



1. Two Sum ↗

Given an array of integers `nums` and an integer `target`, return *indices of the two numbers such that they add up to target*.

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

You can return the answer in any order.

Example 1:

Input: `nums = [2,7,11,15]`, `target = 9`

Output: `[0,1]`

Explanation: Because `nums[0] + nums[1] == 9`, we return `[0, 1]`.

Example 2:

Input: `nums = [3,2,4]`, `target = 6`

Output: `[1,2]`

Example 3:

Input: `nums = [3,3]`, `target = 6`

Output: `[0,1]`

Constraints:

- $2 \leq \text{nums.length} \leq 10^4$
- $-10^9 \leq \text{nums}[i] \leq 10^9$
- $-10^9 \leq \text{target} \leq 10^9$
- **Only one valid answer exists.**

Follow-up: Can you come up with an algorithm that is less than $O(n^2)$ time complexity?

```
# Brute Force O(N^2) Solution
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        for i in range(len(nums)):
            for j in range(i+1, len(nums)):
                if nums[i] + nums[j] == target:
                    return [i, j]
```

11. Container With Most Water ↗

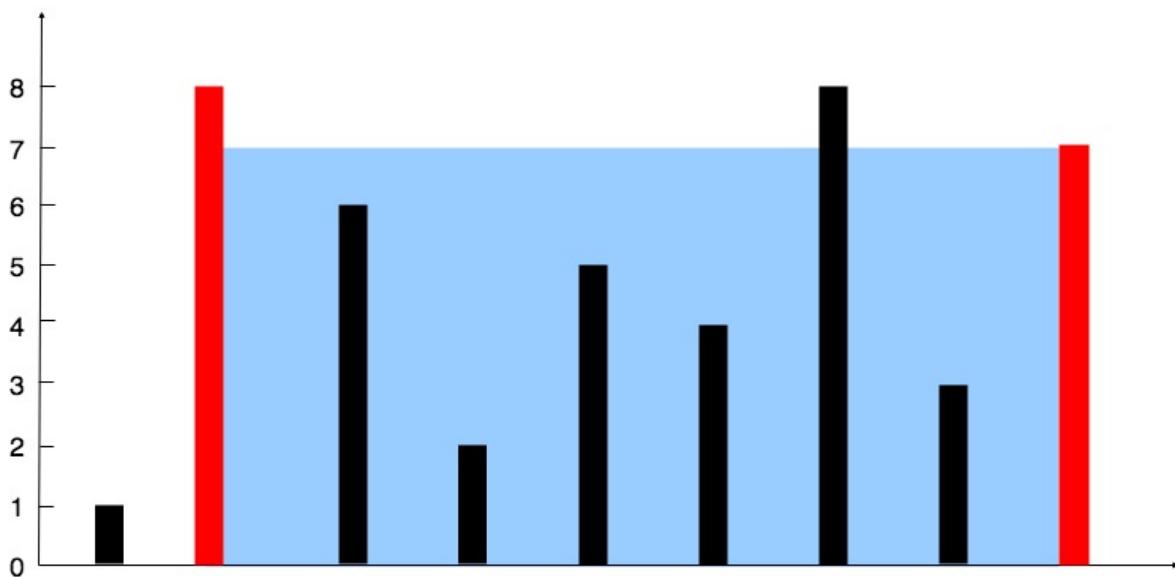
You are given an integer array `height` of length `n`. There are `n` vertical lines drawn such that the two endpoints of the i^{th} line are $(i, 0)$ and $(i, \text{height}[i])$.

Find two lines that together with the x-axis form a container, such that the container contains the most water.

Return *the maximum amount of water a container can store*.

Notice that you may not slant the container.

Example 1:



Input: height = [1,8,6,2,5,4,8,3,7]

Output: 49

Explanation: The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7].

Example 2:

Input: height = [1,1]

Output: 1

Constraints:

- $n == \text{height.length}$
- $2 \leq n \leq 10^5$
- $0 \leq \text{height}[i] \leq 10^4$

```
### intial solution
i , j = 0, len(height)-1
res = []
while i < j:
    if height[i] <= height[j]:
        print(height[i],height[j],height[i]* abs(i-j))
        res.append(height[i]* abs(i-j))
        j-=1
    else:
        res.append(height[j] * abs(i-j))
        print(height[j]* abs(i-j))
        i+=1
print(res)
```

20. Valid Parentheses ↗

Given a string `s` containing just the characters `'('`, `)'`, `{`, `}`, `[` and `]`, determine if the input string is valid.

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.

Example 1:

Input: s = "()"

Output: true

Example 2:

Input: s = "()[]{}"**Output:** true**Example 3:****Input:** s = "()"**Output:** false**Constraints:**

- $1 \leq s.length \leq 10^4$
- s consists of parentheses only '()'[]{}' .

```
# Traditional If Based Solution.
class Solution:
    def isValid(self, s: str) -> bool:
        res = []
        # If the string length is odd then obviously the paranthesis are not balanced
        if len(s) % 2 != 0:
            return False
        else:
            #
            for char in s:
                if char == ')' or char == '}' or char == ']':
                    if len(res) > 0:
                        top_val = res[-1]
                        if top_val == '(' and char == ')':
                            res.pop()
                        elif (top_val == '{' and char == '}'):
                            res.pop()
                        elif (top_val == '[' and char == ']'):
                            res.pop()
                        else:
                            return False
                    else:
                        return False
                else:
                    res.append(char)
        if len(res) == 0:
            return True
        return False
```

28. Implement strStr() ↗

Implement strStr() (<http://www.cplusplus.com/reference/cstring/strstr/>).

Given two strings `needle` and `haystack`, return the index of the first occurrence of `needle` in `haystack`, or `-1` if `needle` is not part of `haystack`.

Clarification:

What should we return when `needle` is an empty string? This is a great question to ask during an interview.

For the purpose of this problem, we will return `0` when `needle` is an empty string. This is consistent to C's `strstr()` (<http://www.cplusplus.com/reference/cstring/strstr/>) and Java's `indexOf()` ([https://docs.oracle.com/javase/7/docs/api/java/lang/String.html#indexOf\(java.lang.String\)](https://docs.oracle.com/javase/7/docs/api/java/lang/String.html#indexOf(java.lang.String))).

Example 1:

```
Input: haystack = "hello", needle = "ll"
Output: 2
```

Example 2:

```
Input: haystack = "aaaaa", needle = "bba"
Output: -1
```

Constraints:

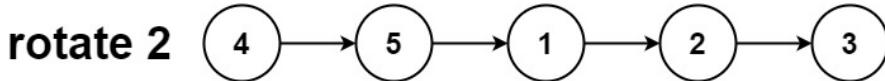
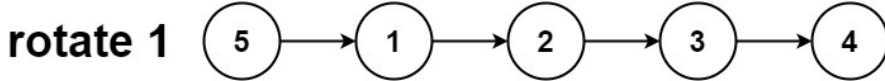
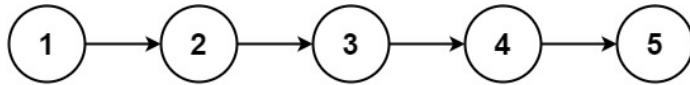
- $1 \leq \text{haystack.length}, \text{needle.length} \leq 10^4$
- `haystack` and `needle` consist of only lowercase English characters.

```
class Solution:
    def strStr(self, haystack: str, needle: str) -> int:
        if not len(needle):
            return 0
        elif needle in haystack:
            return haystack.index(needle)
        else:
            return -1
```

61. Rotate List ↗

Given the head of a linked list, rotate the list to the right by k places.

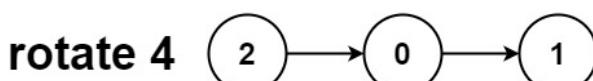
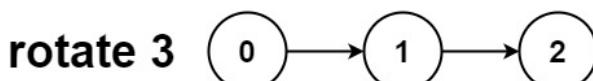
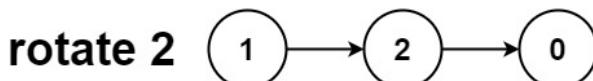
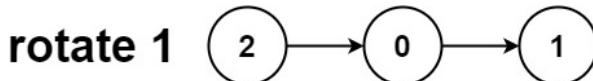
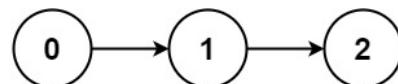
Example 1:



Input: head = [1,2,3,4,5], k = 2

Output: [4,5,1,2,3]

Example 2:



Input: head = [0,1,2], k = 4

Output: [2,0,1]

Constraints:

- The number of nodes in the list is in the range [0, 500].
- $-100 \leq \text{Node.val} \leq 100$
- $0 \leq k \leq 2 * 10^9$

```

class Solution:
    def rotateRight(self, head: Optional[ListNode], k: int) -> Optional[ListNode]:
        if head is None:
            return
        res = []
        # O(N) to iterate over the Linked List
        while head:
            res.append(head.val)
            head = head.next

        # O(N % K ≈ M)
        for i in range(k % len(res)):
            res = [res.pop()] + res

        root = ListNode(-1000)
        head = root
        # O(N) to build the List
        for val in res:
            if root.val == -1000:
                root.val = val
            else:
                head.next = ListNode(val)
                head = head.next
        return root

```

67. Add Binary ↴



Given two binary strings `a` and `b`, return *their sum as a binary string*.

Example 1:

```

Input: a = "11", b = "1"
Output: "100"

```

Example 2:

```

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

```

Constraints:

- $1 \leq a.length, b.length \leq 10^4$
- `a` and `b` consist only of '0' or '1' characters.
- Each string does not contain leading zeros except for the zero itself.

```
class Solution:
    def addBinary(self, a: str, b: str) -> str:
        return bin(int(a,2)+ int(b,2))[2:]
```

74. Search a 2D Matrix ↗

Write an efficient algorithm that searches for a value `target` in an $m \times n$ integer matrix `matrix`. This matrix has the following properties:

- Integers in each row are sorted from left to right.
- The first integer of each row is greater than the last integer of the previous row.

Example 1:

1	3	5	7
10	11	16	20
23	30	34	60

Input: `matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]]`, `target = 3`

Output: `true`

Example 2:

1	3	5	7
10	11	16	20
23	30	34	60

Input: matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 13
Output: false

Constraints:

- $m == \text{matrix.length}$
- $n == \text{matrix}[i].length$
- $1 \leq m, n \leq 100$
- $-10^4 \leq \text{matrix}[i][j], \text{target} \leq 10^4$

```
class Solution:
    def searchMatrix(self, matrix: List[List[int]], target: int) -> bool:
        for i in range(len(matrix)):
            if target in matrix[i]:
                return True
        return False
```

78. Subsets ↗

Given an integer array `nums` of **unique** elements, return *all possible subsets (the power set)*.

The solution set **must not** contain duplicate subsets. Return the solution in **any order**.

Example 1:

Input: nums = [1,2,3]
Output: [[], [1], [2], [1,2], [3], [1,3], [2,3], [1,2,3]]

Example 2:

Input: nums = [0]
Output: [[], [0]]

Constraints:

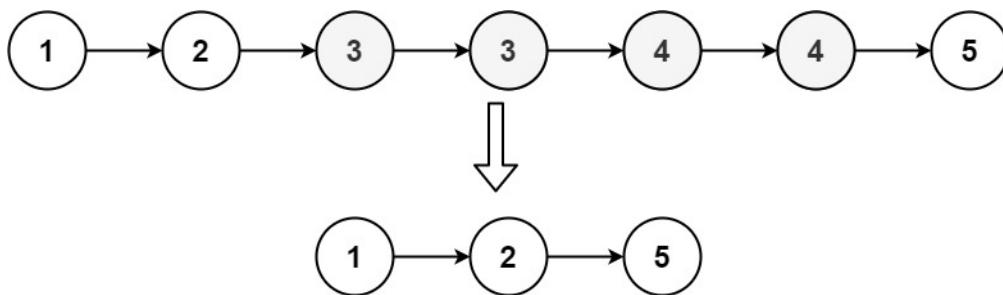
- $1 \leq \text{nums.length} \leq 10$
- $-10 \leq \text{nums}[i] \leq 10$
- All the numbers of `nums` are **unique**.

The most Craziest Problem i ve ever, seen. Initially i thought of a brute force solution, which obviously threwed me an Error.

82. Remove Duplicates from Sorted List II ↗

Given the head of a sorted linked list, *delete all nodes that have duplicate numbers, leaving only distinct numbers from the original list*. Return the linked list **sorted** as well.

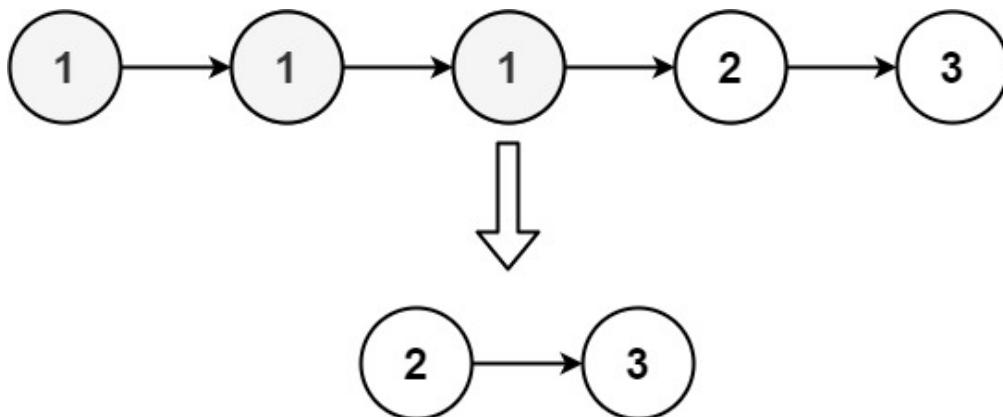
Example 1:



Input: head = [1,2,3,3,4,4,5]

Output: [1,2,5]

Example 2:



Input: head = [1,1,1,2,3]

Output: [2,3]

Constraints:

- The number of nodes in the list is in the range [0, 300].
- $-100 \leq \text{Node.val} \leq 100$
- The list is guaranteed to be **sorted** in ascending order.

```

# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def build_nodes(self, values):
        root = ListNode(-100000)
        head = root
        for k, v in sorted(values.items(), key=lambda item: item[0]):
            if root.val is -100000:
                root.val = k
            else:
                head.next = ListNode(k)
                head = head.next
        return root

    def deleteDuplicates(self, head: Optional[ListNode]) -> Optional[ListNode]:
        if head is None:
            return

        values = {}
        root = head

        while root:
            if root.val not in values:
                values[root.val] = 1
            else:
                values[root.val] += 1
            root = root.next

        value_c = values.copy()
        for k, v in value_c.items():
            if v > 1:
                del values[k]
        print(values)
        if len(values) > 0:
            return self.build_nodes(values)
        else:
            return

```

88. Merge Sorted Array ↗

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

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

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

Example 1:

Input: `nums1 = [1,2,3,0,0,0]`, $m = 3$, `nums2 = [2,5,6]`, $n = 3$

Output: `[1,2,2,3,5,6]`

Explanation: The arrays we are merging are `[1,2,3]` and `[2,5,6]`.

The result of the merge is `[1,2,2,3,5,6]` with the underlined elements coming from `nu`



Example 2:

Input: `nums1 = [1]`, $m = 1$, `nums2 = []`, $n = 0$

Output: `[1]`

Explanation: The arrays we are merging are `[1]` and `[]`.

The result of the merge is `[1]`.

Example 3:

Input: `nums1 = [0]`, $m = 0$, `nums2 = [1]`, $n = 1$

Output: `[1]`

Explanation: The arrays we are merging are `[]` and `[1]`.

The result of the merge is `[1]`.

