]
        curr.is_word = True

    def search(self, word: str) -> bool:
        self.result = False # Imp: note use of this flag
        self.dfs(self.root, word)
        return self.result

    def dfs(self, node, word):
        if len(word) == 0 : # exhausted all the letters in the word
            if node.is_word == True:
                self.result = True
            return
```

```
        if word[0] == ".": # if letter is ., move ahead one node in trie and
next letter in word
            for node in node.children.values():
                self.dfs(node, word[1:])
        else:
            node = node.children.get(word[0])
            if node is None: # next letter in word is not found in trie
                return
            self.dfs(node, word[1:])


# Time Complexity of search = O(m) where m is length of word
# Space Complexity of search = O(1)

# Time and Space Complexity of addWord = O(m)
```

215. Kth Largest Element in an Array

https://www.youtube.com/watch?v=4hkJBcW5Ruk (https://www.youtube.com/watch?v=4hkJBcW5Ruk)

https://runestone.academy/runestone/books/published/pythonds/Trees/BinaryHeapImplementation.html (https://runestone.academy/runestone/books/published/pythonds/Trees/BinaryHeapImplementation.html)

```
import heapq
class Solution:
    def findKthLargest(self, nums: List[int], k: int) -> int:
        heap = []
        for num in nums:
            heapq.heappush(heap, num)
            if len(heap) > k:
                heapq.heappop(heap)
        return heap[0] # Min heap of size k, heap[0] = kth largest element

# Time Complexity = O(N log k), adding to heap is log(size of heap)
# Space Complexity = O(k)
```

---

226. Invert Binary Tree

```
class Solution:
    def invertTree(self, root: TreeNode) -> TreeNode:
        if root is None:
            return None

        queue = []
        queue.append(root)

        temp = 0
        while queue:
            cur = queue.pop(0)

            temp = cur.left
            cur.left = cur.right
            cur.right = temp

            if cur.left:
                queue.append(cur.left)

            if cur.right:
                queue.append(cur.right)

        return root

# Time Complexity = O(n)
# Space Complexity = O(n)
```

230. Kth Smallest Element in a BST

```
class Solution:
    def kthSmallest(self, root: TreeNode, k: int) -> int:
        if root is None: # edge case
            return []

        stack = []
        cur = root

        result = []
        while stack or cur: # Imp: or, at the beginning stack is empty
            if cur:
                stack.append(cur)
                cur = cur.left
            else:
                cur = stack.pop()
                k = k - 1
                if k == 0:
                    return cur.val
                cur = cur.right
```

235. Lowest Common Ancestor of a Binary Search Tree

See solution

```
# Exploit the property of BST
class Solution:
    def lowestCommonAncestor(self, root: 'TreeNode', p: 'TreeNode', q: 'TreeN
ode') -> 'TreeNode':
        node = root

        while node:

            node_val = node.val

            if p.val > node_val and q.val > node_val:
                node = node.right
            elif p.val < node_val and q.val < node_val:
                node = node.left
            else:
                return node

# Time Complexity = O(n)
# Space Complexity = O(1)
```

236. Lowest Common Ancestor of a Binary Tree

```
class Solution:
    def lowestCommonAncestor(self, root: 'TreeNode', p: 'TreeNode', q: 'TreeN
ode') -> 'TreeNode':
        stack = []
        stack.append(root)

        # Dictionary for parent pointers
        parent = {root: None}

        # Iterate until we find both the nodes p and q
        while p not in parent or q not in parent:
            node = stack.pop()

            if node.left:
                parent[node.left] = node
                stack.append(node.left)

            if node.right:
                parent[node.right] = node
                stack.append(node.right)

        # Ancestors set() for node p
        ancestors = set()
        # Process all ancestors for node p using parent pointers
        while p:
            ancestors.add(p)
            p = parent[p]
```

```
        # The first ancestor of q which appears in p's ancestor set() is thei
r lowest common ancestor
        while q not in ancestors:
            q = parent[q]

        return q

# Time Complexity = O(n)
# Space Complexity = O(n) for stack + O(n) for parent pointers dict + O(n) fo
r ancestor's set of p
```

---

237. Delete Node in a Linked List

```
# Replace current node with next node (both val and next)

class Solution:
    def deleteNode(self, node):
        """
        :type node: ListNode
        :rtype: void Do not return anything, modify node in-place instead.
        """
        node.val = node.next.val
        node.next = node.next.next
```

238. Product of Array Except Self

https://leetcode.com/problems/product-of-array-except-self/solution/
(https://leetcode.com/problems/product-of-array-except-self/solution/)

```
class Solution:
    def productExceptSelf(self, nums: List[int]) -> List[int]:
        L = [0] * len(nums)
        L[0] = 1
        for i in range(1,len(nums)):
            L[i] = L[i-1] * nums[i-1]

        R = [0] * len(nums)
        R[len(nums)-1] = 1
        for i in reversed(range(len(nums)-1)):
            R[i] = R[i+1] * nums[i+1]

        result = [0] * len(nums)
        for i in range(len(nums)):
            result[i] = L[i] * R[i]

        return result
```

240. Search a 2D Matrix II

See Approach 4: Search Space Reduction

```
# Start from bottom-left
# if target > current element then increment col index
# if target < current element then decrement row index
# search till row and column remain within bounds else not found

class Solution:
    def searchMatrix(self, matrix, target):
        """
        :type matrix: List[List[int]]
        :type target: int
        :rtype: bool
        """
        row = len(matrix) - 1
        col = 0

        while row >= 0 and col < len(matrix[0]):
            if matrix[row][col] == target:
                return True

            if target > matrix[row][col]:
                col = col + 1

            elif target < matrix[row][col]:
                row = row - 1

        return False

# Time Complexity = O(m+n), note that row or col is decremented/incremented a
```

```
t every iteration.
# Now row cannot be decremented more than number of rows and column cannot be
incremented more
# than number of columns hence loop cannot run for more than m+n times

# Space Complexity = O(1)
```

---

242. Valid Anagram

```
from collections import defaultdict

class Solution:
    def isAnagram(self, s: str, t: str) -> bool:
        d = defaultdict(int)

        for c in s:
            d[c] = d[c] + 1

        for c in t:
            d[c] = d[c] - 1

        for key in d.keys():
            if d[key] != 0:
                return False

        return True

# Time Complexity = O(n)
# Space Complexity = O(1)
```

249. Group Shifted Strings

https://leetcode.com/problems/group-shifted-strings/discuss/282285/Python-Solution-with-Explanation-(44ms-84) (https://leetcode.com/problems/group-shifted-strings/discuss/282285/Python-Solution-with-Explanation-(44ms-84))

```
# map each string in strings to a key in a hashmap
# this key is ord(i+1) - ord(i)
# hence the hash table with key: tuple and value = list of string
# for case such as az and ba to be clubbed together: z-a = 25 and a-b = -1, a
dd +26 and take mod of 26
# (26+25) % 26 => 25 and (26+1-2) %26 => 25

from collections import defaultdict
class Solution:
    def groupStrings(self, strings: List[str]) -> List[List[str]]:
        d = defaultdict(list)
        for s in strings:
            key = ()
            for i in range(len(s)-1): # until second last since i+1 below
                circular_diff = 26 + ord(s[i+1]) - ord(s[i])
                key = key + (circular_diff % 26,) # concat tuple to tuple

            d[key].append(s)

        return d.values()

# Time complexity would be O(ab) where a is the total number of strings and b
is the length of the longest string in strings.
# Space complexity would be O(ab), as the most space we would use is the spac
e required for strings and the keys of our hashmap.
```

252. Meeting Rooms

```
class Solution:
    def canAttendMeetings(self, intervals: List[List[int]]) -> bool:
        intervals.sort(key= lambda x: x[0])

        i = 0
        while i < len(intervals) - 1:
            if intervals[i][1] > intervals[i+1][0]:
                return False
            else:
                i = i + 1
        return True
```

253. Meeting Rooms II

See Solution 2: Chronological ordering

```
class Solution:
    def minMeetingRooms(self, intervals: List[List[int]]) -> int:
        start = [interval[0] for interval in intervals]
        end = [interval[1] for interval in intervals]

        start.sort()
        end.sort()

        start_ptr = 0
        end_ptr = 0

        used_rooms = len(start) # max no. of rooms = total no of meetings
        while start_ptr < len(start):
            if start[start_ptr] >= end[end_ptr]:  # if start time of any meet
ing >= end time of any meeting -> room free
                used_rooms = used_rooms - 1
                end_ptr = end_ptr + 1
                start_ptr = start_ptr + 1
            else:
                start_ptr = start_ptr + 1
        return used_rooms

# Time Complexity = O(n log n) for sorting
# Space Complexity = O(n), for start array O(n) and end array O(n)
```

257. Binary Tree Paths

See solution: Approach 2 Iterations

class Solution: def binaryTreePaths(self, root: TreeNode) -> List[str]: if root is None: # edge case return []

```
    stack = []
    stack.append((root,str(root.val)))

    paths = []
    while stack:
        node, path = stack.pop()
        if node.left is None and node.right is None:
            paths.append(path)

        if node.left:
            stack.append((node.left, path + '->' + str(node.left.val)))

        if node.right:
            stack.append((node.right, path + '->' + str(node.right.val)))

    return paths
```

261. Graph Valid Tree

https://leetcode.com/problems/graph-valid-tree/solution/ (https://leetcode.com/problems/graph-valid-tree/solution/) See Approach 2

```
# No duplicate edges + Undirected graph

# For a graph to be tree - 1) No cycles 2) Fully connected
# For an undirected graph with no duplicate edges, if it has n-1 edges => no
cycle
# Checking whether or not a graph is fully connected is straightforward - we
simply check if all
# nodes are reachable from a search starting at a single node => BFS or DFS
class Solution:
    def validTree(self, n: int, edges: List[List[int]]) -> bool:

        if len(edges) != n-1: # check cycle
            return False

        visited = [0 for _ in range(n)]
        g = {x :[] for x in range(n)}
        for x,y in edges:
            g[x].append(y)
            g[y].append(x)


        def dfs(node, g, visited):
            if visited[node]:
                return
            visited[node] = 1

            for neighbor in g[node]:
                dfs(neighbor, g, visited)
```

```
        dfs(0,g,visited)
        counter = 0
        # counter used within dfs function will not be available outside dfs
function as dfs has no return value + global variable issue for recursive fns
=> so count visited nodes here
        for i in range(n):
            if visited[i] == 1:
                counter = counter + 1

        if counter != n: # check fully connected
            return False
        return True
```

---

266. Palindrome Permutation

