69. Максимальная длина из "1" после удаления одного "0"

```
# Нужно вывести длину максимальной подпоследовательности из "1" при
условии,
# что какой-то нолик можно удалить.

def max_len_of_ones(arr):
    length = 0
    max_length = 0
    mode = False

    for i in range(len(arr)):
        if arr[i] == 0:
            if mode:
                if length > max_length:
                    max_length = length
                i = lastzeroindex
                length = 0
                mode = False
            else:
                lastzeroindex = i
                mode = True
        else:
            length += 1

    if length > max_length:
        max_length = length
    return max_length
```

70. Найти подотрезок с наименьшей суммой по модулю

```
def find_min_abs_subarray(arr):
    max_sum, min_sum, curr_max, curr_min = float('-inf'),
float('inf'), 0, 0
    for num in arr:
        curr_max = max(curr_max + num, num)
        max_sum = max(max_sum, curr_max)
        curr_min = min(curr_min + num, num)
        min_sum = min(min_sum, curr_min)
    return abs(min_sum) if abs(min_sum) < max_sum else max_sum

# Дан целочисленный массив из n элементов.
# Найти непустой подотрезок с наименьшей по модулю суммой. O(nlogn)

def makeprefixsum(nums):
    prefix = []
    prefix.append((0, -1))
    for i in range(1, len(nums) + 1):
        elem = (abs(prefix[i-1][0] + nums[i-1]), i - 1)
        prefix.append(elem)
    return prefix
```

```
def find_min_abs_sum(arr):
    prefix = sorted(makeprefixsum(arr))
    min_diff = abs(arr[0] + arr[1])
    start = 0
    end = len(arr)
    for i in range(1, len(arr) + 1):
        diff = prefix[i][0] - prefix[i-1][0]
        if diff < min_diff:
            min_diff = diff
            start = prefix[i-1][1]
            end = prefix[i][1]
    return start + 1, end

find_min_abs_sum([-21, -24, -33, -6, -100])
```

```
(0, 1)
```

71. Найти подотрезок с наибольшей суммой

```
def getMaxSubSum(arr): # O(n)
    max_sum = 0
    max_l = 0
    max_r = 0
    partial_sum = 0

    for i, item in enumerate(arr):  # для каждого элемента массива
        partial_sum += item  # добавляем значение элемента к
partialSum
        if partial_sum > max_sum:  # запоминаем максимум на данный
момент
            max_sum = partial_sum
        if partial_sum < 0:
            partial_sum = 0  # ноль если отрицательное

    return max_sum

getMaxSubSum([1, 7, -3, 8, -6, 8])
```

```
15
```

72. Определить номер первой колонки, в которой есть хоть одна единица

```
# Дана квадратная матрица NxN, заполненная нулями и единицами таким
образом,
# что в каждой строке все нули располагаются левее всех единиц
# (возможны строки, состоящие полностью из нулей или полностью из
единиц).
# Требуется определить номер первой колонки, в которой есть хоть одна
единица.

def check(matrix):
```

```
    left = 0
    right = len(matrix[0]) - 1
    mid = (left + right) // 2
    for i in matrix:
        while left < right:
            if i[mid] == 1:
                right = mid
            else:
                left = mid + 1
            mid = (left+right) // 2
        right = left
        left = 0
        mid = (left+right) // 2
    return right

print(check([[0,0,1],[0,0,1],[0,0,1]]))
```

```
2
```

73. Можно ли получить одну строку из другой за <= 1 одно исправление

```
# Реализовать функцию, проверяющую,
# можно ли одну строку получить из другой не более, чем за одно
исправление (удаление, добавление, изменение символа)
# def one_edit_apart(s1: str, s2: str) -> bool

def oneEditApart(s1: str, s2: str) -> bool:

    if s1 == s2:
        return True

    len_1 = len(s1)
    len_2 = len(s2)

    if len_2 == len_1:
        flag = 0
        for i in range(len_1):
            if s1[i] != s2[i]:
                if flag == 1:
                    return False
                else:
                    flag = 1
        return True

    if abs(len_1 - len_2) == 1:
        if len_1 < len_2:
            len_1, len_2 = len_2, len_1
            s1, s2 = s2, s1

        i = 0
        flag = 0
```

```
        while i < len_1 - 1:
            if s1[i + flag] != s2[i]:
                if flag == 1:
                    return False
                else:
                    flag = 1
                    continue
            i += 1
        return True
    else:
        return False
```