Note that because $m = 0$, there are no elements in `nums1`. The 0 is only there to ensu



Constraints:

- `nums1.length == m + n`
- `nums2.length == n`
- $0 \leq m, n \leq 200$
- $1 \leq m + n \leq 200$
- $-10^9 \leq \text{nums1}[i], \text{nums2}[j] \leq 10^9$

Follow up: Can you come up with an algorithm that runs in $O(m + n)$ time?

```

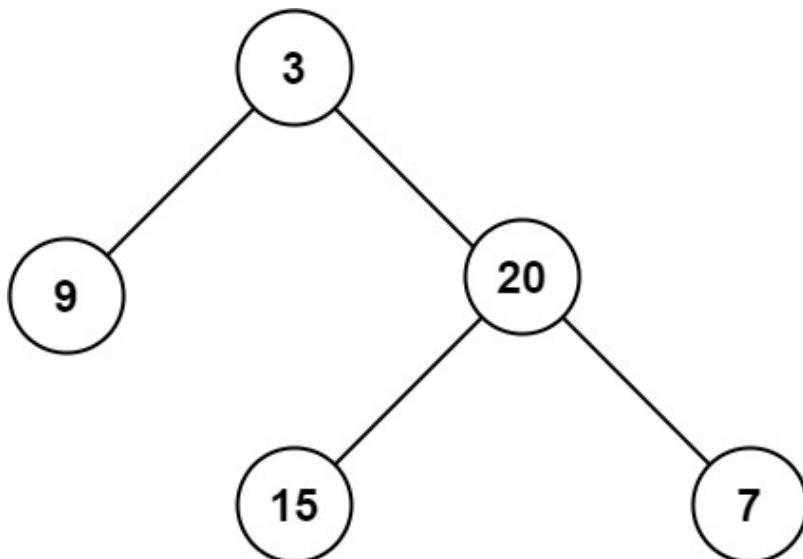
class Solution:
    def merge(self, nums1: List[int], m: int, nums2: List[int], n: int) -> None:
        """
        Do not return anything, modify nums1 in-place instead.
        """
        for i in range(len(nums2)):
            nums1[m+i] = nums2[i]
        nums1.sort()
    
```

104. Maximum Depth of Binary Tree ↗

Given the `root` of a binary tree, return *its maximum depth*.

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

Example 1:



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

Output: 3

Example 2:

Input: root = [1,null,2]

Output: 2

Constraints:

- The number of nodes in the tree is in the range $[0, 10^4]$.
- $-100 \leq \text{Node.val} \leq 100$

```
# 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 __init__(self):
        self.max_height = 0

    def depth(self, head, height):
        if (head is None):
            if (height > self.max_height):
                self.max_height = height

        return

        self.depth(head.left, height+1)

        self.depth(head.right, height+1)

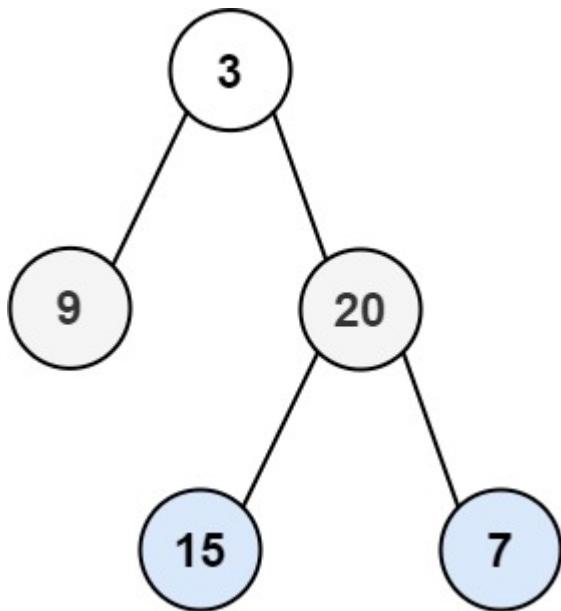
def maxDepth(self, root: Optional[TreeNode]) -> int:
    head = root
    if head is None:
        return 0
    self.depth(head, 0)

    return self.max_height
```

107. Binary Tree Level Order Traversal II ↗

Given the `root` of a binary tree, return *the bottom-up level order traversal of its nodes' values*. (i.e., from left to right, level by level from leaf to root).

Example 1:



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

Output: [[15,7],[9,20],[3]]

Example 2:

Input: root = [1]

Output: [[1]]

Example 3:

Input: root = []

Output: []

Constraints:

- The number of nodes in the tree is in the range [0, 2000].
- $-1000 \leq \text{Node.val} \leq 1000$

The Idea here is simple .. What to do?

We need to print the level order traversal in a reverse manner...

- Firstly we traverse the tree according to the requirement..
- Later we will maintain the track of the index of the list of list..
- Finally we update the list accordingly...

```
# Code to the Above Problem Statement
class Solution(object):
    def levelOrderBottom(self, root):
        r=[]
        temp=[]
        if not root:
            return r
        temp.append((root,0))
        while temp:
            root1,level=temp.pop(0)
            if len(r)==level:
                r.append([root1.val])
            else:
                r[level].append(root1.val)
            if root1.left:
                temp.append((root1.left,level+1))
            if root1.right:
                temp.append((root1.right,level+1))
        return r[::-1]
```

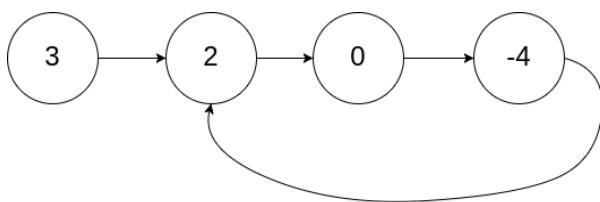
141. Linked List Cycle ↗

Given `head`, the head of a linked list, determine if the linked list has a cycle in it.

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

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

Example 1:

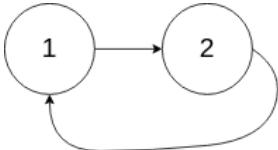


Input: `head = [3,2,0,-4], pos = 1`

Output: `true`

Explanation: There is a cycle in the linked list, where the tail connects to the 1st

Example 2:



Input: head = [1,2], pos = 0

Output: true

Explanation: There is a cycle in the linked list, where the tail connects to the 0th

Example 3:



Input: head = [1], pos = -1

Output: false

Explanation: There is no cycle in the linked list.

Constraints:

- The number of the nodes in the list is in the range $[0, 10^4]$.
- $-10^5 \leq \text{Node.val} \leq 10^5$
- pos is -1 or a **valid index** in the linked-list.

Follow up: Can you solve it using $O(1)$ (i.e. constant) memory?

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

class Solution:
    def hasCycle(self, head: Optional[ListNode]) -> bool:
        if head is None:
            return
        root = head
        address = {}
        flag = 0
        ind = 0
        while root:
            if root.next not in address:
                address[root.next] = ind+1

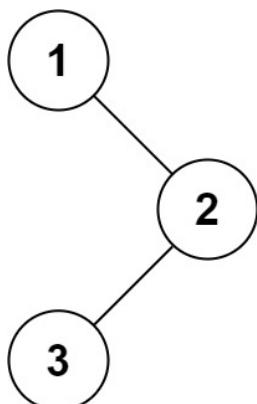
            else:
                flag = 1
                break
            root = root.next

        if flag:
            return True
        else:
            return False
```

144. Binary Tree Preorder Traversal ↗

Given the `root` of a binary tree, return *the preorder traversal of its nodes' values*.

Example 1:



Input: root = [1,null,2,3]
Output: [1,2,3]

Example 2:

Input: root = []
Output: []

Example 3:

Input: root = [1]
Output: [1]

Constraints:

- The number of nodes in the tree is in the range [0, 100].
- $-100 \leq \text{Node.val} \leq 100$

Follow up: Recursive solution is trivial, could you do it iteratively?

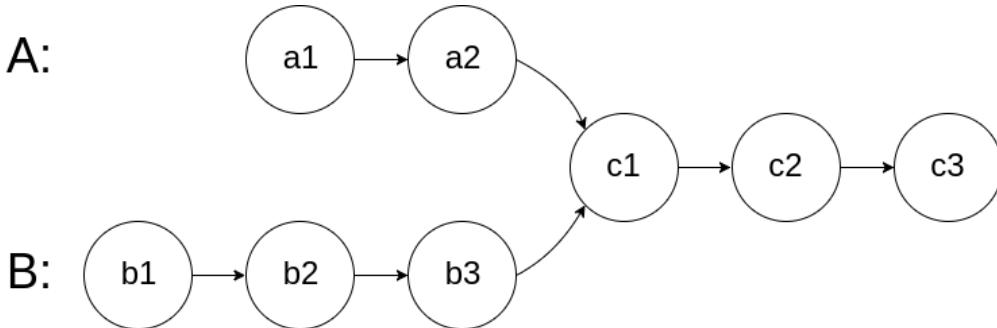
```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def __init__(self):
        self.arr = []
    def preorderTraversal(self, root: Optional[TreeNode]) -> List[int]:
        def test(root):
            if root is None:
                return
            self.arr.append(root.val)
            self.preorderTraversal(root.left)
            self.preorderTraversal(root.right)
        test(root)
        return self.arr
```

160. Intersection of Two Linked Lists ↗



Given the heads of two singly linked-lists `headA` and `headB`, return *the node at which the two lists intersect*. If the two linked lists have no intersection at all, return `null`.

For example, the following two linked lists begin to intersect at node `c1`:



The test cases are generated such that there are no cycles anywhere in the entire linked structure.

Note that the linked lists must **retain their original structure** after the function returns.

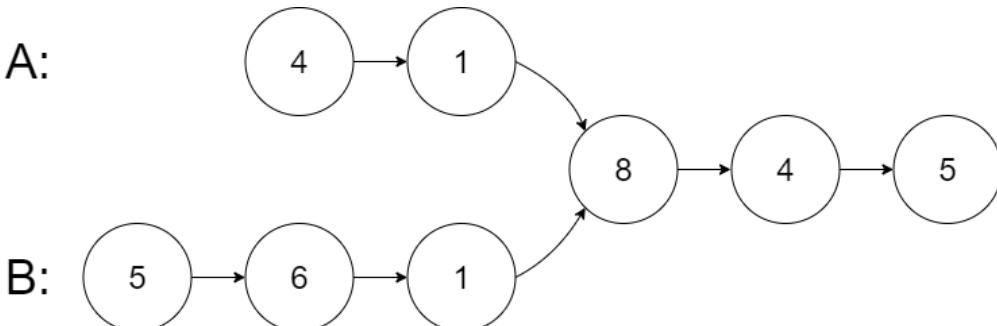
Custom Judge:

The inputs to the **judge** are given as follows (your program is **not** given these inputs):

- `intersectVal` - The value of the node where the intersection occurs. This is `0` if there is no intersected node.
- `listA` - The first linked list.
- `listB` - The second linked list.
- `skipA` - The number of nodes to skip ahead in `listA` (starting from the head) to get to the intersected node.
- `skipB` - The number of nodes to skip ahead in `listB` (starting from the head) to get to the intersected node.

The judge will then create the linked structure based on these inputs and pass the two heads, `headA` and `headB` to your program. If you correctly return the intersected node, then your solution will be **accepted**.

Example 1:

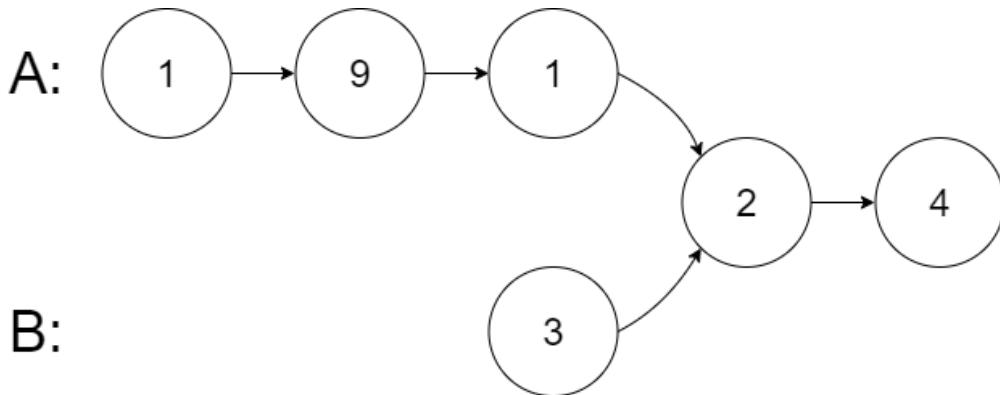


Input: `intersectVal = 8, listA = [4,1,8,4,5], listB = [5,6,1,8,4,5], skipA = 2, skipB = 1`

Output: Intersected at '8'

Explanation: The intersected node's value is 8 (note that this must not be 0 if the lists intersect). From the head of A, it reads as [4,1,8,4,5]. From the head of B, it reads as [5,6,1,8,4,5].

Example 2:

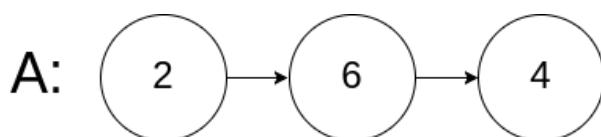


Input: intersectVal = 2, listA = [1,9,1,2,4], listB = [3,2,4], skipA = 3, skipB = 1
Output: Intersected at '2'

Explanation: The intersected node's value is 2 (note that this must not be 0 if the From the head of A, it reads as [1,9,1,2,4]. From the head of B, it reads as [3,2,4]



Example 3:



Input: intersectVal = 0, listA = [2,6,4], listB = [1,5], skipA = 3, skipB = 2

Output: No intersection

Explanation: From the head of A, it reads as [2,6,4]. From the head of B, it reads a

Explanation: The two lists do not intersect, so return null.



Constraints:

- The number of nodes of listA is in the m .
- The number of nodes of listB is in the n .
- $1 \leq m, n \leq 3 * 10^4$
- $1 \leq \text{Node.val} \leq 10^5$
- $0 \leq \text{skipA} < m$
- $0 \leq \text{skipB} < n$
- intersectVal is 0 if listA and listB do not intersect.
- $\text{intersectVal} == \text{listA}[\text{skipA}] == \text{listB}[\text{skipB}]$ if listA and listB intersect.

Follow up: Could you write a solution that runs in $O(m + n)$ time and use only $O(1)$ memory?

```
class Solution:
    def getIntersectionNode(self, headA: ListNode, headB: ListNode) -> Optional[ListNode]:
        l1 = set()
        while headA:
            l1.add(headA)
            headA = headA.next
        while headB:
            if headB in l1:
                return headB
            headB = headB.next
```

169. Majority Element ↗

Given an array `nums` of size `n`, return *the majority element*.

The majority element is the element that appears more than $\lfloor n / 2 \rfloor$ times. You may assume that the majority element always exists in the array.

Example 1:

```
Input: nums = [3,2,3]
Output: 3
```

Example 2:

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

Constraints:

- $n == \text{nums.length}$
- $1 \leq n \leq 5 * 10^4$
- $-10^9 \leq \text{nums}[i] \leq 10^9$

Follow-up: Could you solve the problem in linear time and in $O(1)$ space?

``` class Solution: def majorityElement(self, nums: List[int]) -> int: val = {} # O(N) Tc to Build the Hash Table  
for i in nums: if i not in val: val[i] = 1 else: val[i] += 1 n = len(nums) # once built it takes WC O(N) tc to find the Majority Element. # if we can sort the data based on the Values, then it is O(NlogN) for Tc

```
for k,v in val.items():
 if v > n // 2:
 return k

```

```

173. Binary Search Tree Iterator ↗

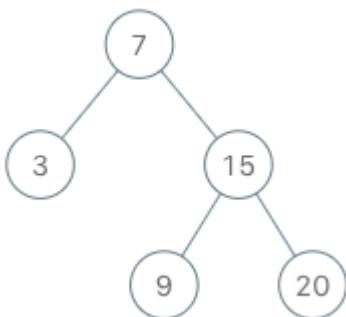
Implement the `BSTIterator` class that represents an iterator over the **in-order traversal** ([https://en.wikipedia.org/wiki/Tree_traversal#In-order_\(LNR\)](https://en.wikipedia.org/wiki/Tree_traversal#In-order_(LNR))) of a binary search tree (BST):

- `BSTIterator(TreeNode root)` Initializes an object of the `BSTIterator` class. The `root` of the BST is given as part of the constructor. The pointer should be initialized to a non-existent number smaller than any element in the BST.
- `boolean hasNext()` Returns `true` if there exists a number in the traversal to the right of the pointer, otherwise returns `false`.
- `int next()` Moves the pointer to the right, then returns the number at the pointer.

Notice that by initializing the pointer to a non-existent smallest number, the first call to `next()` will return the smallest element in the BST.

You may assume that `next()` calls will always be valid. That is, there will be at least a next number in the in-order traversal when `next()` is called.

Example 1:



Input

```
["BSTIterator", "next", "next", "hasNext", "next", "hasNext", "next", "hasNext", "ne  
[[[7, 3, 15, null, null, 9, 20]], [], [], [], [], [], [], []]
```

Output

```
[null, 3, 7, true, 9, true, 15, true, 20, false]
```

Explanation

```
BSTIterator bSTIterator = new BSTIterator([7, 3, 15, null, null, 9, 20]);  
bSTIterator.next(); // return 3  
bSTIterator.next(); // return 7  
bSTIterator.hasNext(); // return True  
bSTIterator.next(); // return 9  
bSTIterator.hasNext(); // return True  
bSTIterator.next(); // return 15  
bSTIterator.hasNext(); // return True  
bSTIterator.next(); // return 20  
bSTIterator.hasNext(); // return False
```

Constraints:

- The number of nodes in the tree is in the range $[1, 10^5]$.
- $0 \leq \text{Node.val} \leq 10^6$
- At most 10^5 calls will be made to `hasNext`, and `next`.

Follow up:

- Could you implement `next()` and `hasNext()` to run in average $O(1)$ time and use $O(h)$ memory, where h is the height of the tree?