```
# All chars should have even freq
# Only one char is allowed to have odd freq

from collections import defaultdict
class Solution:
    def canPermutePalindrome(self, s: str) -> bool:
        d = defaultdict(int)
        for c in s:
            d[c] = d[c] + 1

        count = 0
        for c in d.keys():
            if d[c] % 2 == 0:
                continue
            if d[c] % 2 == 1:
                count = count + 1

        return count <=1

# Time Complexity = O(n)
# Space Complexity = O(1) since keys can be atmost 26 (constant)
```

268. Missing Number

```
class Solution:
    def missingNumber(self, nums: List[int]) -> int:
        total = 0
        maximum = 0


        for num in nums:
            total = total + num

        cumulative_sum = len(nums) * (len(nums)+1) //2

        missing_num = cumulative_sum - total
        return missing_num
```

269. Alien Dictionary

https://leetcode.com/problems/alien-dictionary/discuss/156130/Python-Solution-with-Detailed-Explanation-(91) (https://leetcode.com/problems/alien-dictionary/discuss/156130/Python-Solution-with-Detailed-Explanation-(91))

270. Closest Binary Search Tree Value

```
# Binary search tree : left subtree <= node <= right subtree
# go left if target is smaller than current root value, and go right otherwis
e
# keep track of closest value at each step

class Solution:
    def closestValue(self, root: TreeNode, target: float) -> int:

        closest = root.val

        while root:
            closest = min(root.val, closest, key = lambda x: abs(target -x ))

            if target < root.val:
                root = root.left
            else:
                root = root.right
        return closest

# Time Complexity = O(height of tree)
# Space Complexity = O(1)
```

278. First Bad Version

```
class Solution:
    def firstBadVersion(self, n):
        """
        :type n: int
        :rtype: int
        """
        low = 1
        high = n

        while low <= high:
            pivot = low + (high-low) // 2

            if low == high:
                return low

            if isBadVersion(pivot):
                high = pivot
            else:
                low = pivot + 1

# Time Complexity = O(log n)
# Space Complexity = O(1)
```

279. Perfect Squares

See approach 4 in solution : BFS

```
class Solution:
    def numSquares(self, n: int) -> int:
        # list of square numbers that are less than `n`
        square_nums = [i * i for i in range(1, int(n**0.5)+1)] # +1 since ran
ge excludes last element


        level = 0
        queue = {n}
        while queue:
            level += 1
            # use set() instead of list() for speedup to avoid duplicates at
same level
            # though, list works too (except TLE error in leetcode)
            next_queue = set()
            # construct the queue for the next level
            for remainder in queue:
                if remainder in square_nums:
                    return level  # find the node!
                else:
                    remainders = [remainder - square_num for square_num in sq
uare_nums]
                    next_queue.update(remainders) # adding list to set happen
s via update
            queue = next_queue

# here queue is set (hash or dict) and dict size cannot be changed during ite
```

```
ration
# if same queue is used, line15: queue size changes during iteration and henc
e error is thrown
# if the queue is dict/hash and at every level its size is changing, use anot
her queue which is also a set # to loop around

# Time Complexity = O(n^h/2)
# e.g. n = 13, # of nums whose square is less than or equal to 13 => 1,2,3 =>
~ 13^1/2 (n^1/2)
# At every level h, (n^1/2)*h numbers are present

# Space Complexity = O(n^h/2)
```

---

283. Move Zeroes

https://www.youtube.com/watch?v=XWaVlWRSDx8 (https://www.youtube.com/watch?v=XWaVlWRSDx8)

```
# maintain relative order of elements - tricky part

class Solution:
    def moveZeroes(self, nums: List[int]) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """
        for i in reversed(range(len(nums))): # imp: reverse traversal
            if nums[i] == 0:
                k = i
                while k < len(nums)-1:
                    nums[k], nums[k+1] = nums[k+1], nums[k]
                    k = k+1
```

295. Find Median from Data Stream

https://leetcode.com/problems/find-median-from-data-stream/discuss/74158/Python-O(lgn)-using-two-heapq-data-sturctures (https://leetcode.com/problems/find-median-from-data-stream/discuss/74158/Python-O(lgn)-using-two-heapq-data-sturctures)

Approach 3: Two heaps https://leetcode.com/problems/find-median-from-data-stream/solution/ (https://leetcode.com/problems/find-median-from-data-stream/solution/)

See Further thoughts section in solutions (especially reservoir sampling one)

```
import heapq
class MedianFinder:

    def __init__(self):
        """
        initialize your data structure here.
        """
        self.small = []
        self.large = []

    def addNum(self, num: int) -> None:
        # when self.small is empty
        if len(self.small) == 0:
            heapq.heappush(self.small, -num) # max heap hence push -num
            return

        # if incoming num is smaller than self.small[0] max heap
        if num <= -self.small[0]:
            heapq.heappush(self.small, -num)
        else:
            heapq.heappush(self.large, num)

        # balancing
        if len(self.small) - len(self.large) == 2:
            heapq.heappush(self.large, -heapq.heappop(self.small))
        elif len(self.small) - len(self.large) == -2:
            heapq.heappush(self.small, -heapq.heappop(self.large))
```

```
    def findMedian(self) -> float:
        if len(self.small) == len(self.large):
            return (self.large[0] + (- self.small[0]))/2.0
        if len(self.small) > len(self.large):
            return -self.small[0]
        else:
            return self.large[0]

# Time Complexity addNum() -> O(log n), only heap insertions and heap pops
# Time Complexity findMedian() -> O(1) constant time
# Space Complexity = O(n) to hold inputs
```

---

297. Serialize and Deserialize Binary Tree

```
# Serialize = needs to convert to String (not any other data structure) + lev
el-order traversal with 'null'
# Deserialize = convert string to list first, create root node + left subtree
+ right subtree ( if not 'null')

class Codec:

    def serialize(self, root):
        if root is None: # edge case
            return ''

        queue = []
        queue.append(root)

        result = ''
        while queue:
            node = queue.pop(0)
            if not node: # if node is not present add 'null,' as string
                result = result + 'null,'
                continue
            result = result + str(node.val) + ','
            queue.append(node.left)
            queue.append(node.right)
        return result


    def deserialize(self, data):
        if data == '': # edge case
```

```
            return None

        l = data.split(',')
        queue = []
        root = TreeNode(l[0]) # create a root node from val
        queue.append(root)

        i = 1 # lookahead 1 in list
        while queue and i < len(l): # Imp: i < len(l) condition
            node = queue.pop(0)
            if l[i] != 'null': # left subtree
                left = TreeNode(l[i])
                node.left = left
                queue.append(left)

            if i+1 < len(l):
                i = i+1

            if l[i] != 'null': # right subtree
                right = TreeNode(l[i])
                node.right = right
                queue.append(right)

            i = i+1

        return root

# Time Complexity = O(n) every node is visited once
```

```
# Space Complexity = O(n) space required to store node val in list and buildi
ng a tree
```

301. Remove Invalid Parentheses

https://leetcode.com/problems/remove-invalid-parentheses/discuss/75028/Short-Python-BFS (https://leetcode.com/problems/remove-invalid-parentheses/discuss/75028/Short-Python-BFS) (see in comments, readable version)

```
# Minimum removal - remove one and check whether valid is found, if not then
do so recursively
# Remove one bracket at every position and check if some valids are found
# if valids are not found, recursively remove one more element and check if v
alid is found
class Solution:
    def removeInvalidParentheses(self, s: str) -> List[str]:
        def isValid(s):
            balance = 0
            for c in s:
                if c == '(':
                    balance = balance + 1
                elif c == ')':
                    balance = balance - 1
                    if balance < 0:
                        return False
            return balance == 0

        level = {s} # Only one element at the beginning, set is used here in
order to avoid duplicate

        while len(level) > 0:
            valid = []
            for elem in level:
                if isValid(elem):
                    valid.append(elem)
            if valid: # 2. if any valid found
                return valid
```

```
            # initialize an empty set
            new_level = set()
            # 1. BFS -> Remove one element at every position
            # recursive hence in next iteration, it will remove one more elem
ent, if valid is not found
            for elem in level:
                for i in range(len(elem)):
                    new_level.add(elem[:i] + elem[i + 1:])

            level = new_level

# Time Complexity = O(n * 2^n), all subsets ( 2^n) need to be searched (n)
# Space Complexity = O(n * 2^n)
```

---

311. Sparse Matrix Multiplication

```
class Solution:
    def multiply(self, A: List[List[int]], B: List[List[int]]) -> List[List[i
nt]]:
        C = [[0 for _ in range(len(B[0]))] for _ in range(len(A))]

        for n in range(len(B[0])):
            for m in range(len(A)):

                for r in range(len(A[0])):
                    if A[m][r] == 0 or B[r][n] == 0:
                        continue
                    else:
                        C[m][n] = C[m][n] + A[m][r] * B[r][n]

        return C

# Time Complexity: Worst case: O(m*n*r) though if you skip calc when encounte
ring zero, it would be better
# Space Complexity = O(m*n)
```

322. Coin Change

https://leetcode.com/problems/coin-change/discuss/77361/Fast-Python-BFS-Solution (https://leetcode.com/problems/coin-change/discuss/77361/Fast-Python-BFS-Solution) (in comments)

---

323. Number of Connected Components in an Undirected Graph

https://leetcode.com/problems/number-of-connected-components-in-an-undirected-graph/discuss/77638/Python-DFS-BFS-Union-Find-solutions (https://leetcode.com/problems/number-of-connected-components-in-an-undirected-graph/discuss/77638/Python-DFS-BFS-Union-Find-solutions)

DFS: https://www.youtube.com/watch?v=uT1p5Eiw9CE (https://www.youtube.com/watch?v=uT1p5Eiw9CE) BFS: https://www.youtube.com/watch?v=ZVJFOrsHxMs (https://www.youtube.com/watch?v=ZVJFOrsHxMs)

```
class Solution:
    def countComponents(self, n: int, edges: List[List[int]]) -> int:

        visited = [0 for _ in range(n)]

        g = {x:[] for x in range(n)}
        for x,y in edges:
            g[x].append(y)
            g[y].append(x)

        #def dfs(i,g,visited):
            #if visited[i]:
                #return
            #visited[i] = 1
            #for neighbor in g[i]:
                #dfs(neighbor,g, visited)

        #counter = 0
        #for i in range(n):
            #if not visited[i]:
                #counter = counter + 1
                #dfs(i,g,visited)
        #return counter

        def bfs(i,g,visited):  # visit and adding to queue happens together
            visited[i] = 1
            q =[]
            q.append(i)
```

```
            while q:
                popped = q.pop(0)

                for neighbor in g[popped]:
                    if not visited[neighbor]: # neighbors can have common nei
ghbors
                        q.append(neighbor)
                        visited[neighbor] = 1
        counter = 0
        for i in range(n):
            if not visited[i]:
                counter = counter + 1
                bfs(i,g,visited)

        return counter
```

332. Reconstruct Itinerary

https://leetcode.com/problems/reconstruct-itinerary/discuss/469225/Intution-to-solve-the-question (https://leetcode.com/problems/reconstruct-itinerary/discuss/469225/Intution-to-solve-the-question)

http://buttercola.blogspot.com/2016/06/leetcode-332-reconstruct-itinerary.html (http://buttercola.blogspot.com/2016/06/leetcode-332-reconstruct-itinerary.html)