74. Найти подстроку, которая совпадает с точностью до перестановки

```
# Найти в тексте подстроку, что она совпадает с точностью до
перестановки букв(анаграмма)

def find_substr(line: str, pattern: str):
    dict1 = {}
    dict2 = {}
    for c in pattern:
        if (dict1.get(c) != None):
            dict1[c] += 1
        else:
            dict1[c] = 1
    print(dict1)
    start = 0
    stop = len(pattern)
    for i in range(start,stop):
        if dict2.get(line[i]) != None:
            dict2[line[i]] += 1
        else:
            dict2[line[i]] = 1
    while (stop < len(line)-1):
        print(dict2)
        if dict2 == dict1:
            return start,stop-1
        else:
            if dict2[line[start]] == 1:
                del dict2[line[start]]
            else:
                dict2[line[start]] -= 1
            start += 1
            stop +=1
            if dict2.get(line[stop-1]) != None:
                dict2[line[stop-1]] += 1
            else:
                dict2[line[stop-1]] = 1
    return False
```

75. Перевернуть int

```
def reverse_number(n):
    r = 0
    if n > 0:
        while n > 0:
            r *= 10
            r += n % 10
            n //= 10
        return r
    else:
        n = abs(n)
        while n > 0:
            r *= 10
            r += n % 10
            n //= 10
        r = r - 2*r
        return r

print(reverse_number(1534236469))
```

```
9646324351
```

76. Найти наибольшую сумму в дереве

```
res_max = float('-inf')

def findMax(root):
    if not root:
        return 0
    left_value = findMax(root.left)
    right_value = findMax(root.right)

    left_max = left_value + root.data
    right_max = right_value + root.data
    all_max = left_value + right_value + root.data
    tmp_max = max(left_max, right_max, all_max)
    if tmp_max > res_max:
        res_max = tmp_max
    return tmp_max
```

77. Найти максимальное число постояльцев, которые одновременно проживали в гостинице

Даны даты заезда и отъезда каждого гостя. Для каждого гостя дата заезда строго раньше даты отъезда (то есть каждый гость останавливается хотя бы на одну ночь). В пределах одного дня считается, что сначала старые гости выезжают, а затем въезжают новые. Найти максимальное число постояльцев, которые одновременно проживали в гостинице (считаем, что измерение количества постояльцев происходит в конце дня). sample = [(1, 2), (1, 3), (2, 4), (2, 3),]

```python
from collections import defaultdict
from typing import List

def max_num_guests(guests: List[tuple]) -> int:
    res = 0

    # для каждого дня посчитаем, сколько приехало и сколько отъехало
    arriving = defaultdict(int)
    leaving = defaultdict(int)

    for guest in guests:  # O(n)
        arriving[guest[0]] += 1
        leaving[guest[1]] += 1

    current = 0
    # едем по дням в порядке увеличения, добавлем приехавших и
убавляем уехавших,
    # считаем сколько стало
    for day in
sorted(set(arriving.keys()).union(set(leaving.keys()))):  #
O(n*log(n)) + O(n)
        current -= leaving[day]
        current += arriving[day]

        if current > res:
            res = current

    return res
```

78. Minimum Window Substring

- Input: s = "ADOBECODEBANC", t = "ABC"
- Output: "BANC"

```python
# Time: O(m+n)

class Solution:
    def minWindow(self, s: str, t: str) -> str:
        if t == '':
            return ''

        t_dict, window = {}, {}

        for c in t:
            t_dict[c] = 1 + t_dict.get(c, 0)

        have, need = 0, len(t_dict)
        res, resLen = [-1, -1], float('inf')
        l = 0

        for r in range(len(s)):
```

```
            c = s[r]
            window[c] = 1 + window.get(c, 0)

            if c in t_dict and window[c] == t_dict[c]:
                have += 1

            while have == need:
                # update our result
                if r - l + 1 < resLen:
                    resLen = r - l + 1
                    res = [l, r]

                # pop from left of window
                window[s[l]] -= 1
                if s[l] in t_dict and window[s[l]] < t_dict[s[l]]:
                    have -= 1

                l += 1

        l, r = res
        return s[l:r+1]
```

79. Binary Tree Zigzag Level Order Traversal

Given the root of a binary tree, return the zigzag level order traversal of its nodes' values. (i.e., from left to right, then right to left for the next level and alternate between).