```

# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class BSTIterator:

    def __init__(self, root: TreeNode):
        self.temp=[]
        def inorder(self,root):
            if root is None:
                return
            inorder(self,root.left)
            self.temp.append(root.val)
            inorder(self,root.right)
        inorder(self,root)
    def next(self) -> int:
        return self.temp.pop(0)

    def hasNext(self) -> bool:
        if len(self.temp)!=0:
            return True
        return False

# Your BSTIterator object will be instantiated and called as such:
# obj = BSTIterator(root)
# param_1 = obj.next()
# param_2 = obj.hasNext()

```

175. Combine Two Tables ↗

Table: Person

Column Name	Type
personId	int
lastName	varchar
firstName	varchar

personId is the primary key column for this table.

This table contains information about the ID of some persons and their first and las

Table: Address

Column Name	Type
addressId	int
personId	int
city	varchar
state	varchar

addressId is the primary key column for this table.

Each row of this table contains information about the city and state of one person w

Write an SQL query to report the first name, last name, city, and state of each person in the Person table. If the address of a personId is not present in the Address table, report null instead.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Person table:

personId	lastName	firstName
1	Wang	Allen
2	Alice	Bob

Address table:

addressId	personId	city	state
1	2	New York City	New York
2	3	Leetcode	California

Output:

firstName	lastName	city	state
Allen	Wang	Null	Null
Bob	Alice	New York City	New York

Explanation:

There is no address in the address table for the personId = 1 so we return null in the row where addressId = 1 contains information about the address of personId = 2.



*Mssql Solution to the above problem. *By using Left Join the Problem is Solved.

```

SELECT
P.FIRSTNAME,
P.LASTNAME,
A.CITY,
A.STATE
FROM PERSON P LEFT JOIN ADDRESS A
ON P.PERSONID = A.PERSONID
    
```

176. Second Highest Salary



Table: Employee

Column Name	Type
<code>id</code>	<code>int</code>
<code>salary</code>	<code>int</code>

`id` is the primary key column for this table.

Each row of this table contains information about the salary of an employee.

Write an SQL query to report the second highest salary from the `Employee` table. If there is no second highest salary, the query should report `null`.

The query result format is in the following example.

Example 1:

Input:

`Employee` table:

<code>id</code>	<code>salary</code>
1	100
2	200
3	300

Output:

<code>SecondHighestSalary</code>
200

Example 2:

Input:

`Employee` table:

<code>id</code>	<code>salary</code>
1	100

Output:

<code>SecondHighestSalary</code>
<code>null</code>

*Method-1 Using Max 2 times

```
SELECT MAX(SALARY) AS SecondHighestSalary
FROM EMPLOYEE
WHERE SALARY < (SELECT MAX(SALARY) FROM EMPLOYEE)
```

*Method-2,3 Using Case statements and Limits

```
select
  case
    when salary is null then null
    else salary
  end as SecondHighestSalary
from (select distinct salary from employee order by salary desc limit 2 ) as t1
order by salary limit 1

select
(select distinct Salary
from Employee
order by Salary desc limit 1,1) as SecondHighestSalary
from dual
```

177. Nth Highest Salary ↗

Table: Employee

Column Name	Type
<code>id</code>	<code>int</code>
<code>salary</code>	<code>int</code>

`id` is the primary key column for this table.

Each row of this table contains information about the salary of an employee.

Write an SQL query to report the n^{th} highest salary from the `Employee` table. If there is no n^{th} highest salary, the query should report `null`.

The query result format is in the following example.

Example 1:

Input:

Employee table:

id	salary
1	100
2	200
3	300

n = 2

Output:

getNthHighestSalary(2)
200

Example 2:**Input:**

Employee table:

id	salary
1	100

n = 2

Output:

getNthHighestSalary(2)
null

```

SELECT
CASE
    WHEN A.SALARY IS NULL THEN NULL
    ELSE A.SALARY
END AS SAL
FROM
(
    SELECT
        SALARY,
        RANK() OVER(PARTITION BY SALARY) AS RANK
        FROM EMPLOYEE
    ) AS A
WHERE A.RANK = N;

```

- Finalized Code

```

CREATE FUNCTION getNthHighestSalary(N INT) RETURNS INT
BEGIN
  RETURN (
    SELECT
      DISTINCT
      CASE
        WHEN A.SALARY IS NOT NULL THEN A.SALARY
        ELSE NULL
      END AS SAL
    FROM
      (
        SELECT
          SALARY,
          DENSE_RANK() OVER(ORDER BY SALARY DESC) AS RANKING
        FROM EMPLOYEE
      ) AS A
    WHERE A.RANKING = N
  );
END

```

182. Duplicate Emails ↗

Table: Person

Column Name	Type
id	int
email	varchar

id is the primary key column for this table.

Each row of this table contains an email. The emails will not contain uppercase lett

Write an SQL query to report all the duplicate emails.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Person table:

id	email
1	a@b.com
2	c@d.com
3	a@b.com

Output:

Email
a@b.com

Explanation: a@b.com is repeated two times.

- Mysql Solution Using Group BY

```
SELECT EMAIL
FROM PERSON
GROUP BY EMAIL
HAVING COUNT(*) > 1
```

183. Customers Who Never Order ↗



Table: Customers

Column Name	Type
id	int
name	varchar

id is the primary key column for this table.

Each row of this table indicates the ID and name of a customer.

Table: Orders

Column Name	Type
id	int
customerId	int

id is the primary key column for this table.

customerId is a foreign key of the ID from the Customers table.

Each row of this table indicates the ID of an order and the ID of the customer who o

Write an SQL query to report all customers who never order anything.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Customers table:

id	name
1	Joe
2	Henry
3	Sam
4	Max

Orders table:

id	customerId
1	3
2	1

Output:

Customers
Henry
Max

Solution-1:

- T-SQL query statement below

- I have used Left Join to solve this Problem, Where By Left Joining the Table, the result set is obtained.

```
select c.name as Customers
from customers as c left join orders as o
on c.id = o.customerId
where o.customerId is null
```

Solution-2:

- We can use the not in operator to solve this problem

```
SELECT name AS Customers FROM CUSTOMERS WHERE ID NOT IN (SELECT customerId FROM ORDERS);
```

Solution-3:

- We can also use the, Minus Operator to solve this problem.

```
select name as Customers
from CUSTOMERS
Where id in (SELECT id FROM CUSTOMERS minus SELECT customerId FROM ORDERS);
```

191. Number of 1 Bits ↗

Write a function that takes an unsigned integer and returns the number of '1' bits it has (also known as the Hamming weight (http://en.wikipedia.org/wiki/Hamming_weight)).

Note:

- Note that in some languages, such as Java, there is no unsigned integer type. In this case, the input will be given as a signed integer type. It should not affect your implementation, as the integer's internal binary representation is the same, whether it is signed or unsigned.
- In Java, the compiler represents the signed integers using 2's complement notation (https://en.wikipedia.org/wiki/Two%27s_complement). Therefore, in **Example 3**, the input represents the signed integer. -3 .

Example 1:

Input: n = 00000000000000000000000000001011

Output: 3

Explanation: The input binary string 00000000000000000000000000001011 has a total of

Example 2:

Input: n = 0000000000000000000000000010000000

Output: 1

Explanation: The input binary string 000000000000000000000000001000000 has a total of

Example 3:

Input: n = 11111111111111111111111111111101

Output: 31

Explanation: The input binary string 111111111111111111111111111101 has a total of

Constraints:

- The input must be a **binary string** of length 32 .

Follow up: If this function is called many times, how would you optimize it?

```
class Solution:
    def hammingWeight(self, n: int) -> int:
        return bin(n)[2:].count('1')
```

196. Delete Duplicate Emails ↗

Table: Person

Column Name	Type
id	int
email	varchar

id is the primary key column for this table.

Each row of this table contains an email. The emails will not contain uppercase lett

Write an SQL query to **delete** all the duplicate emails, keeping only one unique email with the smallest id . Note that you are supposed to write a `DELETE` statement and not a `SELECT` one.

After running your script, the answer shown is the `Person` table. The driver will first compile and run your piece of code and then show the `Person` table. The final order of the `Person` table **does not matter**.

The query result format is in the following example.

Example 1:

Input:

Person table:

id	email
1	john@example.com
2	bob@example.com
3	john@example.com

Output:

id	email
1	john@example.com
2	bob@example.com

Explanation: `john@example.com` is repeated two times. We keep the row with the smaller id.

```
-- what we need to do is group by
-- all the emails and must delete
-- all the mails other than min
-- val one.
```

```
DELETE FROM PERSON
WHERE ID NOT IN
(SELECT MIN(ID) FROM PERSON GROUP BY EMAIL)
```

197. Rising Temperature

Table: Weather

Column Name	Type
id	int
recordDate	date
temperature	int

id is the primary key for this table.

This table contains information about the temperature on a certain day.

Write an SQL query to find all dates' Id with higher temperatures compared to its previous dates (yesterday).

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Weather table:

id	recordDate	temperature
1	2015-01-01	10
2	2015-01-02	25
3	2015-01-03	20
4	2015-01-04	30

Output:

id
2
4

Explanation:

In 2015-01-02, the temperature was higher than the previous day (10 → 25).

In 2015-01-04, the temperature was higher than the previous day (20 → 30).

```
# Write your MySQL query statement below

select w1.id
from
weather w1,
weather w2
where
to_days(w1.recordDate) = to_days(w2.recordDate) + 1
and w1.temperature > w2.temperature
```

202. Happy Number ↗



Write an algorithm to determine if a number n is happy.

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

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

Return `true` if n is a happy number, and `false` if not.

Example 1:

Input: $n = 19$
Output: true
Explanation:
 $1^2 + 9^2 = 82$
 $8^2 + 2^2 = 68$
 $6^2 + 8^2 = 100$
 $1^2 + 0^2 + 0^2 = 1$

Example 2:

Input: $n = 2$
Output: false

Constraints:

- $1 \leq n \leq 2^{31} - 1$

Method - 1

1. Mostly Everyone, will get to the point where, it finds the square of its individual digits and then, will also implement the $\text{val} = 1$ logic, but we fail to crack the infinite loop issue.
2. The Trick here is, every number which will never arrive a sum = 1 will repeat its value again and again leading to infinite, iterations.
3. So, to break the loop, we can track the values, by maintaining a Cache. Which when a value is encountered, halts the program.

```
class Solution:
    def calc_square(self, val):
        cache = []
        while True:
            res = 0
            while val > 0:
                res = res + (val % 10) ** 2
                val = val // 10

            if res == 1:
                break
            if res in cache:
                break
            else:
                cache.append(res)

            val = res

        return True if res == 1 else False

    def isHappy(self, n: int) -> bool:
        return self.calc_square(n)
```

Method -2

- Optimizing the solution.
- Using, Cache as Hash Map reduces the Search time to O(1)
- TC O(N*K)

```

class Solution:
    def calc_square(self, val):
        cache = {}
        while True:
            res = 0
            while val > 0:
                res = res + (val % 10) ** 2
                val = val // 10

            if res == 1:
                break
            if res in cache:
                break
            else:
                cache[res] = 1

        val = res

        return True if res == 1 else False

    def isHappy(self, n: int) -> bool:
        return self.calc_square(n)

```

217. Contains Duplicate ↗

Given an integer array `nums`, return `true` if any value appears **at least twice** in the array, and return `false` if every element is distinct.

Example 1:

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

Example 2:

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

Example 3:

Input: `nums = [1,1,1,3,3,4,3,2,4,2]`
Output: `true`

Constraints:

- $1 \leq \text{nums.length} \leq 10^5$
- $-10^9 \leq \text{nums}[i] \leq 10^9$

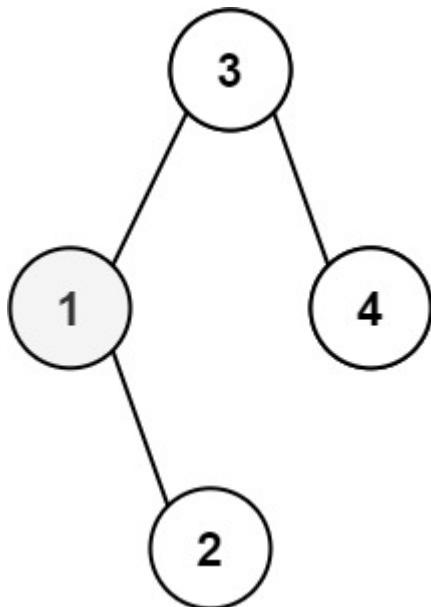
1. Here the Key idea, is to check, whether the arr is having duplicates or not.
2. So i have used, set data structure to validate it.
3. We can also use hashmap to check if it has duplicates or not.

```
# O(N) Space
# O(N) Time

class Solution:
    def containsDuplicate(self, nums: List[int]) -> bool:
        # This Approach Uses O(N) Extra Space
        return True if len(set(nums)) != len(nums) else False
```

230. Kth Smallest Element in a BST ↗

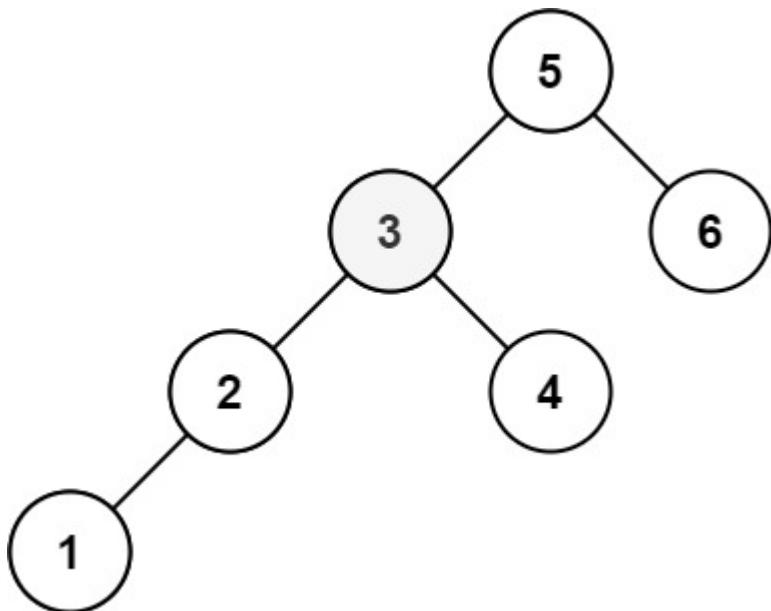
Given the `root` of a binary search tree, and an integer `k`, return *the kth smallest value (**1-indexed**) of all the values of the nodes in the tree*.

Example 1:

Input: root = [3,1,4,null,2], k = 1

Output: 1

Example 2:



Input: root = [5,3,6,2,4,null,null,1], k = 3

Output: 3

Constraints:

- The number of nodes in the tree is n .
- $1 \leq k \leq n \leq 10^4$
- $0 \leq \text{Node.val} \leq 10^4$

Follow up: If the BST is modified often (i.e., we can do insert and delete operations) and you need to find the kth smallest frequently, how would you optimize?