```
# all airports need to be visited => graph is not disconnected (also it is di
rected)
# topological sorting cannot be used as is as cycles are possible + all airpo
rts need to be visited
# dfs cannot be used as is as there can be a "dead end" somewhere in the tick
ets such that we are not able visit
# all airports
# the graph could even have some duplicate edges (i.e. we might have multiple
flights with the same origin and
# destination)

# while neighbors exist dfs + backtrack (visit all edges - eulerian path) : f
orce dfs to visit all edges in a directed connected graph with cycles

from collections import defaultdict
class Solution:
    def findItinerary(self, tickets: List[List[str]]) -> List[str]:

        adj_list = defaultdict(list)
        for x,y in tickets:
            adj_list[x].append(y)

        for key,value in adj_list.items(): # smallest lexical order
            adj_list[key] = sorted(value)

        def dfs(start_node, adj_list, itinerary):
            while len(adj_list[start_node]) > 0:
                neighbor = adj_list[start_node][0]
```

```
                adj_list[start_node].remove(neighbor)
                dfs(neighbor, adj_list, itinerary)


            itinerary.insert(0,start_node)


        itinerary = []
        start_node="JFK"
        dfs(start_node, adj_list,itinerary)
        return itinerary

# Time Complexity = O(V+E) (Similar to DFS)
# Space Complexity = O(V+E)
```

---

339. Nested List Weight Sum

```
# nestedList = sum([x.getList() for x in nestedList if not x.isInteger()],
[])
# This will concatenate the all the lists inside the current nestedList.
# sum([[1,2],[3,4]],[]) will return [1,2,3,4]
# nice trick to flatten a list of lists.

class Solution:
    def depthSum(self, nestedList: List[NestedInteger]) -> int:

        depth, result = 1, 0
        while nestedList:
            result += depth * sum([x.getInteger() for x in nestedList if x.is
Integer()])
            # just flatten for next iteration, do not add to result
                      # if the last arg below [] is not used, throws error
            # unsupported operand type(s) for +: 'int' and 'list'
            nestedList = sum([x.getList() for x in nestedList if not x.isInte
ger()], [])
            depth += 1
        return result

# Time Complexity = O(total number of elements in the input list)
# Space Complexity = O(1)
```

347. Top K Frequent Elements

```
# Here top k of 'freq' needs to be found out hence heap

from collections import Counter
class Solution:
    def topKFrequent(self, nums: List[int], k: int) -> List[int]:
        d = Counter(nums) # O(n)
        return [num for num,freq in d.most_common(k)] # can use heap here wit
h (freq, num) tuple

# Time Complexity = O(N log k) if k < N, if k=N the O(N)
# Space Complexity = O(N + k), hashmap + heap
```

349. Intersection of Two Arrays

```
# Duplicates allowed in input but output should not have duplicates even if t
hey are common in
# both arrays


class Solution:
    def intersection(self, nums1: List[int], nums2: List[int]) -> List[int]:
        s1 = set(nums1)
        s2 = set(nums2)

        result = s1.intersection(s2)
        return list(result)

# Time Complexity = O(m+n) where m and n are array's length. O(m) and O(n) ti
me is used to convert  # list into set

# Space Complexity = O(m+n) space required to create hashmap of arrays with l
ength m and n
```

350. Intersection of Two Arrays II

Hash Table Approach:

1. If nums1 is larger than nums2, swap the arrays.

2. For each element in nums1: Add it to the hash map m. Increment the count if the element is already there.
3. Initialize the insertion pointer (k) with zero.
4. Iterate along nums2: If the current number is in the hash map and count is positive: Copy the number into nums1[k], and increment k. Decrement the count in the hash map.
5. Return first k elements of nums1.

Time Complexity = O(m+n) where m and n are the lengths of the arrays Space Complexity = O(min(m,n)) - we build a hashmap, which is smaller of the two arrays

```
# Duplicates allowed in input and output should have duplicates if they are c
ommon in both arrays
# Two approaches (diff in complexity) - 1)hash map and 2) sort+two pointer ap
proach
# Also see approach 1 using hashmap in solution/notes

class Solution:
    def intersect(self, nums1: List[int], nums2: List[int]) -> List[int]:
        result = []

        nums1.sort()
        nums2.sort()

        j = k = 0

        while j < len(nums1) and k < len(nums2): # can be improved to iterate
for the length of smaller array
            if nums1[j] == nums2[k]:
                result.append(nums1[j]) # can be improved to use smaller arra
y to store results
                j = j+1
                k = k+1
            elif nums1[j] < nums2[k]:
                j = j+1
            elif nums1[j] > nums2[k]:
                k = k+1
        return result
```

```
# Time Complexity = O(m log m + n log n), sort both arrays + linear scan afte
rwards
# Space Complexity = O(m+n), ignore the space to store output but include spa
ce for sort
```

380. Insert Delete GetRandom O(1)

See Solution: Hash table - insert and delete is O(1) however to get random element requires to choose a random index and then retrieve an element with that index. There are no indexes in hashmap and hence to get true random value, one has first to convert hashmap keys in a list, that would take linear time.

The solution here is to build a list of keys aside and to use this list to compute GetRandom in constant time.

List has indexes and could provide Insert and GetRandom in average constant time, though has problems with Delete. To delete a value at arbitrary index takes linear time. The solution here is to always delete the last value: 1)Swap the element to delete with the last one. 2)Pop the last element out. For that, one has to compute an index of each element in constant time, and hence needs a hashmap which stores element -> its index dictionary.

```
 hashmap + list
# hashmap stores val -> index
# list deletion: find the index of value to be deleted from hashmap and swap
with last value, then pop the last value
import random
class RandomizedSet:

    def __init__(self):
        self.list = []
        self.dict = {}


    def insert(self, val: int) -> bool:
        if self.dict.get(val) == None:
            self.list.append(val)
            self.dict[val] = len(self.list) -1
            return True
        else:
            return False

    def remove(self, val: int) -> bool:
        if self.dict.get(val) != None:
            swap_index = self.dict[val]
            # swap in both list and dict
            self.list[-1], self.list[swap_index] = self.list[swap_index], sel
f.list[-1]
            self.dict[self.list[-1]], self.dict[self.list[swap_index]] = sel
f.dict[self.list[swap_index]], self.dict[self.list[-1]]
```

```
            # delete in both list and dict
            self.list.pop()
            del self.dict[val]
            return True
        return False


    def getRandom(self) -> int:

        r = random.randint(0, len(self.list) - 1) # both ends are included
        return self.list[r]
```

384. Shuffle an Array

import python

random.random() -> returns a number b/w [0.0,1.0) random.randint(a,b) -> returns an int between [a,b] random.choice(seq) -> returns an element from seq, throws error if seq is empty

Random = Every element in list should be **equally likely** to be picked up

This question is basically asking to generate random numbers equal to the size of input list and these random numbers should not repeat (sampling without replacement)

Brute Force Approach:

1. Pick a random index, output the number corresponding to it, store in output array
2. Remove that index from input array and repeat step 1 This will require an extra array

Another efficient approach without requiring extra memory

1. Pick a random index and swap the number corresponding to the picked index with the first element of array
2. Now pick a random index from second to last index and swap the number corresponding to the picked index with the second second element of array. This way every index is equally likely to be picked up and the index that is picked in one iteration will not be a candidate in next iteration

Variation of this problem:

1) input has duplicates and output can contain duplicates - same as above 2) input has duplicates and output should not contain duplicates - if the random element chosen is already part of output, throw it away and repeat (rejection sampling)

Complexity Analysis: Time complexity : O(n) The Fisher-Yates algorithm runs in linear time, as generating a random index and swapping two values can be done in constant time.

Space complexity : O(n) Although we managed to avoid using linear space on the auxiliary array from the brute force approach, we still need it for reset, so we're stuck with linear space complexity.

https://leetcode.com/articles/shuffle-an-array/ (https://leetcode.com/articles/shuffle-an-array/)
http://www.radwin.org/michael/2015/01/13/unique-random-numbers-technical-interview-question/ (http://www.radwin.org/michael/2015/01/13/unique-random-numbers-technical-interview-question/)

```
import random

class Solution:

    def __init__(self, nums: List[int]):
        self.nums = nums
        self.original = nums[:] # deep copy
        #(In the Python code)
#For anyone wondering, the utility derived in creating a new List object with
each assignment
#is to create a 'Deep Copy' of the array.

    def reset(self) -> List[int]:
        """
        Resets the array to its original configuration and return it.
        """
        self.nums = self.original # both will change together
        self.original = self.original[:] # deep copy
        return self.nums

    def shuffle(self) -> List[int]:
        """
        Returns a random shuffling of the array.
        """
        # sampling without replacement
        for i in range(len(self.nums)):
            random_index = random.randint(i,len(self.nums)-1)
            self.nums[i], self.nums[random_index] = self.nums[random_index],
```

```
self.nums[i]
        return self.nums

#Complexity Analysis:
#Time complexity : O(n)
#The Fisher–Yates algorithm runs in linear time, as generating a random index
and swapping two values can #be done in constant time.