- **Input:** root = [3,9,20,null,null,15,7]
- **Output:** [[3],[20,9],[15,7]]

```
class Solution:
    def zigzagLevelOrder(self, root):
        if not root:
            return []

        queue = deque([root])
        result, direction = [], 1

        while queue:
            level = []
            for i in range(len(queue)):
                node = queue.popleft()
                level.append(node.val)

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

            result.append(level[::direction])
```

```
            direction *= (-1)
        return result
```

80. Logger Rate Limiter

```
class Logger:
    def __init__(self):
        # [(timestamp, message)]
        self.messageQueue = collections.deque()
        self.messageSet = set()

    def shouldPrintMessage(self, timestamp: int, message: str) ->
bool:
        # Remove messages that are 10 secs from the current timestamp
        while self.messageQueue:
            headTimestamp, headMessage = self.messageQueue[0]
            if timestamp < headTimestamp + 10:
                break
            self.messageQueue.popleft()
            self.messageSet.remove(headMessage)

        if message in self.messageSet:
            return False

        self.messageQueue.append((timestamp, message))
        self.messageSet.add(message)
        return True
```

Rate_Limiter

81. Count Number of Occurrences in a Sorted Array

В отсортированном массивее найти количество вхождений

[1, 2, 2, 2, 2, 3, 4, 5, 6], 2 -> 4

```
# Time O(logn)
def count(arr, x, n):

    i = first(arr, 0, n-1, x, n)

    if i == -1:
        return i

    j = last(arr, i, n-1, x, n);

    return j-i+1;

def first(arr, low, high, x, n):
    if high >= low:
```

```
        mid = (low + high) // 2

        if (mid == 0 or x > arr[mid-1]) and arr[mid] == x:
            return mid
        elif x > arr[mid]:
            return first(arr, (mid + 1), high, x, n)
        else:
            return first(arr, low, (mid -1), x, n)
    return -1;


def last(arr, low, high, x, n):
    if high >= low:

        mid = (low + high) // 2;

        if(mid == n-1 or x < arr[mid+1]) and arr[mid] == x :
            return mid

        elif x < arr[mid]:
            return last(arr, low, (mid -1), x, n)
        else:
            return last(arr, (mid + 1), high, x, n)
    return -1
```

82. Minimize the Maximum Difference between Heights

Даны высоты N башен и значение K, либо увеличить, либо уменьшить высоту каждой башни на K (только один раз), где K > 0. Задача состоит в том, чтобы минимизировать разницу между высотами самой длинной и самой длинной при помощи добавления/вычитания K. Выведите разность самой высокой и самой низкой.

```
def profit(arr, k):
    n = (min(arr) + max(arr)) // 2
    new = []
    for i in arr:
        if max(arr) - min(arr) < k:
            return max(arr) - min(arr)
        elif i >= n:
            new.append(i - k)
        else:
            new.append(i + k)
    return max(new) - min(new)
```

83. Find smallest missing number in sorted array

```
def findFirstMissing(array, start, end):

    if (start > end):
        return end + 1
```

```
    if (start != array[start]):
        return start;

    mid = int((start + end) / 2)

    if (array[mid] == mid):
        return findFirstMissing(array, mid+1, end)

    return findFirstMissing(array, start, mid)
```

84. Minimum Operations to Make Array Equal

Дан массив с n положительными целыми числами. Нам нужно найти минимальное количество операций, чтобы сделать все элементы равными. Мы можем выполнять сложение, умножение, вычитание или деление с любой частью элемента массива.

```
# Time O(n)

def minOperation(arr, n):
    smallest_ele = float("inf")
    for i in arr:
        smallest_ele = min(smallest_ele, i)

    total_sum = sum(arr)

    result = total_sum - n*smallest_ele
    return result
```

Все, что нам нужно сделать в этой задаче, это использовать школьную математику. Нам нужно будет рассмотреть два случая: нечетное и четное n.

Сначала рассмотрим нечетный случай:

- **Нечетный случай**: рассмотрите 1, 3, 5, 7, 9, 11, 13, 15, 17, чтобы увидеть закономерность. Затем нам нужно сделать все числа равными 9. Сколько операций нам нужно? Для пары 7,11 нужно 2 операции, для 5,13 нужно 4 операции, затем 6 и 8. Всего нужно 2+4+6+8 = 20 операций. Как его вычислить в общем случае? Если n = 2k + 1, то нужно вычислить 2 + 4 + ... + 2k, что является суммой арифметической прогрессии и равно k*(k+1)* = (*n*n-1)//4 .
- **Четный случай**: рассмотрите 1, 3, 5, 7, 9, 11, 13, 15, чтобы увидеть закономерность. Затем нам нужно сделать все числа равными 8. Нам нужно 1 + 3+ 5 + 7 = 16 операций и в общем случае нам нужно 1 + 3 + ... + 2k-1 = k*k* = *n*n// 4 операции. Наконец, мы можем записать это как одну формулу, используя округление. n*n/4