```

# 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 kthSmallest(self, root: Optional[TreeNode], k: int) -> int:
        m=[]
        self.temp(root,m)
        return m[k-1]
    def temp(self,root,m):
        if root is None:
            return
        self.temp(root.left,m)
        m.append(root.val)
        self.temp(root.right,m)
  
```

232. Implement Queue using Stacks ↗

Implement a first in first out (FIFO) queue using only two stacks. The implemented queue should support all the functions of a normal queue (`push` , `peek` , `pop` , and `empty`).

Implement the `MyQueue` class:

- `void push(int x)` Pushes element `x` to the back of the queue.
- `int pop()` Removes the element from the front of the queue and returns it.
- `int peek()` Returns the element at the front of the queue.
- `boolean empty()` Returns `true` if the queue is empty, `false` otherwise.

Notes:

- You must use **only** standard operations of a stack, which means only `push` to `top` , `peek/pop from top` , `size` , and `is empty` operations are valid.
- Depending on your language, the stack may not be supported natively. You may simulate a stack using a list or deque (double-ended queue) as long as you use only a stack's standard operations.

Example 1:

Input

```
["MyQueue", "push", "push", "peek", "pop", "empty"]
[], [1], [2], [], [], []]
```

Output

```
[null, null, null, 1, 1, false]
```

Explanation

```
MyQueue myQueue = new MyQueue();
myQueue.push(1); // queue is: [1]
myQueue.push(2); // queue is: [1, 2] (leftmost is front of the queue)
myQueue.peek(); // return 1
myQueue.pop(); // return 1, queue is [2]
myQueue.empty(); // return false
```

Constraints:

- $1 \leq x \leq 9$
- At most 100 calls will be made to `push` , `pop` , `peek` , and `empty` .
- All the calls to `pop` and `peek` are valid.

Follow-up: Can you implement the queue such that each operation is **amortized**

(https://en.wikipedia.org/wiki/Amortized_analysis) $O(1)$ time complexity? In other words, performing n operations will take overall $O(n)$ time even if one of those operations may take longer.

- As per the definition, implement the algorithm.

```
class MyQueue:

    def __init__(self):
        self.arr = []

    def push(self, x: int) -> None:
        self.arr.append(x)

    def pop(self) -> int:
        val = self.arr[0]
        del self.arr[0]
        return val

    def peek(self) -> int:
        return self.arr[0]
```

```
def empty(self) -> bool:
    if not len(self.arr):
        return True
    else:
        return False
```

Your MyQueue object will be instantiated and called as such:

```
obj = MyQueue()
obj.push(x)
param_2 = obj.pop()
param_3 = obj.peek()
param_4 = obj.empty()
...

```

242. Valid Anagram ↗

Given two strings `s` and `t`, return `true` if `t` is an anagram of `s`, and `false` otherwise.

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

Example 1:

Input: s = "anagram", t = "nagaram"
Output: true

Example 2:

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

Constraints:

- $1 \leq s.length, t.length \leq 5 * 10^4$
- s and t consist of lowercase English letters.

Follow up: What if the inputs contain Unicode characters? How would you adapt your solution to such a case?

Method 1

```
# O(N Log N) TC
# Space O(1)

class Solution:
    def isAnagram(self, s: str, t: str) -> bool:
        return True if sorted(s) == sorted(t) else False
```

Method 2

```
class Solution:
    def isAnagram(self, s: str, t: str) -> bool:
        r1 = {}
        for val in s:
            if val not in r1:
                r1[val] = 1
            else:
                r1[val] += 1
        r2 = {}
        for val in t:
            if val not in r2:
                r2[val] = 1
            else:
                r2[val] += 1
        if r1 == r2:
            return True
        return False
```

268. Missing Number ↗

Given an array `nums` containing n distinct numbers in the range $[0, n]$, return *the only number in the range that is missing from the array*.

Example 1:

Input: `nums = [3,0,1]`

Output: 2

Explanation: $n = 3$ since there are 3 numbers, so all numbers are in the range $[0,3]$.



Example 2:

Input: `nums = [0,1]`

Output: 2

Explanation: $n = 2$ since there are 2 numbers, so all numbers are in the range $[0,2]$.



Example 3:

Input: `nums = [9,6,4,2,3,5,7,0,1]`

Output: 8

Explanation: $n = 9$ since there are 9 numbers, so all numbers are in the range $[0,9]$.



Constraints:

- $n == \text{nums.length}$
- $1 \leq n \leq 10^4$
- $0 \leq \text{nums}[i] \leq n$
- All the numbers of `nums` are **unique**.

Follow up: Could you implement a solution using only $O(1)$ extra space complexity and $O(n)$ runtime complexity?

- Python O(1) Solution Without any extra space

```
class Solution:  
    def missingNumber(self, nums: List[int]) -> int:  
        length = len(nums)  
        sum_of_val = int(length * (length+1) / 2)  
        return sum_of_val - sum(nums)
```

283. Move Zeroes ↗

Given an integer array `nums`, move all `0`'s to the end of it while maintaining the relative order of the non-zero elements.

Note that you must do this in-place without making a copy of the array.

Example 1:

```
Input: nums = [0,1,0,3,12]  
Output: [1,3,12,0,0]
```

Example 2:

```
Input: nums = [0]  
Output: [0]
```

Constraints:

- $1 \leq \text{nums.length} \leq 10^4$
- $-2^{31} \leq \text{nums}[i] \leq 2^{31} - 1$

Follow up: Could you minimize the total number of operations done?

```

class Solution:
    def moveZeroes(self, nums: List[int]) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """
        length = len(nums)
        for i in range(length):
            if nums[i] == 0:
                nums[i] = inf
                nums.append(0)

        while True:
            if inf in nums:
                nums.remove(inf)
            else:
                break

        return nums

```

303. Range Sum Query - Immutable

Given an integer array `nums`, handle multiple queries of the following type:

1. Calculate the **sum** of the elements of `nums` between indices `left` and `right inclusive` where `left <= right`.

Implement the `NumArray` class:

- `NumArray(int[] nums)` Initializes the object with the integer array `nums`.
- `int sumRange(int left, int right)` Returns the **sum** of the elements of `nums` between indices `left` and `right inclusive` (i.e. `nums[left] + nums[left + 1] + ... + nums[right]`).

Example 1:

Input

```
[ "NumArray", "sumRange", "sumRange", "sumRange" ]
[[[-2, 0, 3, -5, 2, -1]], [0, 2], [2, 5], [0, 5]]
```

Output

```
[null, 1, -1, -3]
```

Explanation

```
NumArray numArray = new NumArray([-2, 0, 3, -5, 2, -1]);
numArray.sumRange(0, 2); // return (-2) + 0 + 3 = 1
numArray.sumRange(2, 5); // return 3 + (-5) + 2 + (-1) = -1
numArray.sumRange(0, 5); // return (-2) + 0 + 3 + (-5) + 2 + (-1) = -3
```

Constraints:

- $1 \leq \text{nums.length} \leq 10^4$
- $-10^5 \leq \text{nums}[i] \leq 10^5$
- $0 \leq \text{left} \leq \text{right} < \text{nums.length}$
- At most 10^4 calls will be made to `sumRange`.

```
class NumArray:

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

    def sumRange(self, left: int, right: int) -> int:
        if len(self.nums)>0:
            return sum(self.nums[left: right+1])

# Your NumArray object will be instantiated and called as such:
# obj = NumArray(nums)
# param_1 = obj.sumRange(left,right)
```

304. Range Sum Query 2D - Immutable

Given a 2D matrix `matrix`, handle multiple queries of the following type:

- Calculate the **sum** of the elements of `matrix` inside the rectangle defined by its **upper left corner** (`row1, col1`) and **lower right corner** (`row2, col2`).

Implement the `NumMatrix` class:

- `NumMatrix(int[][] matrix)` Initializes the object with the integer matrix `matrix`.
- `int sumRegion(int row1, int col1, int row2, int col2)` Returns the **sum** of the elements of `matrix` inside the rectangle defined by its **upper left corner** (`row1, col1`) and **lower right corner** (`row2, col2`).

Example 1:

3	0	1	4	2
5	6	3	2	1
1	2	0	1	5
4	1	0	1	7
1	0	3	0	5

Input

```
["NumMatrix", "sumRegion", "sumRegion", "sumRegion"]
[[[[3, 0, 1, 4, 2], [5, 6, 3, 2, 1], [1, 2, 0, 1, 5], [4, 1, 0, 1, 7], [1, 0, 3, 0,
Output
[null, 8, 11, 12]
```

Explanation

```
NumMatrix numMatrix = new NumMatrix([[3, 0, 1, 4, 2], [5, 6, 3, 2, 1], [1, 2, 0, 1, 5],
numMatrix.sumRegion(2, 1, 4, 3); // return 8 (i.e sum of the red rectangle)
numMatrix.sumRegion(1, 1, 2, 2); // return 11 (i.e sum of the green rectangle)
numMatrix.sumRegion(1, 2, 2, 4); // return 12 (i.e sum of the blue rectangle)
```

**Constraints:**

- $m == \text{matrix.length}$
- $n == \text{matrix}[i].length$
- $1 \leq m, n \leq 200$
- $-10^4 \leq \text{matrix}[i][j] \leq 10^4$
- $0 \leq \text{row1} \leq \text{row2} < m$
- $0 \leq \text{col1} \leq \text{col2} < n$
- At most 10^4 calls will be made to `sumRegion`.

- TLE Solution where there is no Optimization, as we compute the matrix every time we query it.

```

class NumMatrix:

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

    def sumRegion(self, row1: int, col1: int, row2: int, col2: int) -> int:
        self.block_sum = 0
        for i in range(row1, row2+1):
            for j in range(col1, col2+1):
                self.block_sum += self.matrix[i][j]
        return self.block_sum

# Your NumMatrix object will be instantiated and called as such:
# obj = NumMatrix(matrix)
# param_1 = obj.sumRegion(row1,col1,row2,col2)

```

344. Reverse String ↗

Write a function that reverses a string. The input string is given as an array of characters `s`.

You must do this by modifying the input array in-place (https://en.wikipedia.org/wiki/In-place_algorithm) with $O(1)$ extra memory.

Example 1:

```

Input: s = ["h","e","l","l","o"]
Output: ["o","l","l","e","h"]

```

Example 2:

```

Input: s = ["H","a","n","n","a","h"]
Output: ["h","a","n","n","a","H"]

```

Constraints:

- $1 \leq s.length \leq 10^5$
- $s[i]$ is a printable ascii character (https://en.wikipedia.org/wiki/ASCII#Printable_characters).

```
# O(1) Inplace Solution.
class Solution:
    def reverseString(self, s: List[str]) -> None:
        """
        Do not return anything, modify s in-place instead.
        """
        length = len(s)
        if length % 2 != 0:
            new_length = (length // 2) + 1
            last = length
            for i in range(new_length):
                temp = s[i]
                last = last - 1
                s[i] = s[last]
                s[last] = temp

        else:
            new_length = (length // 2)
            last = length
            for i in range(new_length):
                temp = s[i]
                last = last - 1
                s[i] = s[last]
                s[last] = temp
```

347. Top K Frequent Elements ↗

Given an integer array `nums` and an integer `k`, return *the k most frequent elements*. You may return the answer in **any order**.

Example 1:

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

Example 2:

Input: `nums = [1]`, `k = 1`
Output: `[1]`

Constraints:

- $1 \leq \text{nums.length} \leq 10^5$
- k is in the range $[1, \text{the number of unique elements in the array}]$.

- It is **guaranteed** that the answer is **unique**.

Follow up: Your algorithm's time complexity must be better than $O(n \log n)$, where n is the array's size.

```
class Solution:
    def topKFrequent(self, nums: List[int], k: int) -> List[int]:
        res = {}
        for val in nums:
            if val not in res:
                res[val] = 1
            else:
                res[val] += 1
        return [val[0] for val in sorted(res.items(), key= lambda x:x[1], reverse = True)[0:k]]
```

389. Find the Difference ↗



You are given two strings s and t .

String t is generated by random shuffling string s and then add one more letter at a random position.

Return the letter that was added to t .

Example 1:

```
Input: s = "abcd", t = "abcde"
Output: "e"
Explanation: 'e' is the letter that was added.
```

Example 2:

```
Input: s = "", t = "y"
Output: "y"
```

Constraints:

- $0 \leq s.length \leq 1000$
- $t.length == s.length + 1$
- s and t consist of lowercase English letters.

```
class Solution:  
    def findTheDifference(self, s: str, t: str) -> str:  
        t = list(t)  
        for char in list(s):  
            t.remove(char)  
        return ''.join(t)
```

392. Is Subsequence ↗

Given two strings s and t , return true if s is a **subsequence** of t , or false otherwise.

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

Example 1:

Input: $s = \text{"abc"}$, $t = \text{"ahbgdc"}$
Output: true

Example 2:

Input: $s = \text{"axc"}$, $t = \text{"ahbgdc"}$
Output: false

Constraints:

- $0 \leq s.\text{length} \leq 100$
- $0 \leq t.\text{length} \leq 10^4$
- s and t consist only of lowercase English letters.

Follow up: Suppose there are lots of incoming s , say s_1, s_2, \dots, s_k where $k \geq 10^9$, and you want to check one by one to see if t has its subsequence. In this scenario, how would you change your code?

```
# the most challenging question ever solved.

class Solution:
    def isSubsequence(self, s: str, t: str) -> bool:
        i, j = 0, 0
        while i < len(s) and j < len(t):
            if s[i] == t[j]:
                i+=1
                j+=1
            else:
                j+=1

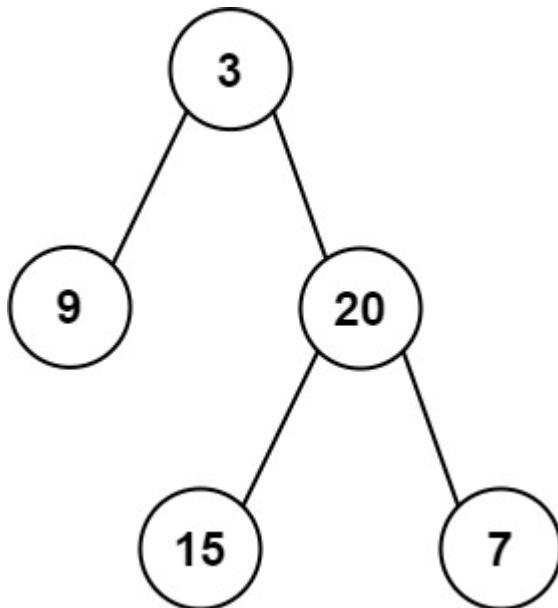
        return True if len(s) == i else False
```

404. Sum of Left Leaves ↗

Given the `root` of a binary tree, return *the sum of all left leaves*.

A **leaf** is a node with no children. A **left leaf** is a leaf that is the left child of another node.

Example 1:



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

Output: 24

Explanation: There are two left leaves in the binary tree, with values 9 and 15 respectively.



Example 2:

Input: root = [1]

Output: 0

Constraints:

- The number of nodes in the tree is in the range [1, 1000].
- $-1000 \leq \text{Node.val} \leq 1000$

```
# 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 sumOfLeftLeaves(self, root: Optional[TreeNode]) -> int:
        temp=[]
        temp.append(root)
        sum=0
        if not root:
            return 0
        left=[]
        while temp:
            temp1=temp.pop(0)
            if temp1.left:
                temp.append(temp1.left)
                left.append(temp1.left)
            y=left.pop(0)
            if y.left is None and y.right is None:
                sum+=y.val
            if temp1.right:
                temp.append(temp1.right)

        return sum
```

413. Arithmetic Slices ↗

An integer array is called arithmetic if it consists of **at least three elements** and if the difference between any two consecutive elements is the same.

- For example, [1,3,5,7,9], [7,7,7,7], and [3,-1,-5,-9] are arithmetic sequences.

Given an integer array `nums`, return *the number of arithmetic **subarrays** of* `nums`.

A **subarray** is a contiguous subsequence of the array.

Example 1:

Input: nums = [1,2,3,4]

Output: 3

Explanation: We have 3 arithmetic slices in nums: [1, 2, 3], [2, 3, 4] and [1,2,3,4]

**Example 2:**

Input: nums = [1]

Output: 0

Constraints:

- $1 \leq \text{nums.length} \leq 5000$
- $-1000 \leq \text{nums}[i] \leq 1000$

```
class Solution:
    def numberOfArithmeticSlices(self, nums: List[int]) -> int:
        length = len(nums)
        dp = [0]*length
        for i in range(2,length):
            if nums[i] - nums[i-1] == nums[i-1] - nums[i-2]:
                dp[i] = 1 + dp[i-1]

        return sum(dp)
```

496. Next Greater Element I ↗

The **next greater element** of some element x in an array is the **first greater** element that is **to the right** of x in the same array.

You are given two **distinct 0-indexed** integer arrays nums1 and nums2 , where nums1 is a subset of nums2 .

For each $0 \leq i < \text{nums1.length}$, find the index j such that $\text{nums1}[i] == \text{nums2}[j]$ and determine the **next greater element** of $\text{nums2}[j]$ in nums2 . If there is no next greater element, then the answer for this query is -1 .

Return an array ans of length nums1.length such that $\text{ans}[i]$ is the **next greater element** as described above.

Example 1:

Input: nums1 = [4,1,2], nums2 = [1,3,4,2]

Output: [-1,3,-1]

Explanation: The next greater element for each value of nums1 is as follows:

- 4 is underlined in nums2 = [1,3,4,2]. There is no next greater element, so the ans
- 1 is underlined in nums2 = [1,3,4,2]. The next greater element is 3.
- 2 is underlined in nums2 = [1,3,4,2]. There is no next greater element, so the ans

Example 2:

Input: nums1 = [2,4], nums2 = [1,2,3,4]

Output: [3,-1]

Explanation: The next greater element for each value of nums1 is as follows:

- 2 is underlined in nums2 = [1,2,3,4]. The next greater element is 3.
- 4 is underlined in nums2 = [1,2,3,4]. There is no next greater element, so the ans

Constraints:

- $1 \leq \text{nums1.length} \leq \text{nums2.length} \leq 1000$
- $0 \leq \text{nums1}[i], \text{nums2}[i] \leq 10^4$
- All integers in nums1 and nums2 are **unique**.
- All the integers of nums1 also appear in nums2 .

Follow up: Could you find an $O(\text{nums1.length} + \text{nums2.length})$ solution?