#Space complexity : O(n)
#Although we managed to avoid using linear space on the auxiliary array from
the brute force approach, we #still need it for reset, so we're stuck with li
near space complexity.
```

---

394. Decode String

https://leetcode.com/problems/decode-string/discuss/508115/Simple-python-with-stack-easy-to-understand (https://leetcode.com/problems/decode-string/discuss/508115/Simple-python-with-stack-easy-to-understand)

```
# Push in stack unless ending brackets
# Extract the string between closing and ending brackets, and digit (can be m
ultiple) before
# opening brackets
# Push the digit times string back into stack

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

        stack = []

        for element in s: # Loop through string
            if element == ']':

                # temp holds the popped element
                temp = stack.pop()
                sub_result = ""
                while (temp != '[' ): # Extract between ] and [
                    sub_result = temp + sub_result
                    temp = stack.pop()

                # temp points to the top of stack and checks whether it is a
digit
                temp_digit = stack[-1]
                digit = ""
                while(temp_digit.isdigit()): # If multiple digits
                    digit = stack.pop() + digit
                    if stack:  # check whether stack is empty
```

```
                        temp_digit = stack[-1]
                    else:
                        break

                sub_result = sub_result * int(digit)

                stack.append(sub_result) # Push back into stack (most importa
nt trick)


            elif element != ']':
                stack.append(element)

        return "".join(stack)

# Time Complexity = O(n)
# Space Complexity = O(n)
```

398. Random Pick Index

https://gregable.com/2007/10/reservoir-sampling.html (https://gregable.com/2007/10/reservoir-sampling.html)

Reservoir Sampling is an algorithm for sampling elements from a stream of data.

Your goal is to efficiently return a random sample of 1,000 elements evenly distributed from the original stream. How would you do it?

The right answer is generating random integers between 0 and N - 1, then retrieving the elements at those indices and you have your answer. If you need to be generate unique elements, then just throw away indices you've already generated (rejection sampling)

So, let me make the problem harder. You don't know N (the size of the stream) in advance and you can't index directly into it.

First, you want to make a reservoir (array) of 1,000 elements and fill it with the first 1,000 elements in your stream. That way if you have exactly 1,000 elements, the algorithm works. This is the base case.

Now we have to make probability of 1001th element being part of 1000 elements = probability that any element within 1000 elements remains in the set

prob that 1001th element becomes part of 1000 elements = 1000/1001 (i)

The probability of removing any one element is the probability of element 1,001th getting selected multiplied by the probability of that element getting randomly chosen as the replacement candidate from the 1,000 elements in the reservoir. That probability is: 1000/1001 * 1/1000 = 1/1001

the probability that the any one element survives this round is: 1 - 1/1001 = 1000/1001 (ii)

Since (i) = (ii)

```
# Brute force: store all numbers and their index which are equal to target, a
nd then pick one randomly
# Brute force approach will need to scan for all elements in input and store
all elements which are equal # to target beforehand -> we don't know size of
array + too much extra space (input array is large with
# duplicates)

# Reservoir Sampling (array size is too large)

# To randomly pick up k elements in an array S with very big size N with the
same probability
# (1) Get the first k elements from S and put them into an array result[]
# (2) for j > k && j < N:
# generate a random number r [0, j) and take its floor int value
# if this random number r < k: result[r] = S[j] else don't replace
# this ensures every element in S has equal probability to be part of k eleme
nts


# let's take: j = k+1
# total possibilities = k (as floor value is taken), if r < k -> replace (ana
logous to 1000/1001)

# here k = 1



import random
```

```
class Solution:

    def __init__(self, nums: List[int]):
        self.nums=nums


    def pick(self, target: int) -> int:
        count = 0
        result = -1 # initialize whatever you want

        for i in range(len(self.nums)):
            if self.nums[i] != target:
                continue
            count = count + 1
            rand = random.randint(1,count)
            if rand == 1:
                result = i

        return result

# Time Complexity = O(n)
# Space Complexity = O(1)
```

To those who don't understand why it works. Consider the example in the OJ {1,2,3,3,3} with target 3, you want to select 2,3,4 with a probability of 1/3 each.

2 : It's probability of selection is 1 * (1/2) * (2/3) = 1/3
3 : It's probability of selection is (1/2) * (2/3) = 1/3
4 : It's probability of selection is just 1/3

So they are each randomly selected.

Thanks. A bit more explanation. Let's take index 2 for example.

The first time we saw target 3 is at index 2. The count is 0. Our reservoir only have 0 and we need to pick rnd.nextInt(++count) == 0. The probability is 1. Result = 2.
Then we went to index 3. The count is 1. Our reservoir is has [0,1]. We say if we get 0, we'll change the the result, otherwise we keep it. Then chance that we keep the result=2 is 1/2 which means we got 1 from the reservoir.
Then we went to index 4. count =2. Our reservoir has [0,1,2]. Same as before, if we get 0, then we'll change the result. The chance we get 0 is 1/3, while the chance we didn't get is 2/3. i.e The chance we keep the result ==2 is 2/3.
The chance we get index=2 is 1*1/2*2/3=1/3

409. Longest Palindrome

```
# Create a hastable with freq of chars
# All chars with even frequency are taken
# All chars with odd freq = take their even component
# You can take one odd component, if there are any

from collections import defaultdict
class Solution:
    def longestPalindrome(self, s: str) -> int:
        d = defaultdict(int)
        result = 0
        for c in s:
            d[c] = d[c] + 1

        even_count = 0
        odd_count_present = 0
        # e.g. abababababa -> 9
        for key in d.keys():
            even_count = even_count + d[key]//2
            if d[key] % 2 != 0:
                odd_count_present = 1

        result = (even_count * 2) + odd_count_present
        return result

# Time Complexity = O(n)
# Space complexity = O(1) as hash table keys are alphabets which can not exce
ed 26 (independent of input size)
```

415. Add Strings

ord() in python: Given a string of length one, return an integer representing the Unicode code point of the character Input : a Output : 97

ord('9') - ord('0') => 9

Imp: If the string length is more then one, and a **TypeError** will be raised.

zfill: The zfill() method adds zeros (0) at the beginning of the string, until the string reaches the specified length provided as len argument Syntax : string.zfill(len)

Also, result[::-1] does not reverse list in-place, it **returns* * reversed list

```
class Solution:
    def addStrings(self, num1: str, num2: str) -> str:

        max_len = max(len(num1), len(num2))
        num1, num2 = num1.zfill(max_len), num2.zfill(max_len)

        p = len(num1) - 1
        carry = 0
        result = []
        while p >= 0:
            x1 = ord(num1[p]) - ord('0')
            x2 = ord(num2[p]) - ord('0')

            sum = (x1 + x2 + carry) % 10
            carry = (x1 + x2 + carry) // 10
            result.append(sum)
            p = p -1

        if carry > 0:
            result.append(carry)

        return "".join(str(x) for x in result[::-1])

# Time and Space Complexity = O(max(n1,n2)) where n1 and n2 are lengths of nu
m1 and num2
```

417. Pacific Atlantic Water Flow

DFS Solution:

https://leetcode.com/problems/pacific-atlantic-water-flow/discuss/90733/Java-BFS-and-DFS-from-Ocean (https://leetcode.com/problems/pacific-atlantic-water-flow/discuss/90733/Java-BFS-and-DFS-from-Ocean)

```
class Solution:
    def pacificAtlantic(self, matrix: List[List[int]]) -> List[List[int]]:
        if (len(matrix) == 0) or (len(matrix[0]) == 0):
            return []
        rows = len(matrix)
        cols = len(matrix[0])

        pacific_visited = [[0 for _ in range(cols)] for _ in range(rows)]
        atlantic_visited = [[0 for _ in range(cols)] for _ in range(rows)]

        directions = [[1,0], [0,1], [-1,0], [0,-1]]

        def dfs(matrix, visited, i, j):

            visited[i][j] = 1

            for direction in directions:
                next_i, next_j = i + direction[0], j + direction[1]
                if 0 <= next_i < len(matrix) and 0 <= next_j < len(matrix[0])
and visited[next_i][next_j] != 1  and matrix[i][j] <= matrix[next_i][next_j]:
                    dfs(matrix, visited, next_i,next_j)


        for i in range(rows):
            dfs(matrix, pacific_visited,i,0)
            dfs(matrix, atlantic_visited,i, cols-1)

        for i in range(cols):
```

```
            dfs(matrix, pacific_visited, 0, i)
            dfs(matrix, atlantic_visited, rows -1, i)

        result = []
        for i in range(rows):
            for j in range(cols):
                if pacific_visited[i][j] == 1 and atlantic_visited[i][j] ==
1:
                    result.append([i,j])
        return result

# Time Complexity - Since we keep a visited list for each ocean, we only visi
t a cell if it is not visited before. # For each ocean, the worst case is NM
thus totally O(NM)
```

424. Longest Repeating Character Replacement

https://leetcode.com/problems/minimum-window-substring/discuss/26808/here-is-a-10-line-template-that-can-solve-most-substring-problems (https://leetcode.com/problems/minimum-window-substring/discuss/26808/here-is-a-10-line-template-that-can-solve-most-substring-problems)

437. Path Sum III

https://leetcode.com/problems/path-sum-iii/discuss/350205/anybody-has-some-suggestions-on-implementing-prefix-sum-solution-iteratively (https://leetcode.com/problems/path-sum-iii/discuss/350205/anybody-has-some-suggestions-on-implementing-prefix-sum-solution-iteratively)

https://leetcode.com/problems/path-sum-iii/discuss/141424/Python-step-by-step-walk-through.-Easy-to-understand.-Two-solutions-comparison.-%3A- (https://leetcode.com/problems/path-sum-iii/discuss/141424/Python-step-by-step-walk-through.-Easy-to-understand.-Two-solutions-comparison.-%3A-))

https://leetcode.com/problems/path-sum-iii/discuss/91892/Python-solution-with-detailed-explanation (https://leetcode.com/problems/path-sum-iii/discuss/91892/Python-solution-with-detailed-explanation)

```
class Solution:
    def pathSum(self, root: TreeNode, sum: int) -> int:
        if not root:
            return 0

        prefix_sum = defaultdict(int)
        prefix_sum[0] = 1 # dict (map) that will be used to keep track of pre
sum values
        stack = [(root, 0, prefix_sum)]
        count = 0

        while stack:
            root, curr, prefix_sum = stack.pop()
            curr += root.val

            # check to see if we've found any path with given sum
            if curr - sum in prefix_sum:
                count = count + prefix_sum[curr - sum]

            # update prefix_sum
            prefix_sum[curr] = prefix_sum.get(curr, 0) + 1

            # go to subtrees
            # need to create copy of prefix_sum as when unwinding the stack p
refix_sum state needs to be restored from the point it was left

            if root.left:
                stack.append((root.left, curr, dict(prefix_sum)))
```

```
            if root.right:
                stack.append((root.right, curr, dict(prefix_sum)))

        return count

# Time Complexity = O
```

---

438. Find All Anagrams in a String

```
# This is a problem of multiple pattern search in a string. All such problems
usually could be
# solved by sliding window approach in a linear time.
# Hashmap - since order comparison is not required

from collections import Counter
class Solution:
    def findAnagrams(self, s: str, p: str) -> List[int]:
        if len(p) > len(s): # edge case
            return []

        p_dict = Counter(p)
        s_dict = Counter() # empty
        result = []
        for i in range(len(s)):
            # add one more letter on the right side of the window: build wind
ow of size pattern + keep adding to it
            s_dict[s[i]] = s_dict[s[i]] + 1

            # remove one letter from the left side of the window: slide the w
indow
            if i>=len(p):
                if s_dict[s[i-len(p)]] == 1:
                    del s_dict[s[i-len(p)]]
                else:
                    s_dict[s[i-len(p)]] = s_dict[s[i-len(p)]] -1 # e.g. s = a
bca and p = abc
            # compare
```

```
            if s_dict == p_dict:
                result.append(i-len(p)+1)
        return result

# Time Complexity = O(s + p) where s,p = length of string s and p (one-pass)
# Space Complexity = O(1), keys of dict cannot be more than 26 chars
```

---

442. Find All Duplicates in an Array

Approach 4: Mark Visited Elements in the Input Array itself