85. Remove All Occurrences of a Substring

- Input: s = "axxxxyyyyb", part = "xy"
- Output: "ab"

```
class Solution:
    def removeOccurrences(self, s: str, part: str) -> str:
        stack = []
        for ch in s:
            stack.append(ch)
            if "".join(stack[-len(part):]) == part:
                for _ in range(len(part)): stack.pop()
        return "".join(stack)
```

86. Find Smiles Position

Найти позиции смайлов в строке. Смайл начинается с ':-', потом последовательность ')' или '('

```
def find_smiles(text):
    smiles = []
    i = 0
    while i < len(text):
        if text[i:i+2] == ":-":
            j = i+2
            if text[j] == "(":
                while j < len(text) and text[j] == "(":
                    j += 1
                if j > i+2:
                    smiles.append((i, j-1))
            elif text[j] == ")":
                while j < len(text) and text[j] == ")":
                    j += 1
                if j > i+2:
                    smiles.append((i, j-1))
            i = j
        else:
            i += 1
    return smiles

print(find_smiles(':-(((:-('))
print(find_smiles(':-()'))
print(find_smiles(''))
```

```
[(0, 4), (5, 7)]
[(0, 2)]
[]
```

87. Здания, которые могут увидеть закат

Дан массив, представляющий высоты зданий. В массиве есть здания слева направо, как показано на диаграмме ниже, подсчитайте количество

зданий, обращенных к закату. Предполагается, что высоты всех зданий различны.

```
def countBuildings(arr, n):
    count = 1

    curr_max = arr[0]
    for i in range(1, n):
        if arr[i] >= curr_max:

            count += 1
            curr_max = arr[i]

    return count
```

88. Здания, которые могут увидеть океан

В ряду n зданий. Вам дан целочисленный массив высот размера n, представляющий высоты зданий в строке.

Океан находится справа от зданий. Здание имеет вид на океан, если здание может беспрепятственно видеть океан. Формально здание имеет вид на океан, если все здания справа от него имеют меньшую высоту.

Возвращает список индексов (с индексом 0) зданий с видом на океан, отсортированных в порядке возрастания.

```
class Solution:
    def findBuildings(self, heights: List[int]) -> List[int]:
        results = []

        prev_max = None

        for i in range(len(heights) - 1, -1, -1):
            height = heights[i]

            if not prev_max:
                prev_max = height
                results.append(i)
            else:
                if height > prev_max:
                    results.append(i)
                prev_max = max(prev_max, height)


        return sorted(results)
```

89. Сериализация/десериализация BST

```
class Codec:
    def serialize(self, root):
        l = []
```

```
        def preOrder(root):
            if root:
                l.append(root.val)
                preOrder(root.left)
                preOrder(root.right)
        preOrder(root)
        return ' '.join(map(str, l))

    def deserialize(self, data):
        vals = collections.deque([int(val) for val in data.split()])

        def buildTree(vals, minVal, maxVal):
            while vals and minVal < vals[0] < maxVal:
                val = vals.popleft()
                root = TreeNode(val)
                root.left = buildTree(vals, minVal, val)
                root.right = buildTree(vals, val, maxVal)
                return root
        return buildTree(vals, float('-inf'), float('inf'))
```

90. Развернуть матрицу на 90 градусов

```
from typing import List

def rotate(matrix: List[List[int]]) -> None:
        lMat = len(matrix)
        for i in range(lMat):
            for j in range(i + 1,lMat):
                matrix[i][j], matrix[j][i] = matrix[j][i], matrix[i]
[j]
        for i in range(lMat):
            matrix[i].reverse()
```

91. Генерация спиральной матрицы

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

        i = 0
        j = 1
        while j <= n*n:
            for rightcol in range(i, n-i):
                ans[i][rightcol] = j
                j += 1

            downrow = -1
            for downrow in range(i+1, n-i):
                ans[downrow][rightcol] = j
                j += 1
            if downrow == -1:
                break
            icol = -1
```

```
            for icol in range(rightcol-1, -1+i, -1):
                ans[downrow][icol] = j
                j += 1
            if icol == -1:
                break
            for uprow in range(downrow-1, i, -1):
                ans[uprow][icol] = j
                j += 1
            i += 1
        return ans
```

92. Найти первое вхождение одной строки в другую