```

class Solution:
    def nextGreaterElement(self, nums1: List[int], nums2: List[int]) -> List[int]:
        ans = []
        # iterate the values O(N) time.
        for val in nums1:
            ind = nums2.index(val)
            if len(nums2[ind+1:]) >= 1 :
                res = 0
                # fetch the first maximum value from the right
                for new_val in nums2[ind+1:]:
                    if new_val > val:
                        res = new_val
                        break
                # if the value is zero then assign -1 to the list
                if not res:
                    ans.append(-1)
                # else append the value to the list
                else:
                    ans.append(res)
            else:
                # edge case scenario for the end value in the list.
                ans.append(-1)

        return ans

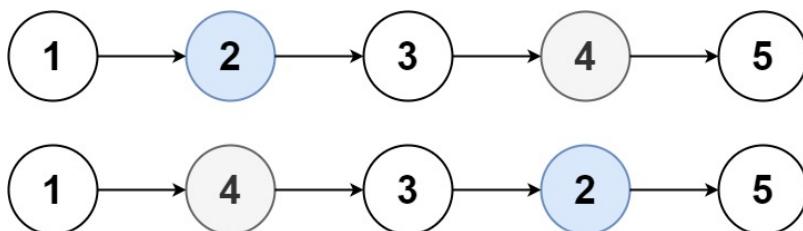
```

1721. Swapping Nodes in a Linked List ↗

You are given the head of a linked list, and an integer k .

Return the head of the linked list after **swapping** the values of the kth node from the beginning and the kth node from the end (the list is 1-indexed).

Example 1:



Input: head = [1,2,3,4,5], k = 2

Output: [1,4,3,2,5]

Example 2:

Input: head = [7,9,6,6,7,8,3,0,9,5], k = 5

Output: [7,9,6,6,8,7,3,0,9,5]

Constraints:

- The number of nodes in the list is n .
- $1 \leq k \leq n \leq 10^5$
- $0 \leq \text{Node.val} \leq 100$

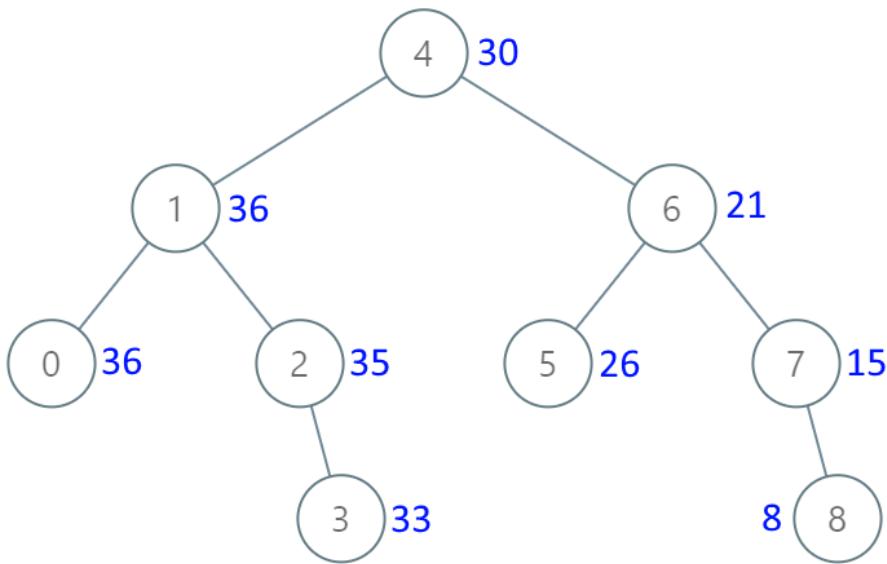
```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def swapNodes(self, head: Optional[ListNode], k: int) -> Optional[ListNode]:
        res = []
        while head:
            res.append(head.val)
            head = head.next
        if len(res) >= k:
            temp = res[k-1]
            res[k-1] = res[-k]
            res[-k] = temp
        node = ListNode(-1)
        root = node
        for val in res:
            if node.val == -1:
                node.val = val
            else:
                node.next = ListNode(val)
                node = node.next
        return root
```

538. Convert BST to Greater Tree ↗

Given the `root` of a Binary Search Tree (BST), convert it to a Greater Tree such that every key of the original BST is changed to the original key plus the sum of all keys greater than the original key in BST.

As a reminder, a *binary search tree* is a tree that satisfies these constraints:

- The left subtree of a node contains only nodes with keys **less than** the node's key.
- The right subtree of a node contains only nodes with keys **greater than** the node's key.
- Both the left and right subtrees must also be binary search trees.

Example 1:

Input: root = [4,1,6,0,2,5,7,null,null,null,3,null,null,null,8]
Output: [30,36,21,36,35,26,15,null,null,null,33,null,null,null,8]

Example 2:

Input: root = [0,null,1]
Output: [1,null,1]

Constraints:

- The number of nodes in the tree is in the range $[0, 10^4]$.
- $-10^4 \leq \text{Node.val} \leq 10^4$
- All the values in the tree are **unique**.
- `root` is guaranteed to be a valid binary search tree.

Note: This question is the same as 1038: <https://leetcode.com/problems/binary-search-tree-to-greater-sum-tree/> (<https://leetcode.com/problems/binary-search-tree-to-greater-sum-tree/>)

```

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

```

561. Array Partition ↗



Given an integer array `nums` of $2n$ integers, group these integers into n pairs $(a_1, b_1), (a_2, b_2), \dots, (a_n, b_n)$ such that the sum of $\min(a_i, b_i)$ for all i is **maximized**. Return *the maximized sum*.

Example 1:

Input: `nums = [1,4,3,2]`

Output: 4

Explanation: All possible pairings (ignoring the ordering of elements) are:

1. $(1, 4), (2, 3) \rightarrow \min(1, 4) + \min(2, 3) = 1 + 2 = 3$
2. $(1, 3), (2, 4) \rightarrow \min(1, 3) + \min(2, 4) = 1 + 2 = 3$
3. $(1, 2), (3, 4) \rightarrow \min(1, 2) + \min(3, 4) = 1 + 3 = 4$

So the maximum possible sum is 4.

Example 2:

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

Output: 9

Explanation: The optimal pairing is $(2, 1), (2, 5), (6, 6)$. $\min(2, 1) + \min(2, 5) +$

Constraints:

- $1 \leq n \leq 10^4$
- $\text{nums.length} == 2 * n$

- $-10^4 \leq \text{nums}[i] \leq 10^4$

- the biggest problem with greedy, is many problems can be sorted, first and then, later we can perform, the operations.

566. Reshape the Matrix ↗

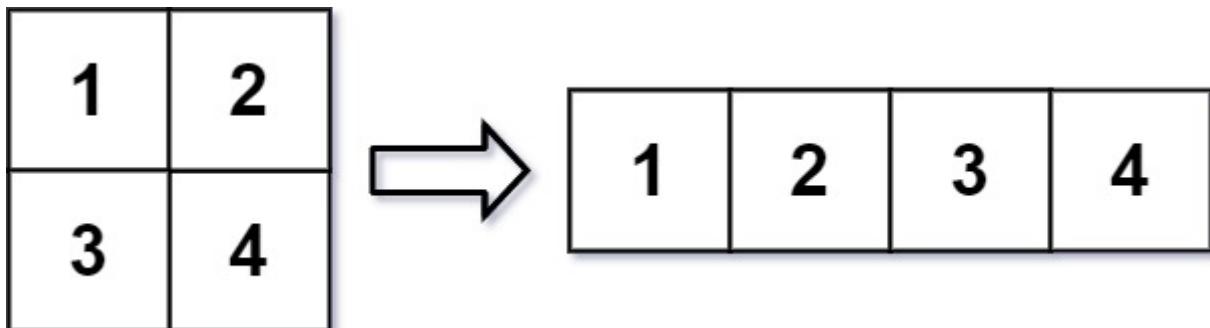
In MATLAB, there is a handy function called `reshape` which can reshape an $m \times n$ matrix into a new one with a different size $r \times c$ keeping its original data.

You are given an $m \times n$ matrix `mat` and two integers `r` and `c` representing the number of rows and the number of columns of the wanted reshaped matrix.

The reshaped matrix should be filled with all the elements of the original matrix in the same row-traversing order as they were.

If the `reshape` operation with given parameters is possible and legal, output the new reshaped matrix; Otherwise, output the original matrix.

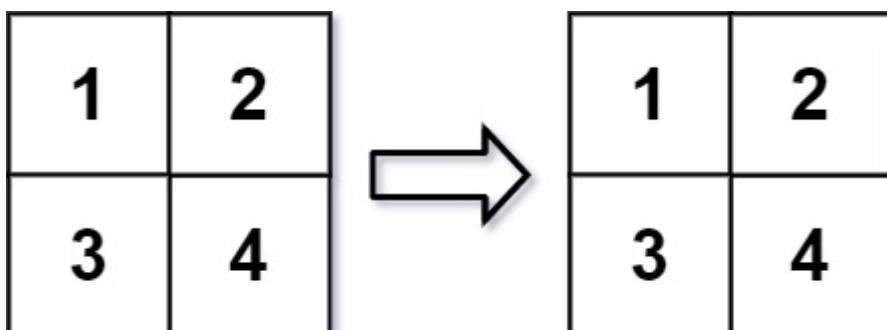
Example 1:



Input: mat = [[1,2],[3,4]], r = 1, c = 4

Output: [[1,2,3,4]]

Example 2:



Input: mat = [[1,2],[3,4]], r = 2, c = 4

Output: [[1,2],[3,4]]

Constraints:

- $m == \text{mat.length}$
- $n == \text{mat[i].length}$
- $1 \leq m, n \leq 100$
- $-1000 \leq \text{mat}[i][j] \leq 1000$
- $1 \leq r, c \leq 300$

```
class Solution:
    def check_if_Valid(self, mat):
        count = 0
        for i in range(len(mat)):
            for j in mat[i]:
                count+=1

        return count

    def matrixReshape(self, mat: List[List[int]], r: int, c: int) -> List[List[int]]:
        if (r*c) == self.check_if_Valid(mat):
            new_mat = []
            max_limit = (r*c)/r
            count = 1
            new_mat.append([])
            level = 0

            for i in range(len(mat)):
                for val in mat[i]:
                    new_mat[level].append(val)
                    if count == max_limit:
                        count = 1
                        level += 1
                        new_mat.append([])
                    else:
                        count +=1

            return new_mat[0:-1]

        else:
            return mat
```

570. Managers with at Least 5 Direct Reports ↗



Table: Employee

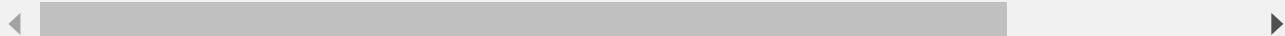
Column Name	Type
id	int
name	varchar
department	varchar
managerId	int

`id` is the primary key column for this table.

Each row of this table indicates the name of an employee, their department, and the

If `managerId` is null, then the employee does not have a manager.

No employee will be the manager of themself.



Write an SQL query to report the managers with at least **five direct reports**.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:**Input:**

Employee table:

id	name	department	managerId
101	John	A	None
102	Dan	A	101
103	James	A	101
104	Amy	A	101
105	Anne	A	101
106	Ron	B	101

Output:

name
John

- LeftJoin + Group BY + Having

```

SELECT E1.NAME
FROM EMPLOYEE AS E1 LEFT JOIN EMPLOYEE AS E2
ON E1.ID = E2.MANAGERID
WHERE E2.MANAGERID IS NOT NULL
GROUP BY E1.ID
HAVING COUNT(E1.ID) >= 5

```

- 2-Solution Using In operator + Join + Group BY

```

SELECT NAME
FROM EMPLOYEE
WHERE ID IN
(
    SELECT
        E1.ID
    FROM EMPLOYEE AS E1 JOIN EMPLOYEE AS E2
    ON E1.ID = E2.MANAGERID
    GROUP BY E1.ID
    HAVING COUNT(DISTINCT E2.ID) >= 5
)

```

- 3-Solution Using the Solution.

```

SELECT
    E1.NAME
FROM EMPLOYEE AS E1 JOIN EMPLOYEE AS E2
ON E1.ID = E2.MANAGERID
GROUP BY E1.ID
HAVING COUNT(DISTINCT E2.ID) >= 5

```

574. Winning Candidate ↗

Table: Candidate

Column Name	Type
id	int
name	varchar

id is the primary key column for this table.

Each row of this table contains information about the id and the name of a candidate

Table: Vote

```
+-----+-----+
| Column Name | Type |
+-----+-----+
| id          | int   |
| candidateId | int   |
+-----+-----+
id is an auto-increment primary key.
candidateId is a foreign key to id from the Candidate table.
Each row of this table determines the candidate who got the ith vote in the elections.
```

Write an SQL query to report the name of the winning candidate (i.e., the candidate who got the largest number of votes).

The test cases are generated so that **exactly one candidate wins** the elections.

The query result format is in the following example.

Example 1:

Input:

Candidate table:

id	name
1	A
2	B
3	C
4	D
5	E

Vote table:

id	candidateId
1	2
2	4
3	3
4	2
5	5

Output:

name
B

Explanation:

Candidate B has 2 votes. Candidates C, D, and E have 1 vote each.

The winner is candidate B.

- Using Left Join + Order By + Limit to Get the Highest Candidate Votes `` SELECT C.NAME FROM CANDIDATE AS C LEFT JOIN VOTE AS V ON C.ID = V.CANDIDATEID GROUP BY C.ID ORDER BY COUNT(*) DESC LIMIT 1 ``

577. Employee Bonus ↗



Table: Employee

Column Name	Type
empId	int
name	varchar
supervisor	int
salary	int

empId is the primary key column for this table.

Each row of this table indicates the name and the ID of an employee in addition to t

Table: Bonus

Column Name	Type
empId	int
bonus	int

empId is the primary key column for this table.

empId is a foreign key to empId from the Employee table.

Each row of this table contains the id of an employee and their respective bonus.

Write an SQL query to report the name and bonus amount of each employee with a bonus **less than 1000**.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Employee table:

empId	name	supervisor	salary
3	Brad	null	4000
1	John	3	1000
2	Dan	3	2000
4	Thomas	3	4000

Bonus table:

empId	bonus
2	500
4	2000

Output:

name	bonus
Brad	null
John	null
Dan	500

- First Time when i was solving this Question i had faced an issue due to union but later i found

```

SELECT
E.NAME,
CASE
    WHEN B.BONUS < 1000 THEN B.BONUS
    ELSE NULL
END AS BONUS
FROM EMPLOYEE AS E LEFT JOIN BONUS AS B
ON E.EMPID = B.EMPID
WHERE B.BONUS < 1000 OR B.BONUS IS NULL

```

578. Get Highest Answer Rate Question ↗



Table: SurveyLog

Column Name	Type
id	int
action	ENUM
question_id	int
answer_id	int
q_num	int
timestamp	int

There is no primary key for this table. It may contain duplicates.

action is an ENUM of the type: "show", "answer", or "skip".

Each row of this table indicates the user with ID = id has taken an action with the

If the action taken by the user is "answer", answer_id will contain the id of that a
q_num is the numeral order of the question in the current session.



The **answer rate** for a question is the number of times a user answered the question by the number of times a user showed the question.

Write an SQL query to report the question that has the highest **answer rate**. If multiple questions have the same maximum **answer rate**, report the question with the smallest `question_id`.

The query result format is in the following example.

Example 1:

Input:

SurveyLog table:

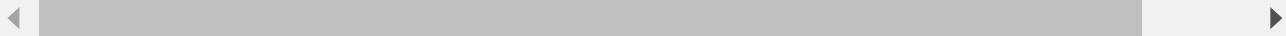
id	action	question_id	answer_id	q_num	timestamp
5	show	285	null	1	123
5	answer	285	124124	1	124
5	show	369	null	2	125
5	skip	369	null	2	126

Output:

survey_log
285

Explanation:

Question 285 was showed 1 time and answered 1 time. The answer rate of question 285
 Question 369 was showed 1 time and was not answered. The answer rate of question 369
 Question 285 has the highest answer rate.