```
# The integers in the input array arr satisfy 1 ≤ arr[i] ≤ n, where n is the
size of array
# All the integers present in the array are positive
# The decrement of any integers present in the array must be an accessible in
dex in the array

# Iterate over the array and for every element x in the array, negate the val
ue at index abs(x)-1
# The negation operation effectively marks the value abs(x) as seen / visited

# Double negation i.e. +ve means the element was seen twice

class Solution:
    def findDuplicates(self, nums: List[int]) -> List[int]:
        result = []

        for num in nums:
            nums[abs(num)-1] = nums[abs(num)-1] * -1

        for num in nums:
            if nums[abs(num)-1] > 0:
                result.append(abs(num)) # Imp: add abs(x) in result
                nums[abs(num)-1] = nums[abs(num)-1] * -1  # negate so as to a
void double counting at second occurence


        return result
```

```
# Time Complexity = O(n)
# Space Complexity = O(1)
```

---

449. Serialize and Deserialize BST

https://leetcode.com/problems/serialize-and-deserialize-bst/discuss/93171/Python-O(-N-)-solution.-easy-to-understand (https://leetcode.com/problems/serialize-and-deserialize-bst/discuss/93171/Python-O(-N-)-solution.-easy-to-understand)

```
# Serialize and Deserialize binary tree method (LC 297) cannot be used as the question mentions encoded string should be 'as compact as possible' meaning you cannot use 'null' for empty left or right child (though we can use a delimiter to separate out nodes)

# Unique BST could be constructed from preorder or postorder traversal only (not inorder or level order)

# That means that BST structure is already encoded in the preorder or postorder traversal and hence they are both suitable for the compact serialization

from collections import deque
class Codec:

    def serialize(self, root):
        vals = []

        def preOrder(node):
            if node:
                vals.append(node.val)
                preOrder(node.left)
                preOrder(node.right)

        preOrder(root) # preorder is DLR

        return ' '.join(map(str, vals))

    def deserialize(self, data):
```

```
        vals = deque(int(val) for val in data.split()) # put all elements in
deque

        def build(minVal, maxVal):
            # left subtree - if the next element is between minVal and val
            # right subtree - if the next element is between val and maxVal
            if vals and minVal < vals[0] < maxVal:
                val = vals.popleft() # leftmost elemnt is popped out (remove
d)
                #print("val = {}".format(val))
                #print("minval = {}".format(minVal))
                #print("maxval = {}".format(maxVal))
                root = TreeNode(val)
                root.left = build(minVal, val)
                root.right = build(val, maxVal)
                return root

        return build(float('-infinity'), float('infinity'))

# Time Complexity = O(n)
# Space Complexity = O(n)

# Deserialization = deque takes O(n), build takes O(n) => O(n)
```

490. The Maze

https://leetcode.com/problems/the-maze/discuss/198453/Python-BFS-tm
(https://leetcode.com/problems/the-maze/discuss/198453/Python-BFS-tm)

```
# DFS time limit exceeded, BFS - shortest path
class Solution:
    def hasPath(self, maze: List[List[int]], start: List[int], destination: L
ist[int]) -> bool:
        q = []
        q.append(start)

        visited = [[0 for _ in range(len(maze[0]))] for _ in range(len(maz
e))]
        visited[start[0]][start[1]] = 1
        directions = [[-1,0], [0,1], [1,0], [0,-1]]

        while q:
            i,j = q.pop(0)

            if i == destination[0] and j == destination[1]:
                return True

            for direction in directions:
                next_i, next_j = i + direction[0], j + direction[1]
                # Roll the ball until it hits a wall
                while 0 <= next_i < len(maze) and 0 <= next_j < len(maze[0])
and maze[next_i][next_j] == 0:
                    next_i = next_i + direction[0]
                    next_j = next_j + direction[1]

                # next_i,next_j hit a wall when exiting the above while loop,
so we need to backtrack 1 position
```

```
                next_i = next_i - direction[0]
                next_j = next_j - direction[1]
                if visited[next_i][next_j] == 0: # if not visited
                    q.append([next_i, next_j])
                    visited[next_i][next_j] = 1

        return False
```

---

515. Find Largest Value in Each Tree Row

https://medium.com/@timpark0807/leetcode-is-easy-binary-tree-patterns-1-2-6e1793b76415 (https://medium.com/@timpark0807/leetcode-is-easy-binary-tree-patterns-1-2-6e1793b76415)

```
# Keep track of levels, for same level keep track of max, when level changes
append this max value and
# reset max and prev_level
class Solution:
    def largestValues(self, root: TreeNode) -> List[int]:
        if root is None: # edge case
            return []

        queue = []

        queue.append((root,0))

        prev_level = 0
        result = []
        max_val = float('-inf')

        while queue:
            cur, level = queue.pop(0)
            if level == prev_level: # for the same level, keep track of max_v
alue
                max_val = max(max_val, cur.val)


            elif level != prev_level: # if level changes, append max and rese
t max and reset prev_level
                result.append(max_val)
                max_val = max(float('-inf'), cur.val) # new max is this new l
evel's first element
```

```
                prev_level = level

            if cur.left:
                queue.append((cur.left, level+1))

            if cur.right:
                queue.append((cur.right, level+1))

            prev_level = level

        result.append(max_val) # last max was not appended since level did no
t change
        return result

# Time Complexity = O(n)
# Space Complexity = O(n) for storing elements in queue
```

523. Continuous Subarray Sum

whenever the same sum%k value is obtained corresponding to two indices i and j, it implies that sum of elements betweeen those indices is an integer multiple of k

a%k = x b%k = x (a - b) %k = x -x = 0 here a - b = the sum between i and j.

For anybody confused about map.put(0,-1); In the case nums = [1, 5] k = 6, at i=1, sum % k is 0, so we need a key '0' in the map, and it must be comply with the continuous condition, i - map.get(sum) > 1, so we give an arbitrary value of -1.

Lots of edge cases: [0] 0 Expected: false

[0,0] 0 Expected: true

[23,2,6,4,7] 0 Expected: false

[0,1,0] 0 Expected: false

[15,0,0,3] 4 Expected: true (it returns true as it should because 0+0 = 0*4)

https://leetcode.com/problems/continuous-subarray-sum/discuss/99503/Need-to-pay-attention-to-a-lot-of-corner-cases (https://leetcode.com/problems/continuous-subarray-sum/discuss/99503/Need-to-pay-attention-to-a-lot-of-corner-cases)...

```
# Create a map of form: cumsum%k:index

# whenever the same sum%k value is obtained corresponding to two indices i an
d j, it implies that sum of
# elements betweeen those indices is an integer multiple of k

# if at any point cumsum%k already exists in map and index difference is > 1,
return True
# Edge cases: [0] 0 -> False, [0,0] 0 -> True

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

        d = {0 :-1} # to handle case: nums = [1, 5] k = 6 & comply with condi
tion: index - d[cumsum] > 1
                    # also to handle edge cases
        cumsum = 0

        for index, num in enumerate(nums):
            cumsum = cumsum + nums[index]
            if k != 0: # k should not be zero else division by zero error
                cumsum = cumsum%k
            # below get executed even for k == 0
            if d.get(cumsum) != None:
                if index - d[cumsum] > 1: # subarray size atleast 2
                    return True
            else:
                d[cumsum] = index
```

```
        return False

# Time Complexity = O(n)
# Space Compexity = O(min(n,k)) : HashMap can contain upto min(n,k) different
pairings.
```

542. 01 Matrix

https://algorithms.tutorialhorizon.com/breadth-first-search-bfs-in-2d-matrix-2d-array/ (https://algorithms.tutorialhorizon.com/breadth-first-search-bfs-in-2d-matrix-2d-array/)

https://medium.com/@silasburger/01-matrix-leetcode-javascript-walkthrough-3747f894092c (https://medium.com/@silasburger/01-matrix-leetcode-javascript-walkthrough-3747f894092c)

https://medium.com/@lenchen/leetcode-542-01-matrix-b85e06193ec8 (https://medium.com/@lenchen/leetcode-542-01-matrix-b85e06193ec8)

Going level by level, or distance by distance in this case, is the nature of BFS. You should be reminded of BFS whenever you are asked to find the absolute shortest distance in any matrix or graph problem.

https://medium.com/basecs/going-broad-in-a-graph-bfs-traversal-959bd1a09255 (https://medium.com/basecs/going-broad-in-a-graph-bfs-traversal-959bd1a09255)

543. Diameter of Binary Tree

https://leetcode.com/problems/diameter-of-binary-tree/discuss/101145/Simple-Python (https://leetcode.com/problems/diameter-of-binary-tree/discuss/101145/Simple-Python)

```
class Solution:
    def diameterOfBinaryTree(self, root: TreeNode) -> int:
        self.result = 0 # self is needed as it needs to keep track of max lef
t+right heights

        def get_height(node):
            if node is None:
                return 0

            left = get_height(node.left)
            right = get_height(node.right)
            self.result = max(self.result, left+right) # path thru this node
is len of left+righ subtree
            return max(left, right) + 1

        get_height(root)
        return self.result

# Time Complexity = O(n)
# Space Complexity = O(n) for keeping track of recursion stack
```

547. Friend Circles

https://leetcode.com/problems/friend-circles/discuss/101431/Stupid-question%3A-How-is-this-question-different-from-Number-of-Islands (https://leetcode.com/problems/friend-circles/discuss/101431/Stupid-question%3A-How-is-this-question-different-from-Number-of-Islands)

https://leetcode.com/problems/friend-circles/discuss/228414/Wrong-problem-statement-made-me-waste-half-an-hour-looking-for-a-solution-for-a-complex-problem (https://leetcode.com/problems/friend-circles/discuss/228414/Wrong-problem-statement-made-me-waste-half-an-hour-looking-for-a-solution-for-a-complex-problem)

https://leetcode.com/problems/friend-circles/discuss/201096/Pythonthe-classic-DFS-super-easy-to-understand-comments-the-whole-shabang (https://leetcode.com/problems/friend-circles/discuss/201096/Pythonthe-classic-DFS-super-easy-to-understand-comments-the-whole-shabang)!

https://leetcode.com/problems/friend-circles/discuss/101349/Python-Simple-Explanation (https://leetcode.com/problems/friend-circles/discuss/101349/Python-Simple-Explanation)

```
# Adj Matrix and not adj list is given (square matrix)
# symmetric matrix
# Understand the input matrix representation

class Solution:
    def findCircleNum(self, M: List[List[int]]) -> int:
        # Create adj list using adj matrix
        g = {x:[] for x in range(len(M))}
        for i in range(len(M)):
            for j in range(len(M[0])):
                if M[i][j] == 1 and i != j: # Important: avoid self loop
                    g[i].append(j)

        visited = [0 for _ in range(len(M))]

        # usual dfs
        def dfs(node,g,visited):
            if visited[node] == 1:
                return
            visited[node] = 1

            for neighbor in g[node]:
                dfs(neighbor, g, visited)

        friend_circle = 0
        for start_node in range(len(g)):
            if visited[start_node] == 0:
                friend_circle = friend_circle + 1
```

```
                dfs(start_node, g, visited)

        return friend_circle
```

---

560. Subarray Sum Equals K

if the cumulative sum upto two indices, say i and j is at a difference of k i.e. if sum[i] - sum[j] =k, the sum of elements lying between indices i and j is k.

Approach 4 - https://leetcode.com/problems/subarray-sum-equals-k/solution/ (https://leetcode.com/problems/subarray-sum-equals-k/solution/) (see the animation)

We traverse over the array nums and keep on finding the cumulative sum. Every time we encounter a new sum, we make a new entry in the hashmap corresponding to that sum. If the same sum occurs again, we increment the count corresponding to that sum in the hashmap. **Further, for every sum encountered, we also determine the number of times the sum 'sum–k' has occured already, since it will determine the number of times a subarray with sum k has occured upto the current index**. We increment the count by the same amount.