```
def strStr(haystack: str, needle: str) -> int:
        if not needle:
            return 0;

        n = len(haystack)
        m = len(needle)

        for i in range(n-m+1):
            if haystack[i:m+i] == needle:
                return i
        return -1
```

93. Найти первое и последнее вхождение в отсортированном списке

```
def findFirstOccurrence(nums, target):
        left, right = 0, len(nums) - 1
        firstOccurrence = -1
        while left <= right:
            middle = left + (right - left) // 2
            if target == nums[middle]:
                firstOccurrence = middle
                right = middle - 1
            elif target < nums[middle]:
                right = middle - 1
            else:
                left = middle + 1
        return firstOccurrence

def findLastOccurrence(nums, target):
        left, right = 0, len(nums) - 1
        lastOccurrence = -1
        while left <= right:
            middle = left + (right - left) // 2
            if target == nums[middle]:
                lastOccurrence = middle
                left = middle + 1
            elif target < nums[middle]:
                right = middle - 1
            else:
```

```
                left = middle + 1
        return lastOccurrence

def searchRange(nums: List[int], target: int) -> List[int]:
    return [findFirstOccurrence(nums, target),
findLastOccurrence(nums, target)]
```

94. Найти К ближайших элементов

Дан массив отсортированных целых чисел arr, функция возвращает k целых чисел, ближайших к x в массиве. Результат также должен быть отсортирован в порядке возрастания.

Целое число a ближе к x, чем целое число b, если:

- |a - x| < |b - x|, или
- |a - x| == |б - x| и a < б

```
def findClosestElements(arr: List[int], k: int, x: int) -> List[int]:
        i, j = 0, len(arr) - k
        while i < j:
            mid = (i & j) + ((i^j) >> 1)
            if x - arr[mid] > arr[mid + k] - x: # arr is sorted
                i = mid + 1
            else:
                j = mid

        return arr[i:i+k]
```

95. Найти количество не пустых подмассивов, сумма которых делится на k

```
def subarraysDivByK(nums: List[int], k: int) -> int:
        ans = 0
        prefix = 0
        count = [1] + [0] * (k - 1)

        for val in nums:
            prefix = (prefix + val) % k
            ans += count[prefix]
            count[prefix] += 1

        return ans
```

96. Проложить путь для аэропортов

```
def findItinerary(self, tickets: List[List[str]]) -> List[str]:
        def dfs(graph, airport, res):
            while graph[airport]:
                next = graph[airport].pop()
                dfs(graph, next, res)
            res.append(airport)

        graph = collections.defaultdict(list)
```

```
        for frm, to in tickets:
            graph[frm].append(to)
        for key in graph:
            graph[key].sort(reverse = True)
        res = []
        dfs(graph, 'JFK', res)
        return res[::-1]
```

97. Path Sum

Дан корень двоичного дерева и целое число targetSum, вернуть true, если дерево имеет путь от корня к листу, такой что суммирование всех значений на пути равно targetSum.

Лист — это узел без потомков.

```
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def hasPathSum(self, root, sum):
        if root is None:
            return False
        stack = [(root, sum)]
        while stack:
            node, _sum = stack.pop()
            if node.left is node.right is None and node.val == _sum:
                return True
            if node.left:
                stack.append((node.left, _sum - node.val))
            if node.right:
                stack.append((node.right, _sum - node.val))
        return False
```

98. 3 Sum

```
def three_sum(nums):
    n = len(nums)
    if n < 3:
        return []

    res = set()
    nums.sort()

    for i, v in enumerate(nums[:-2]):
        if i != 0 and v == nums[i-1]:
            continue

        l = i+1
        r = n-1
```

```
        target = -v

        while l < r:
            if nums[l] + nums[r] == target:
                res.add((v, nums[l], nums[r]))
                l += 1
                r -= 1
            elif nums[l] + nums[r] > target:
                r -= 1
            else:
                l += 1

    return list(map(list, res))
```

99. Найти индексы подмассива, сумма которого дает таргет

```
def subarray_with_given_sum(arr, given_sum):
    n = len(arr)
    start, end = 0, 0
    current_sum = arr[0]

    while end < n:
        if current_sum == given_sum:
            return arr[start:end+1]

        if current_sum < given_sum:
            end += 1
            if end < n:
                current_sum += arr[end]

        else:
            current_sum -= arr[start]
            start += 1

    return None
```