```

SELECT
B.QUESTION_ID AS SURVEY_LOG
FROM
(
  SELECT
  QUESTION_ID,
  DENSE_RANK() OVER(ORDER BY A.RES_COUNT DESC, QUESTION_ID ASC) AS CRANK
  FROM
  (
    SELECT
    QUESTION_ID,
    SUM(CASE WHEN ACTION = 'answer' THEN 1 ELSE 0 END)/
    SUM(CASE WHEN ACTION = 'show' THEN 1 ELSE 0 END) AS RES_COUNT
    FROM SURVEYLOG
    GROUP BY QUESTION_ID
  ) A
)B
WHERE B.CRANK = 1
  
```

580. Count Student Number in Departments ↗



Table: Student

Column Name	Type
student_id	int
student_name	varchar
gender	varchar
dept_id	int

student_id is the primary key column for this table.

dept_id is a foreign key to dept_id in the Department tables.

Each row of this table indicates the name of a student, their gender, and the id of



Table: Department

Column Name	Type
dept_id	int
dept_name	varchar

dept_id is the primary key column for this table.

Each row of this table contains the id and the name of a department.

Write an SQL query to report the respective department name and number of students majoring in each department for all departments in the Department table (even ones with no current students).

Return the result table **ordered** by student_number **in descending order**. In case of a tie, order them by dept_name **alphabetically**.

The query result format is in the following example.

Example 1:

Input:

Student table:

student_id	student_name	gender	dept_id
1	Jack	M	1
2	Jane	F	1
3	Mark	M	2

Department table:

dept_id	dept_name
1	Engineering
2	Science
3	Law

Output:

dept_name	student_number
Engineering	2
Science	1
Law	0

```

SELECT
D.DEPT_NAME,
CASE
    WHEN COUNT(DISTINCT S.STUDENT_ID) IS NULL THEN 0
    ELSE COUNT(DISTINCT S.STUDENT_ID)
END AS STUDENT_NUMBER
FROM DEPARTMENT AS D LEFT JOIN STUDENT AS S
ON D.DEPT_ID = S.DEPT_ID
GROUP BY D.DEPT_NAME
ORDER BY STUDENT_NUMBER DESC, DEPT_NAME ASC

```

584. Find Customer Referee ↗



Table: Customer

Column Name	Type
id	int
name	varchar
referee_id	int

`id` is the primary key column for this table.

Each row of this table indicates the id of a customer, their name, and the id of the

Write an SQL query to report the names of the customer that are **not referred by** the customer with `id = 2`.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Customer table:

id	name	referee_id
1	Will	null
2	Jane	null
3	Alex	2
4	Bill	null
5	Zack	1
6	Mark	2

Output:

name
Will
Jane
Bill
Zack

```
select name from customer where referee_id not in (2) or referee_id is null
```

586. Customer Placing the Largest Number of Orders



Table: Orders

```
+-----+-----+
| Column Name | Type   |
+-----+-----+
| order_number | int    |
| customer_number | int   |
+-----+-----+
```

order_number is the primary key for this table.

This table contains information about the order ID and the customer ID.

Write an SQL query to find the `customer_number` for the customer who has placed **the largest number of orders**.

The test cases are generated so that **exactly one customer** will have placed more orders than any other customer.

The query result format is in the following example.

Example 1:

Input:

Orders table:

```
+-----+-----+
| order_number | customer_number |
+-----+-----+
| 1            | 1              |
| 2            | 2              |
| 3            | 3              |
| 4            | 3              |
+-----+-----+
```

Output:

```
+-----+
| customer_number |
+-----+
| 3             |
+-----+
```

Explanation:

The customer with number 3 has two orders, which is greater than either customer 1 or 2. So the result is `customer_number` 3.



Follow up: What if more than one customer has the largest number of orders, can you find all the `customer_number` in this case?

- Mysql Based Solution using group by and count using sorting too

```
SELECT CUSTOMER_NUMBER
FROM ORDERS
GROUP BY CUSTOMER_NUMBER
ORDER BY COUNT(*) DESC LIMIT 1
```

595. Big Countries ↗

Table: World

Column Name	Type
name	varchar
continent	varchar
area	int
population	int
gdp	int

name is the primary key column for this table.

Each row of this table gives information about the name of a country, the continent

A country is **big** if:

- it has an area of at least three million (i.e., 3000000 km^2), or
- it has a population of at least twenty-five million (i.e., 25000000).

Write an SQL query to report the name, population, and area of the **big countries**.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

World table:

name	continent	area	population	gdp
Afghanistan	Asia	652230	25500100	20343000000
Albania	Europe	28748	2831741	12960000000
Algeria	Africa	2381741	37100000	188681000000
Andorra	Europe	468	78115	3712000000
Angola	Africa	1246700	20609294	100990000000

Output:

name	population	area
Afghanistan	25500100	652230
Algeria	37100000	2381741

```
select name, population, area
from world
where area >= 3000000 or
population >= 25000000
```

597. Friend Requests I: Overall Acceptance Rate ↗ ▾

Table: FriendRequest

Column Name	Type
sender_id	int
send_to_id	int
request_date	date

There is no primary key for this table, it may contain duplicates.

This table contains the ID of the user who sent the request, the ID of the user who

Table: RequestAccepted

Column Name	Type
requester_id	int
accepter_id	int
accept_date	date

There is no primary key for this table, it may contain duplicates.

This table contains the ID of the user who sent the request, the ID of the user who

Write an SQL query to find the overall acceptance rate of requests, which is the number of acceptance divided by the number of requests. Return the answer rounded to 2 decimal places.

Note that:

- The accepted requests are not necessarily from the table `friend_request`. In this case, Count the total accepted requests (no matter whether they are in the original requests), and divide it by the number of requests to get the acceptance rate.
- It is possible that a sender sends multiple requests to the same receiver, and a request could be accepted more than once. In this case, the 'duplicated' requests or acceptances are only counted once.
- If there are no requests at all, you should return 0.00 as the `accept_rate`.

The query result format is in the following example.

Example 1:

Input:

FriendRequest table:

sender_id	send_to_id	request_date
1	2	2016/06/01
1	3	2016/06/01
1	4	2016/06/01
2	3	2016/06/02
3	4	2016/06/09

RequestAccepted table:

requester_id	accepter_id	accept_date
1	2	2016/06/03
1	3	2016/06/08
2	3	2016/06/08
3	4	2016/06/09
3	4	2016/06/10

Output:

accept_rate
0.8

Explanation:

There are 4 unique accepted requests, and there are 5 requests in total. So the rate

Follow up:

- Could you write a query to return the acceptance rate for every month?
 - Could you write a query to return the cumulative acceptance rate for every day?
-
- Initial Draft

```

SELECT
CASE
    WHEN SUM(A.TOTAL_COUNT) IS NOT NULL THEN
        ROUND(SUM(A.TOTAL_COUNT)/ (SELECT COUNT(*) FROM REQUESTACCEPTED),2)
    ELSE 0.00
END AS ACCEPT_RATE
FROM
(
    SELECT
        COUNT(DISTINCT REQUESTER_ID, ACCEPTER_ID) AS TOTAL_COUNT
    FROM REQUESTACCEPTED
    GROUP BY REQUESTER_ID, ACCEPTER_ID
) A

```

- Final Solution.

SELECT CASE

```

WHEN SUM(A.TOTAL_COUNT) IS NOT NULL THEN ROUND(SUM(A.TOTAL_COUNT) / (SELECT SUM(B.RES)
FROM (SELECT COUNT(DISTINCT SENDER_ID,SEND_TO_ID) AS RES FROM FRIENDREQUEST GROUP BY
SENDER_ID, SEND_TO_ID ) B),2) ELSE 0.00 END AS ACCEPT_RATE FROM
(
    SELECT COUNT(DISTINCT RQ.REQUESTER_ID, RQ.ACCEPTER_ID) AS TOTAL_COUNT
    FROM REQUESTACCEPTED AS RQ GROUP BY RQ.REQUESTER_ID, RQ.ACCEPTER_ID ) A

```

...

602. Friend Requests II: Who Has the Most Friends



Table: RequestAccepted

Column Name	Type
requester_id	int
accepter_id	int
accept_date	date

(requester_id, accepter_id) is the primary key for this table.

This table contains the ID of the user who sent the request, the ID of the user who

Write an SQL query to find the people who have the most friends and the most friends number.

The test cases are generated so that only one person has the most friends.

The query result format is in the following example.

Example 1:

Input:

RequestAccepted table:

requester_id	accepter_id	accept_date
1	2	2016/06/03
1	3	2016/06/08
2	3	2016/06/08
3	4	2016/06/09

Output:

id	num
3	3

Explanation:

The person with id 3 is a friend of people 1, 2, and 4, so he has three friends in total.

Follow up: In the real world, multiple people could have the same most number of friends. Could you find all these people in this case?

- Solution Using SUM + GROUP BY

```

SELECT
A.ID,
SUM(FCOUNT) AS NUM
FROM
(
    SELECT
        REQUESTER_ID AS ID,
        COUNT(*) AS FCOUNT
    FROM REQUESTACCEPTED
    GROUP BY REQUESTER_ID

    UNION ALL

    SELECT
        ACCEPTER_ID AS ID,
        COUNT(*) AS FCOUNT
    FROM REQUESTACCEPTED
    GROUP BY ACCEPTER_ID
) A
GROUP BY A.ID
ORDER BY NUM DESC LIMIT 1

```

603. Consecutive Available Seats ↗

Table: Cinema

Column Name	Type
seat_id	int
free	bool

seat_id is an auto-increment primary key column for this table.

Each row of this table indicates whether the i^{th} seat is free or not. 1 means free while 0 means occupied.

Write an SQL query to report all the consecutive available seats in the cinema.

Return the result table **ordered** by `seat_id` **in ascending order**.

The test cases are generated so that more than two seats are consecutively available.

The query result format is in the following example.

Example 1:

Input:

Cinema table:

seat_id	free
1	1
2	0
3	1
4	1
5	1

Output:

seat_id
3
4
5

```
SELECT DISTINCT C.SEAT_ID
FROM CINEMA AS C JOIN CINEMA AS CN
ON C.FREE = CN.FREE
WHERE C.FREE = 1 AND CN.FREE = 1 AND ABS(C.SEAT_ID - CN.SEAT_ID) = 1
ORDER BY C.SEAT_ID
```

605. Can Place Flowers ↗

You have a long flowerbed in which some of the plots are planted, and some are not. However, flowers cannot be planted in **adjacent** plots.

Given an integer array `flowerbed` containing 0's and 1's, where 0 means empty and 1 means not empty, and an integer `n`, return if `n` new flowers can be planted in the `flowerbed` without violating the no-adjacent-flowers rule.

Example 1:

Input: flowerbed = [1,0,0,0,1], n = 1
Output: true

Example 2:

Input: flowerbed = [1,0,0,0,1], n = 2

Output: false

Constraints:

- $1 \leq \text{flowerbed.length} \leq 2 * 10^4$
- $\text{flowerbed}[i]$ is 0 or 1.
- There are no two adjacent flowers in `flowerbed`.
- $0 \leq n \leq \text{flowerbed.length}$

** Assume it as, a Long Garden with, a Infinite length **

- After a bit of brainstorming, i have decided to pick the 1st index, and perform required transformations.

```
class Solution:
    def canPlaceFlowers(self, flowerbed: List[int], n: int) -> bool:
        log = len(flowerbed)
        count = 0
        for i in range(0,log):
            # this is the edge case for last index and first index
            if (i == log-1 or i == 1) and flowerbed[i-1] == 0 and flowerbed[i] == 0:
                flowerbed[i-1] = 1
                count+=1
            elif (flowerbed[i-1] == 0 and flowerbed[i] == 0 and flowerbed[i+1] == 0) and i != log-1:
                # this is for the middle zeroes.
                flowerbed[i] = 1
                count +=1
        if count >= n:
            return True
        else:
            return False
```

607. Sales Person ↗

Table: SalesPerson

Column Name	Type
sales_id	int
name	varchar
salary	int
commission_rate	int
hire_date	date

sales_id is the primary key column for this table.

Each row of this table indicates the name and the ID of a salesperson alongside their salary and commission rate.

Table: Company

Column Name	Type
com_id	int
name	varchar
city	varchar

com_id is the primary key column for this table.

Each row of this table indicates the name and the ID of a company and the city in which it is located.

Table: Orders

Column Name	Type
order_id	int
order_date	date
com_id	int
sales_id	int
amount	int

order_id is the primary key column for this table.

com_id is a foreign key to com_id from the Company table.

sales_id is a foreign key to sales_id from the SalesPerson table.

Each row of this table contains information about one order. This includes the ID of the order, the date it was placed, the ID of the company it was placed by, the ID of the salesperson who took the order, and the total amount of the order.

Write an SQL query to report the names of all the salespersons who did not have any orders related to the company with the name "**RED**".

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

SalesPerson table:

sales_id	name	salary	commission_rate	hire_date
1	John	100000	6	4/1/2006
2	Amy	12000	5	5/1/2010
3	Mark	65000	12	12/25/2008
4	Pam	25000	25	1/1/2005
5	Alex	5000	10	2/3/2007

Company table:

com_id	name	city
1	RED	Boston
2	ORANGE	New York
3	YELLOW	Boston
4	GREEN	Austin

Orders table:

order_id	order_date	com_id	sales_id	amount
1	1/1/2014	3	4	10000
2	2/1/2014	4	5	5000
3	3/1/2014	1	1	50000
4	4/1/2014	1	4	25000

Output:

name
Amy
Mark
Alex

Explanation:

According to orders 3 and 4 in the Orders table, it is easy to tell that only salesp

- MSSQL Solution To the Above Problem.

```

SELECT NAME
FROM
SALESPERSON
WHERE SALES_ID NOT IN
(
  SELECT O.SALES_ID FROM ORDERS AS O LEFT JOIN COMPANY C
  ON O.COM_ID = C.COM_ID
  WHERE C.NAME = 'RED'
)

```

608. Tree Node ↗



Table: Tree

Column Name	Type
id	int
p_id	int

id is the primary key column for this table.

Each row of this table contains information about the id of a node and the id of its parent node. The given structure is always a valid tree.



Each node in the tree can be one of three types:

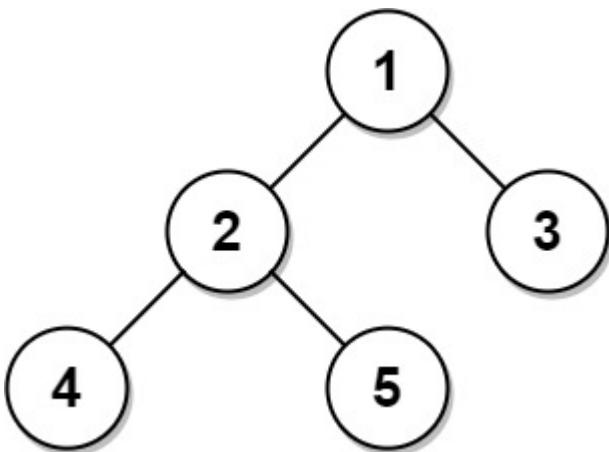
- "**Leaf**": if the node is a leaf node.
- "**Root**": if the node is the root of the tree.
- "**Inner**": If the node is neither a leaf node nor a root node.

Write an SQL query to report the type of each node in the tree.

Return the result table **ordered** by **id in ascending order**.

The query result format is in the following example.

Example 1:

**Input:**

Tree table:

id	p_id
1	null
2	1
3	1
4	2
5	2

Output:

id	type
1	Root
2	Inner
3	Leaf
4	Leaf
5	Leaf

Explanation:

Node 1 is the root node because its parent node is null and it has child nodes 2 and 3.

Node 2 is an inner node because it has parent node 1 and child node 4 and 5.

Nodes 3, 4, and 5 are leaf nodes because they have parent nodes and they do not have

**Example 2:**

Input:

Tree table:

id	p_id
1	null

Output:

id	type
1	Root

Explanation: If there is only one node on the tree, you only need to output its root

*Applying the Algorithmic Thinking Helps a Lot....