```
# Create a map of cumsum: #of occurrence

# if the cumulative sum upto two indices, say i and j is at a difference of k
i.e. if sum[i] - sum[j] # =k, the sum of elements lying between indices i and
j is k.

# if cumsum-k exists, it means # of occurrences of cumsum -k should be added
# if cumsum == k or cumsum-k == 0 (base case),  add it's # of occurence

# e.g. [1,1,1] k= 2, o/p -> 2

class Solution:
    def subarraySum(self, nums: List[int], k: int) -> int:
        d = {0:1} # to cover base case cumsum = 0 occurs 1 time, also for cas
e when cumsum == k
        cumsum = 0
        counter = 0

        for i in range(len(nums)):
            cumsum = nums[i] + cumsum

            if d.get(cumsum - k) != None:
                counter = counter + d[cumsum - k]


            if d.get(cumsum) == None:
                d[cumsum] = 1
            else:
```

```
                d[cumsum] = d[cumsum] + 1


        return counter

# Time Complexity = O(n)
# Space Complexity = O(n)
```

---

566. Reshape the Matrix

```
class Solution:
    def matrixReshape(self, nums: List[List[int]], r: int, c: int) -> List[Li
st[int]]:
        self.nums = nums
        ro = len(self.nums)
        co = len(self.nums[0])
        so = ro*co
        s = r*c
        if (so != s):
            return self.nums

        flatten = [x for y in self.nums for x in y] # Flatten matrix
        k = 0
        result = []
        for i in range(r):
            sub_result = []
            for j in range(c):
                sub_result.append(flatten[k])
                k = k+1
            result.append(sub_result)
        return result
```

567. Permutation in String

https://leetcode.com/problems/permutation-in-string/discuss/102594/Python-Simple-with-Explanation (https://leetcode.com/problems/permutation-in-string/discuss/102594/Python-Simple-with-Explanation)

```
class Solution:
    def checkInclusion(self, s1: str, s2: str) -> bool:
        if len(s1) > len(s2): # edge case
            return []

        p_dict = Counter(s1)
        s_dict = Counter() # empty

        for i in range(len(s2)):
            # add one more letter on the right side of the window: build wind
ow of size pattern + keep adding to it
            s_dict[s2[i]] = s_dict[s2[i]] + 1

            # remove one letter from the left side of the window: slide the w
indow
            if i>=len(s1):
                if s_dict[s2[i-len(s1)]] == 1:
                    del s_dict[s2[i-len(s1)]]
                else:
                    s_dict[s2[i-len(s1)]] = s_dict[s2[i-len(s1)]] -1 # e.g. s
= abca and p = abc
            # compare
            if s_dict == p_dict:
                return True
        return False

# Time Complexity = O(len(s1) + len(s2))
# Space Complexity = O(1), keys of dict cannot be more than 26 chars
```

572. Subtree of Another Tree

First comment in: https://leetcode.com/problems/subtree-of-another-tree/discuss/102741/Python-Straightforward-with-Explanation-(O(ST)-and-O(S%2BT)-approaches (https://leetcode.com/problems/subtree-of-another-tree/discuss/102741/Python-Straightforward-with-Explanation-(O(ST)-and-O(S%2BT)-approaches))

Approach 2 in Solutions

```
# 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 isSubtree(self, s: TreeNode, t: TreeNode) -> bool:

        # checks if two trees are same
        def isMatch(s, t):
            if (s is None and t is not None) or (s is not None and t is Non
e):
                return False
            elif s is None and t is None:
                return True

            if s.val == t.val:
                if isMatch(s.left, t.left) and isMatch(s.right, t.right):
                    return True
                else:
                    return False

        if isMatch(s, t): # Both are exact same trees
            return True
        if s is None: # check needed as below isSubtree uses s.left and s.rig
ht
            return False
```

```
        # recursively check s left subtree with t and s right subtree with t
        if self.isSubtree(s.left, t) or self.isSubtree(s.right, t):
            return True
        else:
            return False

# Time Complexity = O(m*n) where m, n is number of nodes in t and s
# Space Complexity = O(n) where n number of nodes in s
```

---

621. Task Scheduler

```
# List approach to calculate freq bcz sorting by dict key cannot be done in-p
lace and needs extra space
# calc initial total idle slots: (max freq -1) * cool-off time
# calc how much of idle time can be utilized with tasks with lower/equal freq
than the max freq task  : idle_slots = idle_slots - min(f_max - 1, freq.pop
())

class Solution:
    def leastInterval(self, tasks: List[str], n: int) -> int:

        freq = [0] * 26

        for t in tasks:
            freq[ord(t) - ord('A')] = freq[ord(t) - ord('A')] + 1

        freq.sort()

        f_max = freq.pop()
        idle_slots = (f_max -1) * n

        while idle_slots > 0 and freq:
            # if top two tasks have same freq f_max-1 will be taken else fre
q.pop() will be taken
            # e.g top two tasks with same freq (e.g. 2) and n= 1
            idle_slots = idle_slots - min(f_max - 1, freq.pop())

        idle_slots = max(0, idle_slots) # idle time cannot go below 0
```

```
        return idle_slots + len(tasks)


# Time Complexity = O(n) where n = total number of tasks. freq list is of len
gth 26 irrespective of length of tasks list
# Space Complexity = O(1) since freq list is always of length 26
```

---

636. Exclusive Time of Functions

https://leetcode.com/problems/exclusive-time-of-functions/discuss/105100/Python-Straightforward-with-Explanation (https://leetcode.com/problems/exclusive-time-of-functions/discuss/105100/Python-Straightforward-with-Explanation)

```
# end timestamp is "ending at the end" of the timestamp hence while pop ts+1
used in substraction
# e.g. 0:start:0, 1:start:2, 1:end:5, 0:end:10

class Solution:
    def exclusiveTime(self, n: int, logs: List[str]) -> List[int]:
        result = [0] * n
        stack = []
        prev_ts = 0

        for log in logs:
            fn_id, event, ts = log.split(":")
            ts = int(ts)
            fn_id = int(fn_id) # needed else result[stack[-1]] will throw err
or: list indices must be int

            if event == 'start':
                if stack:
                    result[stack[-1]] = result[stack[-1]] + ts - prev_ts

                stack.append(fn_id)
                prev_ts = ts

            else:
                result[stack.pop()] += ts+1 - prev_ts  # use + else two pop()
and will throw error
                prev_ts = ts+1
```

```
        return result

# Time Complexity = O(number of elements in logs) i.e. len(logs)
# Space Complexity = O (len(logs) / 2), stackstack can grow upto a depth of a
tmost n/2
```

---

647. Palindromic Substrings

https://www.techiedelight.com/find-possible-palindromic-substrings-string/
(https://www.techiedelight.com/find-possible-palindromic-substrings-string/)

```
# All "possible" palindrome substrings
# Substrings - continguous
# Notice that if [a, b] is a palindromic interval (meaning S[a], S[a+1], ...,
S[b] is a
# palindrome), then [a+1, b-1] is one too
# expand around center and count all valid palindromes

class Solution:
    def countSubstrings(self, s: str) -> int:
        def expand(s,low,high):
            l = []
            count = 0
            while(low >= 0 and high <len(s) and s[low]==s[high]):
                low = low-1
                high = high +1

                count = count + 1
                l.append(s[low+1:high]) # keep adding all possible palindrome
s for a given low and high
            return count
            #return len(l) # if s[low+1:high] = "" then len = 0



        counter = 0
        for i in range(len(s)):
            curr_odd_len = expand(s,i,i)
            counter = counter + curr_odd_len
```

```
            curr_even_len = expand(s, i, i+1)
            counter = counter + curr_even_len

        return counter

# Time Complexity = O(n^2) , at each position in n, expand till ends of strin
g of len n
# Space Complexity = O(1)
```

658. Find K Closest Elements

element do not have to be closest by position in array closest on a number line

```
import bisect
class Solution:
    def findClosestElements(self, arr: List[int], k: int, x: int) -> List[in
t]:
        # if the target x is more or equal than the last element in the sorte
d array, the last k elements are the result
        # If the target x is less or equal than the first element in the sort
ed array, the first k elements are the result
        if x >= arr[-1]:
            return arr[len(arr)-k: ]
        if x <= arr[0]:
            return arr[:k]

        # find the index of the element, which is equal (when this list has
x) or a little bit             larger than x (when this list does not have i
t)
        index = bisect.bisect_left(arr, x)

        # Find the index window between which search needs to be done
        low = max(0, index-k)
        high = min(len(arr)-1, index+k)

        # binary search,  closest not by position but on number line
                # [1,2,3,4,5] x = 3 and k = 4: ans -> [1,2,3,4]
        while high-low >= k:
            if (x - arr[low] <= arr[high] -x): # In case of tie, pick smaller
elements
                high = high - 1
```

```
            elif (x - arr[low] > arr[high] - x):
                low = low + 1

        return arr[low:high+1]


# Time Complexity = O(log n + k), O(log n) for binary search and O(k) for sli
cing array
# Space Complexity = O(k) for slicing to generate output
```

---

670. Maximum Swap

map(func, iter) -> Returns a list of the results(iterable) after applying the given function to each item of a given iterable

hence num = map(list, str(num)) will not work as each element of num will become list

https://leetcode.com/problems/maximum-swap/discuss/107066/Python-Straightforward-with-Explanation (https://leetcode.com/problems/maximum-swap/discuss/107066/Python-Straightforward-with-Explanation)

```
# Create map of each digit in num (key = digit and value = index)
# Iterate thru each digit in num and if any digit from 9 to x+1 is already in
num and if it is at a later  # position than the digit, swap and return resul
t (max 1 swap allowed)