```

SELECT ID, 'Root' as type
FROM TREE
WHERE P_ID IS NULL

UNION

(
SELECT DISTINCT
T1.ID, 'Inner' as type
FROM TREE AS T1 JOIN TREE AS T2
ON T1.ID = T2.P_ID

EXCEPT

SELECT ID, 'Inner' as type
FROM TREE
WHERE P_ID IS NULL

)

UNION

SELECT ID , 'Leaf' AS type
FROM TREE
WHERE P_ID IS NOT NULL AND
ID NOT IN
(
SELECT DISTINCT T1.ID
FROM TREE AS T1 JOIN TREE AS T2
ON T1.ID = T2.P_ID
)

```

610. Triangle Judgement ↗

Table: Triangle

Column Name	Type
x	int
y	int
z	int

(x, y, z) is the primary key column for this table.

Each row of this table contains the lengths of three line segments.

Write an SQL query to report for every three line segments whether they can form a triangle.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Triangle table:

x	y	z
13	15	30
10	20	15

Output:

x	y	z	triangle
13	15	30	No
10	20	15	Yes

```

SELECT
X, Y, Z,
CASE
    WHEN X + Y > Z AND Y+Z > X AND X + Z > Y THEN 'Yes'
    ELSE 'No'
END AS TRIANGLE
FROM TRIANGLE

```

613. Shortest Distance in a Line ↗

Table: Point

Column Name	Type
x	int

x is the primary key column for this table.

Each row of this table indicates the position of a point on the X-axis.

Write an SQL query to report the shortest distance between any two points from the `Point` table.

The query result format is in the following example.

Example 1:

Input:

Point table:

	x
1	-1
2	0
3	2

Output:

	shortest
1	1

Explanation: The shortest distance is between points -1 and 0 which is $|(-1) - 0| = 1$



Follow up: How could you optimize your query if the `Point` table is ordered **in ascending order**?

- A good Solution.

```
# Solution Using Lag + Order By
SELECT
MIN(FIN.RES) AS SHORTEST
FROM
(
    SELECT
        ABS(TEMP.X - LAG(TEMP.X,1) OVER (ORDER BY X)) AS RES
    FROM
    (
        SELECT *
        FROM POINT
        ORDER BY X
    ) AS TEMP
) AS FIN
```



614. Second Degree Follower ↗

Table: Follow

Column Name	Type
followee	varchar
follower	varchar

(followee, follower) is the primary key column for this table.

Each row of this table indicates that the user follower follows the user followee on There will not be a user following themself.



A **second-degree follower** is a user who:

- follows at least one user, and
- is followed by at least one user.

Write an SQL query to report the **second-degree users** and the number of their followers.

Return the result table **ordered** by follower **in alphabetical order**.

The query result format is in the following example.

Example 1:

Input:

Follow table:

followee	follower
Alice	Bob
Bob	Cena
Bob	Donald
Donald	Edward

Output:

follower	num
Bob	2
Donald	1

Explanation:

User Bob has 2 followers. Bob is a second-degree follower because he follows Alice,
User Donald has 1 follower. Donald is a second-degree follower because he follows Bob
User Alice has 1 follower. Alice is not a second-degree follower because she does no

- Using INTERSECTION to Solve the problem

```

SELECT
FOLLOWEE AS FOLLOWER,
COUNT(*) AS NUM
FROM FOLLOW
WHERE FOLLOWEE IN
(
    SELECT FOLLOWEE AS TEM
    FROM FOLLOW
    GROUP BY FOLLOWEE
    HAVING COUNT(*) >= 1

    INTERSECT

    SELECT FOLLOWER AS TEM
    FROM FOLLOW
    GROUP BY FOLLOWER
    HAVING COUNT(*) >= 1
)
GROUP BY FOLLOWEE

```

619. Biggest Single Number ↗

Table: MyNumbers

Column Name	Type
num	int

There is no primary key for this table. It may contain duplicates.
Each row of this table contains an integer.

A **single number** is a number that appeared only once in the `MyNumbers` table.

Write an SQL query to report the largest **single number**. If there is no **single number**, report `null`.

The query result format is in the following example.

Example 1:**Input:**

MyNumbers table:

num
8
8
3
3
1
4
5
6

Output:

num
6

Explanation: The single numbers are 1, 4, 5, and 6.

Since 6 is the largest single number, we return it.

Example 2:

Input:

MyNumbers table:

num
8
8
7
7
3
3
3

Output:

num
null

Explanation: There are no single numbers in the input table so we return null.

```
SELECT IFNULL(MAX(NUM),NULL) AS NUM
FROM (SELECT NUM FROM MYNUMBERS GROUP BY NUM HAVING COUNT(*) = 1)
```

626. Exchange Seats ↗



Table: Seat

Column Name	Type
id	int
name	varchar

id is the primary key column for this table.

Each row of this table indicates the name and the ID of a student.

id is a continuous increment.

Write an SQL query to swap the seat id of every two consecutive students. If the number of students is odd, the id of the last student is not swapped.

Return the result table ordered by **id in ascending order**.

The query result format is in the following example.

Example 1:**Input:**

Seat table:

id	student
1	Abbot
2	Doris
3	Emerson
4	Green
5	Jeames

Output:

id	student
1	Doris
2	Abbot
3	Green
4	Emerson
5	Jeames

Explanation:

Note that if the number of students is odd, there is no need to change the last one'

```

SELECT
ID,
CASE
    WHEN ID % 2 = 0 THEN LAG(STUDENT,1) OVER()
    WHEN ID % 2 != 0 AND
        ID != (SELECT COUNT(*) FROM SEAT)
        THEN LEAD(STUDENT,1) OVER()
    ELSE STUDENT
END AS STUDENT
FROM SEAT

```

627. Swap Salary ↗

Table: Salary

Column Name	Type
id	int
name	varchar
sex	ENUM
salary	int

id is the primary key for this table.

The sex column is ENUM value of type ('m', 'f').

The table contains information about an employee.

Write an SQL query to swap all 'f' and 'm' values (i.e., change all 'f' values to 'm' and vice versa) with a **single update statement** and no intermediate temporary tables.

Note that you must write a single update statement, **do not** write any select statement for this problem.

The query result format is in the following example.

Example 1:

Input:

Salary table:

id	name	sex	salary
1	A	m	2500
2	B	f	1500
3	C	m	5500
4	D	f	500

Output:

id	name	sex	salary
1	A	f	2500
2	B	m	1500
3	C	f	5500
4	D	m	500

Explanation:

(1, A) and (3, C) were changed from 'm' to 'f'.

(2, B) and (4, D) were changed from 'f' to 'm'.

- This kind of problems are Quite Interesting and Important where it enhances developer thinking capability and his ingenuity. ``

- We Can Use Case Statement Inorder to Solve This Problem. UPDATE SALARY SET SEX = CASE

```

WHEN SEX = 'm' THEN 'f'
WHEN SEX = 'f'  THEN 'm'
END

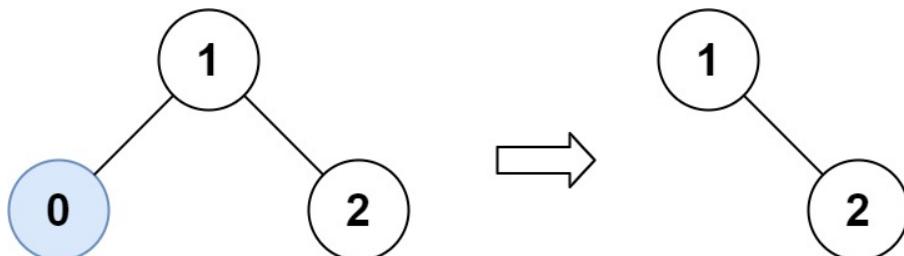
```

669. Trim a Binary Search Tree ↗

Given the `root` of a binary search tree and the lowest and highest boundaries as `low` and `high`, trim the tree so that all its elements lies in `[low, high]`. Trimming the tree should **not** change the relative structure of the elements that will remain in the tree (i.e., any node's descendant should remain a descendant). It can be proven that there is a **unique answer**.

Return *the root of the trimmed binary search tree*. Note that the root may change depending on the given bounds.

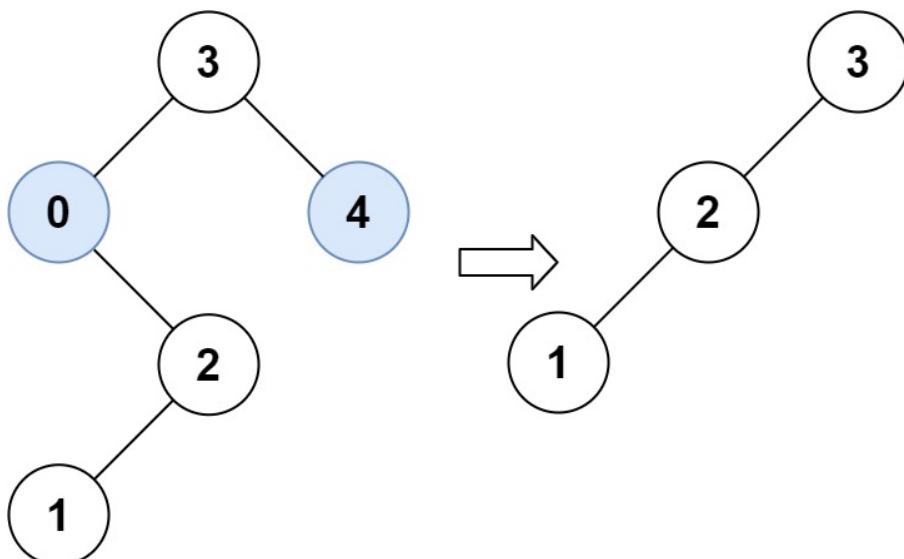
Example 1:



Input: `root = [1,0,2], low = 1, high = 2`

Output: `[1,null,2]`

Example 2:



Input: root = [3,0,4,null,2,null,null,1], low = 1, high = 3
Output: [3,2,null,1]

Constraints:

- The number of nodes in the tree is in the range $[1, 10^4]$.
- $0 \leq \text{Node.val} \leq 10^4$
- The value of each node in the tree is **unique**.
- `root` is guaranteed to be a valid binary search tree.
- $0 \leq \text{low} \leq \text{high} \leq 10^4$

```
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution(object):
    def trimBST(self, root, L, R):
        if root is None:
            return
        # it means if the present value is very much greater than the
        Right bound neglect it by traversing it to the left side
        if root.val > R:
            return self.trimBST(root.left,L,R)
        if root.val < L:
            return self.trimBST(root.right,L,R)
        root.left=self.trimBST(root.left,L,R)
        root.right=self.trimBST(root.right,L,R)
        return root
```

680. Valid Palindrome II ↗

Given a string `s`, return `true` if the `s` can be palindrome after deleting **at most one** character from it.

Example 1:

Input: s = "aba"
Output: true

Example 2:

Input: s = "abca"
Output: true
Explanation: You could delete the character 'c'.

Example 3:

Input: s = "abc"
Output: false

Constraints:

- $1 \leq s.length \leq 10^5$
- s consists of lowercase English letters.

...

Algorithm:

First remove one character from the string
and check whether the generated string is palindrome or
not if yes then return True
else return False
...

TLE Solution

class Solution:

 def validPalindrome(self, s: str) -> bool:
 ...

Algorithm:

First remove one character from the string
and check whether the generated string is palindrome or
not if yes then return True
else return False
...

def check_palindrome(check):

 if check == check[::-1]:
 return True
 return None

O(N) Comparisons

for i in range(len(s)):

 if i == 0:

 check = s[i+1::]

 else:

 check = s[0:i] + s[i+1::]

 if check_palindrome(check):

 return True

return False

682. Baseball Game ↗

You are keeping score for a baseball game with strange rules. The game consists of several rounds, where the scores of past rounds may affect future rounds' scores.

At the beginning of the game, you start with an empty record. You are given a list of strings `ops`, where `ops[i]` is the i^{th} operation you must apply to the record and is one of the following:

1. An integer x - Record a new score of x .
2. "+" - Record a new score that is the sum of the previous two scores. It is guaranteed there will always be two previous scores.
3. "D" - Record a new score that is double the previous score. It is guaranteed there will always be a previous score.
4. "C" - Invalidate the previous score, removing it from the record. It is guaranteed there will always be a previous score.

Return *the sum of all the scores on the record*. The test cases are generated so that the answer fits in a 32-bit integer.

Example 1:

Input: ops = ["5", "2", "C", "D", "+"]

Output: 30

Explanation:

"5" - Add 5 to the record, record is now [5].
 "2" - Add 2 to the record, record is now [5, 2].
 "C" - Invalidate and remove the previous score, record is now [5].
 "D" - Add $2 * 5 = 10$ to the record, record is now [5, 10].
 "+" - Add $5 + 10 = 15$ to the record, record is now [5, 10, 15].
 The total sum is $5 + 10 + 15 = 30$.

Example 2:

Input: ops = ["5", "-2", "4", "C", "D", "9", "+", "+"]

Output: 27

Explanation:

"5" - Add 5 to the record, record is now [5].
 "-2" - Add -2 to the record, record is now [5, -2].
 "4" - Add 4 to the record, record is now [5, -2, 4].
 "C" - Invalidate and remove the previous score, record is now [5, -2].
 "D" - Add $2 * -2 = -4$ to the record, record is now [5, -2, -4].
 "9" - Add 9 to the record, record is now [5, -2, -4, 9].
 "+" - Add $-4 + 9 = 5$ to the record, record is now [5, -2, -4, 9, 5].
 "+" - Add $9 + 5 = 14$ to the record, record is now [5, -2, -4, 9, 5, 14].
 The total sum is $5 + -2 + -4 + 9 + 5 + 14 = 27$.

Example 3:

Input: ops = ["1", "C"]

Output: 0

Explanation:

"1" - Add 1 to the record, record is now [1].

"C" - Invalidate and remove the previous score, record is now [].

Since the record is empty, the total sum is 0.

Constraints:

- $1 \leq \text{ops.length} \leq 1000$
- $\text{ops}[i]$ is "C", "D", "+", or a string representing an integer in the range $[-3 * 10^4, 3 * 10^4]$.
- For operation "+", there will always be at least two previous scores on the record.
- For operations "C" and "D", there will always be at least one previous score on the record.