# Be careful with converting int to list of strings

class Solution:
    def maximumSwap(self, num: int) -> int:
        A = list(str(num))  # list of string elements
        d = { int(x): i for i, x in enumerate(A) }

        for i,x in enumerate(A):
            for digit in range(9, int(x),-1):  # Imp: from 9,8,7...,x+1
                if d.get(digit) != None and d.get(digit) > i:

                    A[i], A[d[digit]] = A[d[digit]], A[i]
                    return int("".join(A))  # not break since two for loops

        return num # if the num is already in descending order e.g. 9973 (abo
ve return will not be hit)

# Time Complexity = O(n); iterating from 9 to x+1 in descending order is cons
tant (independent of input)
# Space Complexity = O(n) i.e. storage for A; d is constant as it will be max
10 for any input
```

680. Valid Palindrome II

https://leetcode.com/problems/valid-palindrome-ii/discuss/107718/Easy-to-Understand-Python-Solution (https://leetcode.com/problems/valid-palindrome-ii/discuss/107718/Easy-to-Understand-Python-Solution)

```
# Two cases if letters do not match at left and right:
# s[start+1] == s[stop] or s[start] == s[stop-1] or both
# e.g. ebcbb ec ecabbac ec bbcbe" , "abc"

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

        def isPalindrome(ss):
            begin = 0
            end = len(ss) - 1
            while begin < end:
                if ss[begin] != ss[end]:
                    return False
                begin = begin +1
                end = end -1
            return True

        left = 0
        right = len(s) - 1


        while left < right:
            if s[left] != s[right]:  # at most one deletion allowed
                return isPalindrome(s[left+1: right+1]) or isPalindrome(s[lef
t:right])

            left = left +1
            right = right -1
```

```
        return True

# Time Complexity = O(n)
# Space Complexity = O(n) due to slicing operation
```

---

721. Accounts Merge

```
from collections import defaultdict
class Solution(object):
    def accountsMerge(self, accounts):
        email_to_name = {}
        email_to_emails = defaultdict(set)

        # Build email to emails dict (value = set)
        # Build email to name dict
        for acc in accounts:
            name = acc[0]
            for email in acc[1:]:
                email_to_emails[acc[1]].add(email)
                email_to_emails[email].add(acc[1])
                email_to_name[email] = name

        # DFS
        visited = set()
        result = []

        for email in email_to_emails:
            if email not in visited:
                visited.add(email)

                stack = []
                stack.append(email)

                component = []
```

```
                while stack:
                    node = stack.pop()
                    component.append(node)

                    for neighbor in email_to_emails[node]:
                        if neighbor not in visited:
                            visited.add(neighbor)
                            stack.append(neighbor)

                result.append([email_to_name[email]] + sorted(component))

        return result

# Time Complexity = O(Summation of a[i] * log a[i]) where a[i] = length of ac
count i
# Without the log factor, this is the complexity to build the graph and searc
h for each component. The
# log factor is for sorting each component at the end.

# Space Complexity = O(Summation of a[i]) where a[i] = length of account i, t
he space used by email_to_emails dict
```

766. Toeplitz Matrix

```
class Solution:
    def isToeplitzMatrix(self, matrix: List[List[int]]) -> bool:
        for r, row in enumerate(matrix):
            for c, val in enumerate(row):
                if r > 0 and c > 0 and matrix[r-1][c-1] != matrix[r][c]:
                    return False

        return True

# Time complexity = O(M*N)
# Space Complexity = O(1)
```

767. Reorganize String

https://www.youtube.com/watch?v=xx8xZSaPZQ4 (https://www.youtube.com/watch?v=xx8xZSaPZQ4)

from collections import Counter S = "ababbaa" print(Counter(S).most_common(1))

O/P:

[('a', 4)]

```
# No solution: if freq of most common char > (len(string) + 1) //2 (aab -> ab
a sol exists (+1 term))
# Otherwise sol exists

# Arrange string by freq of chars in descending order (a 3 b 2 -> aaabb)
# Place higher half freq chars at even index and lower half freq chars at odd
index -> a3 b2

from collections import Counter
class Solution:
    def reorganizeString(self, S: str) -> str:
        if (Counter(S).most_common(1)[0][1]) > (len(S)+1)//2: # No solution
            return ""

        a = []
        for c, x in sorted((S.count(x), x) for x in set(S)):  # ascending ord
er, not descending
            a.extend(c * x)


        ans = [None] * len(S)
        h = len(S)//2 # in python 3 explicitly make h integer, len(S)/2 will
return float

        ans[::2], ans[1::2] = a[h:], a[:h]

        return "".join(ans)
```

```
# Time Complexity = O(n log n) -> sorted (), O(n log n) -> most_common(), O
(n) -> filling ans
# Space Complexity = O(n) -> a, O(n) -> ans

# intuition for placing higher half freq chars at even index => if the length
of string is odd and higher freq chars are populated at even indexes, greater
room for spreading out
```

---

704. Binary Search

```
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        low = 0
        high = len(nums) - 1

        while low<=high: # imp: <=
            middle = low + (high-low)//2

            if  target == nums[middle]:
                return middle

            if  target < nums[middle]:
                high = middle-1

            elif target > nums[middle]:
                low = middle + 1

        return -1

# Time Complexity = O(log n)
# Space Complexity = O(1)
```

785. Is Graph Bipartite?

https://www.geeksforgeeks.org/bipartite-graph/ (https://www.geeksforgeeks.org/bipartite-graph/)

```
# There are no self edges or parallel edges
# Graph can be disconnected => each of its connected components should be bip
artite
class Solution:
    def isBipartite(self, graph: List[List[int]]) -> bool:
        color = {}


        for node in range(len(graph)): # Iterate for each of connected compon
ents

            if node not in color:
                q = []
                color[node] = 0 # can be 0 or 1 as we just need to flip it fo
r neighbors
                q.append(node)

                while q:
                    popped_node = q.pop(0) # BFS

                    for neighbor in graph[popped_node]:
                        if neighbor not in color:
                            color[neighbor] = color[popped_node] ^ 1  # Bitwi
se XOR: 1->0 and 0->1
                            q.append(neighbor)
                        elif color[neighbor] == color[popped_node]:
                            return False
        return True
```

796. Rotate String

https://www.youtube.com/watch?v=4jY57Ehc14Y (https://www.youtube.com/watch?v=4jY57Ehc14Y)

```
class Solution(object):
    def rotateString(self, A, B):
        return len(A) == len(B) and B in A+A

# Time Complexity = O(n^2), searching a substring in a string without special
ized algos such as KMP is n^2
# Space Complexity = O(2n)
```

836. Rectangle Overlap

```
class Solution:
    def isRectangleOverlap(self, rec1: List[int], rec2: List[int]) -> bool:
        start_x = 0
        start_y = 1
        end_x = 2
        end_y = 3

        # Conditions for overlap
        if rec1[start_x] >= rec2[end_x]: return False
        if rec1[end_x] <= rec2[start_x]: return False
        if rec1[start_y] >= rec2[end_y]: return False
        if rec1[end_y] <= rec2[start_y]: return False

        return True

# Time and Space Complexity = O(1)
```

---

710. Random Pick with Blacklist

We can use rejectionsampling but since the question mentions "Optimize it such that it minimizes the call to system’s Math.random()." - avoid rejection sampling

https://leetcode.com/problems/random-pick-with-blacklist/discuss/144624/Java-O(B)-O(1)-HashMap (https://leetcode.com/problems/random-pick-with-blacklist/discuss/144624/Java-O(B)-O(1)-HashMap)

---

867. Transpose Matrix

```
class Solution:
    def transpose(self, A: List[List[int]]) -> List[List[int]]:
        return [[A[j][i] for j in range(len(A)) ] for i in range(len(A[0]))]

class Solution(object):
    def transpose(self, A):
        R, C = len(A), len(A[0])
        ans = [[None] * R for _ in xrange(C)]
        for r, row in enumerate(A):
            for c, val in enumerate(row):
                ans[c][r] = val
        return ans

# A=transpose = list(map(list, zip(*A))
```

---

470. Implement Rand10() Using Rand7()

Two cases for this type of question:

1) M > N (this question) 2) M < N (rejection sampling) or (using map if N is a multiple of M)

https://leetcode.com/problems/implement-rand10-using-rand7/discuss/338395/In-depth-straightforward-detailed-explanation.-Java-Solution (https://leetcode.com/problems/implement-rand10-using-rand7/discuss/338395/In-depth-straightforward-detailed-explanation.-Java-Solution).

See solution section of this problem

Extra Reading: (the product or sum of two uniform distributions is not uniform) https://www.youtube.com/watch?v=5nqHLvWh1Q8 (https://www.youtube.com/watch?v=5nqHLvWh1Q8) https://www.noudaldenhoven.nl/wordpress/?p=117 (https://www.noudaldenhoven.nl/wordpress/?p=117) https://www.quora.com/What-is-the-intuition-behind-the-acceptance-rejection-method-the-method-used-to-generate-random-numbers-for-specific-probability-distribution (https://www.quora.com/What-is-the-intuition-behind-the-acceptance-rejection-method-the-method-used-to-generate-random-numbers-for-specific-probability-distribution)

```
class Solution:
    def rand10(self):
        """
        :rtype: int
        """
        r = 49

        while(r>40):
            column = rand7()
            row = rand7()
            r = (row - 1) * 7 + column



        result = (r-1) % 10 + 1
        return result

# Time Complexity: O(1)average, but O(∞) worst case.
# Space Complexity: O(1)

Rejection Sampling is a Geometric distribution, so to calculate the expected
value for the number of calls to rand7(), there is a very simple formula: E=
1/p, so E=1/(40/49)=49/40, and for every success we need call 2 times, so E=2
* 49/40=49/20=2.45
```

876. Middle of the Linked List

```
# Two pointer: one slow and another fast, fast moves twice than slow
class Solution:
    def middleNode(self, head: ListNode) -> ListNode:
        curr = head

        while head and head.next: # imp
            curr = curr.next
            head = head.next.next

        return curr
```

528. Random Pick with Weight

http://blog.gainlo.co/index.php/2016/11/11/uber-interview-question-weighted-random-numbers/ (http://blog.gainlo.co/index.php/2016/11/11/uber-interview-question-weighted-random-numbers/)

1.W is the sum of all the weights (length of the horizontal line) 2. Get a random number R from [0, W] (randomly select a point) => This ensures all points are equally likely 3. Go over each element in order and keep the sum of weights of visited elements. Once the sum is larger than R, return the current element. This is finding which area includes the point.

https://leetcode.com/problems/random-pick-with-weight/discuss/154475/Python-1-liners-using-builtin-functions (https://leetcode.com/problems/random-pick-with-weight/discuss/154475/Python-1-liners-using-builtin-functions)

```
# Weighted Sampling
# Naive soln (extra space)=> expand list by freq and choose random element
e.g. [1,2] -> [0,1,1]

# Prefix sum + Binary search

import bisect
import random
class Solution:

    def __init__(self, w: List[int]): # Time & Space Complexity = O(n)
        self.w = w
        self.prefix_sum = [ 0 ] * (len(self.w))
        self.prefix_sum[0] = self.w[0]
        for i in range(1,len(self.w)):
             self.prefix_sum[i] = self.prefix_sum[i-1] + self.w[i]


    def pickIndex(self) -> int: # Space Complexity = O(1)

        random_number = random.randint(1,self.prefix_sum[-1]) # Time Complexi
ty = O(1)

        result = bisect.bisect_left(self.prefix_sum, random_number) # Time Co
mplexity = O(log n)

        return result
```

896. Monotonic Array

```
# e.g. [6,5,4,4] or [2,3,3,4] -> True

class Solution:
    def isMonotonic(self, A: List[int]) -> bool:

        asc = desc = True

        # if adj values are same, do nothing and iterate ahead
        for i in range(len(A)-1):
            if A[i] < A[i+1]:
                desc = False
            if A[i] > A[i+1]:
                asc = False

        return asc or desc

# Time Complexity = O(n)
# Space Complexity = O(1)
```

938. Range Sum of BST

```
# Binary Search Tree
# if node value > L -> move left
# if node value < R -> move right

class Solution:
    def rangeSumBST(self, root: TreeNode, L: int, R: int) -> int:
        sum = 0
        stack = [root]
        while stack:
            node = stack.pop()

            if L <= node.val <= R:
                sum = sum + node.val


            if node.val > L and node.left:
                stack.append(node.left)
            if node.val < R and node.right:
                stack.append(node.right)

        return sum

# Time Complexity = O(n) worst case, though we skip nodes whose values lie ou
tside [L,R]
# Space Complexity = O(height of tree)
```

939. Minimum Area Rectangle

https://leetcode.com/problems/minimum-area-rectangle/discuss/240341/Python-O(n2)-easy-to-understand.-Good-for-beginners (https://leetcode.com/problems/minimum-area-rectangle/discuss/240341/Python-O(n2)-easy-to-understand.-Good-for-beginners)

```
# For each pair of points in the array, consider them to be the long diagonal
of a potential
# rectangle. We can check if all 4 points are there using a Set.
# There could be duplicate points
# area could be zero in which case it is not a rectangle

import sys
class Solution:
    def minAreaRect(self, points: List[List[int]]) -> int:
        min_area = sys.maxsize
        s = set()
        for x,y in points:
            s.add((x,y))

        for x1,y1 in s:
            for x2,y2 in s:
                if x1 > x2 and y1 > y2: # only look at pairs already not seen
(points are sorted)
                    if x1 == x2 or y1 == y2: # area becomes zero
                        continue
                    else:
                        if (x1,y2) in s and (x2,y1) in s:
                            area = abs(x2-x1) * abs(y2-y1)
                            min_area = min(area, min_area)

        if min_area == sys.maxsize:
            return 0
```

```
        else:
            return min_area
```

---

953. Verifying an Alien Dictionary

```
# Two strings are lexicographically ordered if first non-matching char is in
order
class Solution:
    def isAlienSorted(self, words: List[str], order: str) -> bool:
        char_order = {c:i for i,c in enumerate(order)}
        char_order['#'] = -1 # padding char order should be lower than lowest
one to handle cases such as "apple, "app" -> False

        for i in range(len(words) -1): # Compare in batches of word i and wor
d i+1
            word1 = words[i]
            word2 = words[i+1]

            # Make the length of both strings equal with padding
            if len(word1) < len(word2):
                word1 = word1 + "#" * (len(word2) - len(word1))
            elif len(word1) > len(word2):
                word2 = word2 + "#" * (len(word1) - len(word2))

            for j in range(len(word1)): # both words are now of same length
                if word1[j] != word2[j]: # As soon as non-matching char is fo
und
                    if char_order[word1[j]] > char_order[word2[j]]:
                        return False
                    break # Imp: if the first non-matching char follows corre
ct order break the inner loop
        return True
```

```
# Time Complexity = O(nk) + O(d) where n = no. of words and k = max number of
chars in any string, and d = size of dictionary to store order
# Space Complexity = O(d)
```

---

973. K Closest Points to Origin

https://leetcode.com/problems/k-closest-points-to-origin/discuss/294389/Easy-to-read-Python-min-heap-solution-(-beat-99-python-solutions-) (https://leetcode.com/problems/k-closest-points-to-origin/discuss/294389/Easy-to-read-Python-min-heap-solution-(-beat-99-python-solutions-))

https://www.youtube.com/watch?v=4hkJBcW5Ruk (https://www.youtube.com/watch?v=4hkJBcW5Ruk)

https://runestone.academy/runestone/books/published/pythonds/Trees/BinaryHeapImplementation.html (https://runestone.academy/runestone/books/published/pythonds/Trees/BinaryHeapImplementation.html)

```
import heapq
class Solution:
    def kClosest(self, points: List[List[int]], K: int) -> List[List[int]]:
        heap = []
        for x,y in points:
            dist = -(x*x + y*y) # Top k smallest - Max heap
            heapq.heappush(heap, (dist,x,y))

            if len(heap) == K+1: # When heap size > K -> pop
                heapq.heappop(heap)

        return [[x,y] for dist,x,y in heap]

# Time Complexity = O(N log k)
# Space Complexity = O(k)
```

986. Interval List Intersections

```
# Max of starting point + Min of ending point
# Whichever point ends first, go to the next point in the same list

class Solution:
    def intervalIntersection(self, A: List[List[int]], B: List[List[int]]) ->
List[List[int]]:

        i=j=0
        result = []

        while i < len(A) and j < len(B):
            low = max(A[i][0], B[j][0])
            high = min(A[i][1], B[j][1])
            if low <= high: # result could be [5,5]
                result.append([low,high])

            if A[i][1] < B[j][1]:
                i = i+1
            else:
                j = j+1

        return result

# Time Complexity = O(m+n) where m,n = size of list A and B
# Space Complexity = O(1), if space required by result is not considered, els
e O(m+n)
```

994. Rotting Oranges

```
   #Collect list of all rotten oranges
    # Add them to queue with level 0
    #Deque and traverse neighbors, if neighbor == 1 then make it 2 and add to
queue with level incremented *

            #This is not simple BFS, as all rotten one will contaminate neigh
bors parallely
```

1060. Missing Element in Sorted Array

See solution

```
# Build missing list
# i.e. number of elements missing upto each index in the list: nums[index] - nums[0] - index
# Since this missing list is directly a function of index and constant value at nums[0], use lambda fn
# else time and space complexity will be O(n)

# Find the first 'index' in the missing list where the missing list element >= k

# Find the kth smallest element: nums[index-1] + k - missing[index -1]

# E.g. [4,7,9,10] k = 1 : ans => 5, if k= 3 : ans => 8

# Array is sorted: binary search

import bisect
class Solution:
    def missingElement(self, nums: List[int], k: int) -> int:

        missing = lambda index : nums[index] - nums[0] - index

        # Edge case: If kth missing number is larger than the last element of the array
        if k > missing(len(nums) - 1):
            return nums[-1] + k - missing(len(nums)- 1)
```

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

        while left < right:
            pivot = left + (right-left) // 2

            if missing(pivot) < k:
                left = pivot + 1
            else:
                right = pivot

        return nums[left-1] + k - missing(left - 1)



# Time Complexity  = O(log n)
# Space Complexity = O(1)
```

1091. Shortest Path in Binary Matrix

https://leetcode.com/problems/shortest-path-in-binary-matrix/discuss/312827/Python-Concise-BFS (https://leetcode.com/problems/shortest-path-in-binary-matrix/discuss/312827/Python-Concise-BFS)

```
# Shortest path => BFS
class Solution:
    def shortestPathBinaryMatrix(self, grid: List[List[int]]) -> int:
        n = len(grid)
        if grid[0][0] != 0 or grid[n-1][n-1] != 0: # top-left and bottom-righ
t should be zero
            return -1


        directions = [[-1,-1], [-1,0], [-1,1], [0,-1], [0,1], [1,-1], [1,0],
[1,1]]

        q = []
        q.append([0,0,1]) # co-ordinate of starting element and initial count
er value
        grid[0][0] = 1 # add to queue and mark as visited
        while q:
            i,j,counter = q.pop(0)

            if i == j == n-1: # Stopping condition
                return counter

            for direction in directions:
                next_i, next_j = i + direction[0], j + direction[1]
                if 0 <= next_i < n and 0 <= next_j < n and grid[next_i][next_
j] == 0:
                    # add to queue and mark as visited
                    q.append([next_i, next_j, counter+1]) # Imp: calculate le
```

```
vel/counter and append
                    grid[next_i][next_j] = 1 # Imp
                     # For BFS, you need to mark a node the first time you 's
ee' it instead of when you 'visit'
# it.The reason is, if C is neighboring to A and B, when we visit A, we will
put C to the queue. If we didn't mark # C here, next time when we visit B, we
will put C again to the queue. There could be quite a lot redundant search
# this works as visited matrix - set the visited grid as non-empty to avoid r
evisiting
        return -1
```

1249. Minimum Remove to Make Valid Parentheses

```
# 1: extra ')' - when no complement exists in stack  and 2: extra '(' remaini
ng in stack at the end

class Solution:
    def minRemoveToMakeValid(self, s: str) -> str:
        indexes_to_remove = set()
        stack = []

        # Get indices of all extra ')' and '('
        for i, c in enumerate(s):
            if c not in "()":
                continue

            if c == "(":
                stack.append(i)
            elif c == ")":
                if len(stack) == 0: # extra ')'
                    indexes_to_remove.add(i)
                else:
                    stack.pop()

        indexes_to_remove = indexes_to_remove.union(set(stack)) # extra '(' r
emaining in stack

        result = []
        for i, c in enumerate(s):
            if i not in indexes_to_remove:
                result.append(c)
```

```
        return "".join(result)

# Time Complexity = O(n)
# Space Complexity = O(n)
```

1428. Leftmost Column with at Least a One

```
# Start from top-right
# encounter 0 => move down, encounter 1=> move left
# Edge case : if all elements are 0

class Solution:
    def leftMostColumnWithOne(self, binaryMatrix: 'BinaryMatrix') -> int:
        rows,cols = binaryMatrix.dimensions()

        current_col = cols - 1
        current_row = 0

        while current_row < rows and current_col >= 0:
            if binaryMatrix.get(current_row, current_col) == 0:
                current_row = current_row + 1
            else:
                current_col = current_col - 1

        # if all elements are zero
        if current_col == cols - 1:
            return -1
        else:
            return current_col + 1 #Imp: +1

# Time Complexity = O(row + cols)
# Space Complexity = O(1)
```