```
# O(N) Solution
class Solution:
    def calPoints(self, ops: List[str]) -> int:
        res = []
        for char in ops:
            if char == '+':
                res.append(res[-1] + res[-2])
            elif char == 'C':
                res.pop()
            elif char == 'D':
                val = res[-1] * 2
                res.append(val)
            else:
                res.append(int(char))

        return sum(res)
```

709. To Lower Case ↗

Given a string s , return the string after replacing every uppercase letter with the same lowercase letter.

Example 1:

Input: s = "Hello"

Output: "hello"

Example 2:

```
Input: s = "here"
Output: "here"
```

Example 3:

```
Input: s = "LOVELY"
Output: "lovely"
```

Constraints:

- $1 \leq s.length \leq 100$
- s consists of printable ASCII characters.

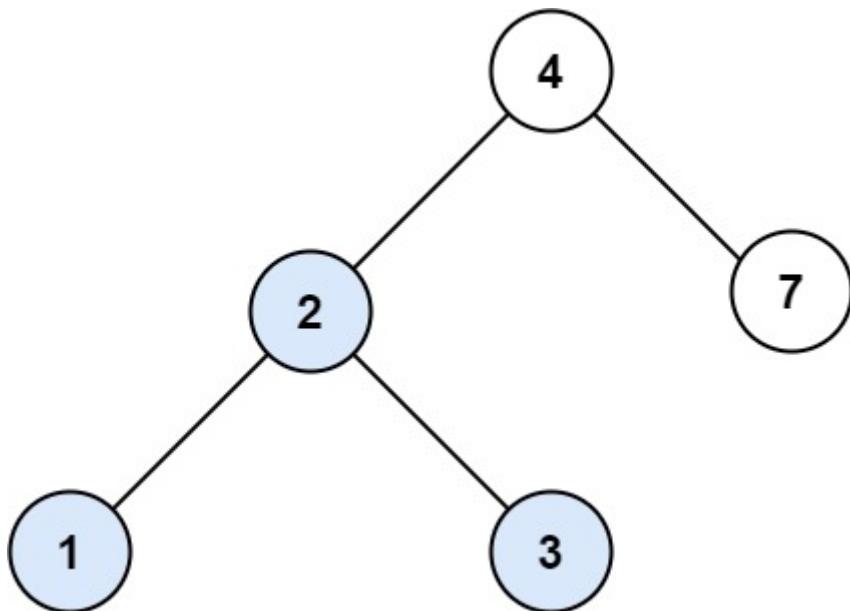
```
class Solution:
    def toLowerCase(self, s: str) -> str:
        res = ""
        for ind in range(len(s)):
            if ord(s[ind]) >= 65 and ord(s[ind]) <= 90:
                res += chr(ord(s[ind])+ 32)
            else:
                res += s[ind]
        return res
```

700. Search in a Binary Search Tree

You are given the `root` of a binary search tree (BST) and an integer `val`.

Find the node in the BST that the node's value equals `val` and return the subtree rooted with that node.
If such a node does not exist, return `null`.

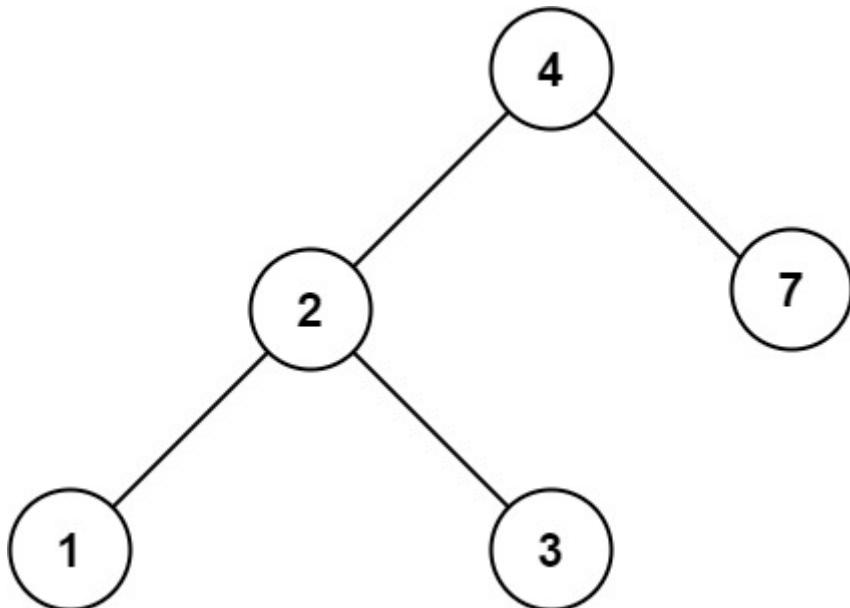
Example 1:



Input: root = [4,2,7,1,3], val = 2

Output: [2,1,3]

Example 2:



Input: root = [4,2,7,1,3], val = 5

Output: []

Constraints:

- The number of nodes in the tree is in the range [1, 5000].
- $1 \leq \text{Node.val} \leq 10^7$
- `root` is a binary search tree.
- $1 \leq \text{val} \leq 10^7$

```
# 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 __init__(self):
        self.temp = None
    def dfs(self,root,val):
        if root is None:
            return root
        if root.val == val:
            self.temp = root
            self.dfs(root.left,val)
            self.dfs(root.right,val)
    def searchBST(self, root: Optional[TreeNode], val: int) -> Optional[TreeNode]:
        self.dfs(root,val)
        return self.temp
```

705. Design HashSet ↗

Design a HashSet without using any built-in hash table libraries.

Implement `MyHashSet` class:

- `void add(key)` Inserts the value `key` into the HashSet.
- `bool contains(key)` Returns whether the value `key` exists in the HashSet or not.
- `void remove(key)` Removes the value `key` in the HashSet. If `key` does not exist in the HashSet, do nothing.

Example 1:

Input

```
["MyHashSet", "add", "add", "contains", "contains", "add", "contains", "remove", "co  
[], [1], [2], [1], [3], [2], [2], [2], [2]]
```

Output

```
[null, null, null, true, false, null, true, null, false]
```

Explanation

```
MyHashSet myHashSet = new MyHashSet();  
myHashSet.add(1);      // set = [1]  
myHashSet.add(2);      // set = [1, 2]  
myHashSet.contains(1); // return True  
myHashSet.contains(3); // return False, (not found)  
myHashSet.add(2);      // set = [1, 2]  
myHashSet.contains(2); // return True  
myHashSet.remove(2);   // set = [1]  
myHashSet.contains(2); // return False, (already removed)
```

**Constraints:**

- $0 \leq \text{key} \leq 10^6$
- At most 10^4 calls will be made to add , remove , and contains .

```

class MyHashSet:

    def __init__(self):
        self.hash = []

    def add(self, key: int) -> None:
        if key not in self.hash:
            self.hash.append(key)

    def remove(self, key: int) -> None:
        if key in self.hash:
            self.hash.remove(key)

    def contains(self, key: int) -> bool:
        if key in self.hash:
            return True
        else:
            return False

# Your MyHashSet object will be instantiated and called as such:
# obj = MyHashSet()
# obj.add(key)
# obj.remove(key)
# param_3 = obj.contains(key)

```

706. Design HashMap ↗

Design a HashMap without using any built-in hash table libraries.

Implement the `MyHashMap` class:

- `MyHashMap()` initializes the object with an empty map.
- `void put(int key, int value)` inserts a `(key, value)` pair into the HashMap. If the `key` already exists in the map, update the corresponding `value`.
- `int get(int key)` returns the `value` to which the specified `key` is mapped, or `-1` if this map contains no mapping for the `key`.
- `void remove(key)` removes the `key` and its corresponding `value` if the map contains the mapping for the `key`.

Example 1:

Input

```
["MyHashMap", "put", "put", "get", "get", "put", "get", "remove", "get"]
[], [1, 1], [2, 2], [1], [3], [2, 1], [2], [2], [2]]
```

Output

```
[null, null, null, 1, -1, null, 1, null, -1]
```

Explanation

```
MyHashMap myHashMap = new MyHashMap();
myHashMap.put(1, 1); // The map is now [[1,1]]
myHashMap.put(2, 2); // The map is now [[1,1], [2,2]]
myHashMap.get(1); // return 1, The map is now [[1,1], [2,2]]
myHashMap.get(3); // return -1 (i.e., not found), The map is now [[1,1], [2,2]]
myHashMap.put(2, 1); // The map is now [[1,1], [2,1]] (i.e., update the existing val)
myHashMap.get(2); // return 1, The map is now [[1,1], [2,1]]
myHashMap.remove(2); // remove the mapping for 2, The map is now [[1,1]]
myHashMap.get(2); // return -1 (i.e., not found), The map is now [[1,1]]
```

**Constraints:**

- $0 \leq \text{key, value} \leq 10^6$
- At most 10^4 calls will be made to `put`, `get`, and `remove`.

```
class MyHashMap:

    def __init__(self):
        self.hmap = {}

    def put(self, key: int, value: int) -> None:
        self.hmap[key] = value

    def get(self, key: int) -> int:
        if key in self.hmap:
            return self.hmap[key]
        else:
            return -1

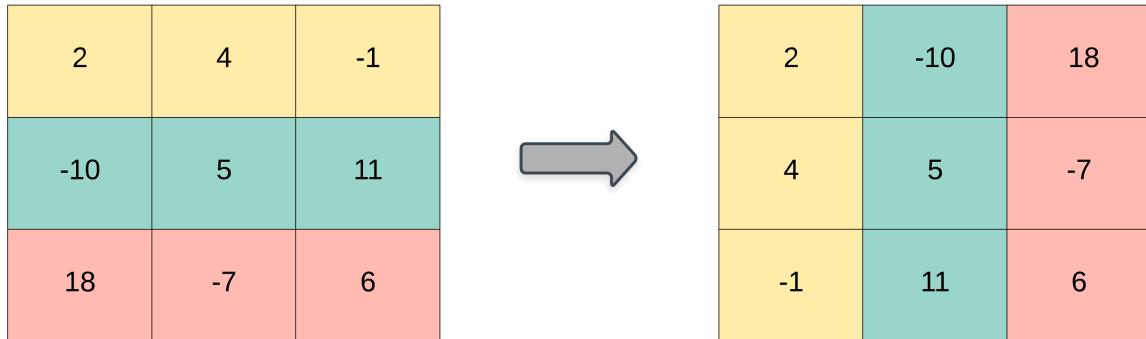
    def remove(self, key: int) -> None:
        if key in self.hmap:
            del self.hmap[key]

# Your MyHashMap object will be instantiated and called as such:
# obj = MyHashMap()
# obj.put(key,value)
# param_2 = obj.get(key)
# obj.remove(key)
```

867. Transpose Matrix ↗

Given a 2D integer array `matrix`, return *the transpose of matrix*.

The **transpose** of a matrix is the matrix flipped over its main diagonal, switching the matrix's row and column indices.



Example 1:

```
Input: matrix = [[1,2,3],[4,5,6],[7,8,9]]
Output: [[1,4,7],[2,5,8],[3,6,9]]
```

Example 2:

```
Input: matrix = [[1,2,3],[4,5,6]]
Output: [[1,4],[2,5],[3,6]]
```

Constraints:

- `m == matrix.length`
- `n == matrix[i].length`
- `1 <= m, n <= 1000`
- `1 <= m * n <= 105`
- `-109 <= matrix[i][j] <= 109`

- Matrix Solution For the Problem...

```

class Solution:
    def transpose(self, matrix: List[List[int]]) -> List[List[int]]:
        # in order to perform transpose of a matrix we can't directly
        # do it, and we need to have an auxiliary matrix
        # to perform this operations
        m = len(matrix[0])
        n = len(matrix)
        aux_matrix = [[None] * n for _ in range(m)]
        ...

        Never Ever create the list of lists by above declared approach..
        ...

        for i in range(len(matrix)):
            for j in range(len(matrix[0])):
                aux_matrix[j][i] = matrix[i][j]
        return aux_matrix

```

876. Middle of the Linked List ↗

Given the `head` of a singly linked list, return *the middle node of the linked list*.

If there are two middle nodes, return **the second middle** node.

Example 1:

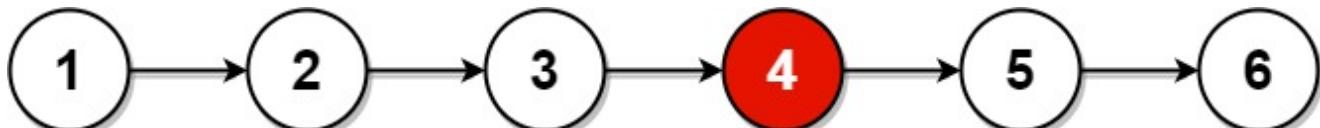


Input: head = [1,2,3,4,5]

Output: [3,4,5]

Explanation: The middle node of the list is node 3.

Example 2:



Input: head = [1,2,3,4,5,6]

Output: [4,5,6]

Explanation: Since the list has two middle nodes with values 3 and 4, we return the

Constraints:

- The number of nodes in the list is in the range [1, 100].
- $1 \leq \text{Node.val} \leq 100$

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def middleNode(self, head: Optional[ListNode]) -> Optional[ListNode]:
        root = head
        res = []
        while root:
            res.append(root.val)
            root = root.next

        root = ListNode(-1)
        root2 = root
        for val in res[len(res)//2:]:
            if root.val == -1:
                root.val = val
            else:
                root.next = ListNode(val)
                root = root.next
        return root2
```

896. Monotonic Array ↗

An array is **monotonic** if it is either monotone increasing or monotone decreasing.

An array `nums` is monotone increasing if for all $i \leq j$, $\text{nums}[i] \leq \text{nums}[j]$. An array `nums` is monotone decreasing if for all $i \leq j$, $\text{nums}[i] \geq \text{nums}[j]$.

Given an integer array `nums`, return `true` if the given array is monotonic, or `false` otherwise.

Example 1:

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

Example 2:

Input: `nums = [6,5,4,4]`
Output: `true`

Example 3:

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

Constraints:

- $1 \leq \text{nums.length} \leq 10^5$
- $-10^5 \leq \text{nums}[i] \leq 10^5$

Failed Approach

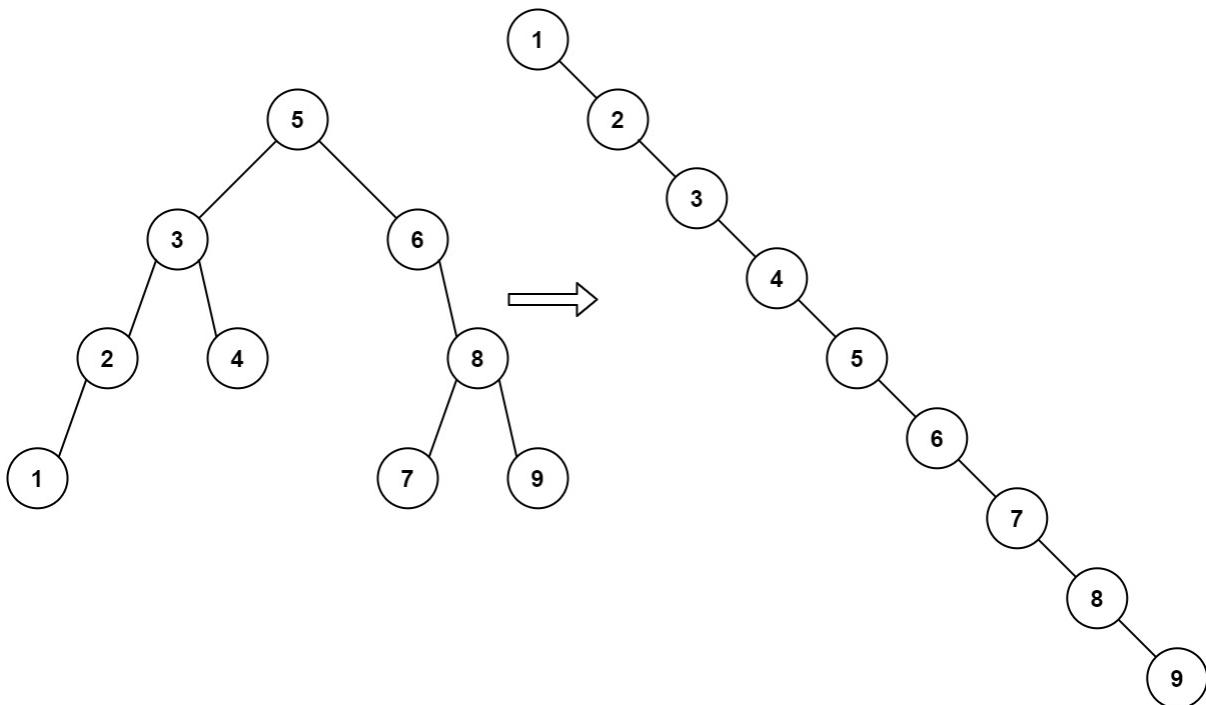
```
class Solution:
    def isMonotonic(self, nums: List[int]) -> bool:
        count = 1
        if len(nums) == 1:
            return True
        for ind in range(1, len(nums)):
            if nums[ind-1] >= nums[ind]:
                count +=1
            elif nums[ind-1] <= nums[ind]:
                count +=1

        print(count)
        if count == len(nums):
            return True
        return False
```

897. Increasing Order Search Tree ↗

Given the `root` of a binary search tree, rearrange the tree in **in-order** so that the leftmost node in the tree is now the root of the tree, and every node has no left child and only one right child.

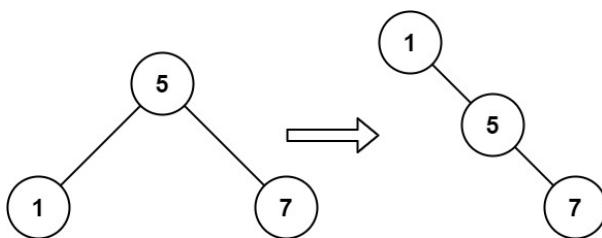
Example 1:



Input: root = [5,3,6,2,4,null,8,1,null,null,null,7,9]

Output: [1,null,2,null,3,null,4,null,5,null,6,null,7,null,8,null,9]

Example 2:



Input: root = [5,1,7]

Output: [1,null,5,null,7]

Constraints:

- The number of nodes in the given tree will be in the range [1, 100].
- $0 \leq \text{Node.val} \leq 1000$

```

def increasingBST(self, root):
    temp=[]
    self.inorder(root,temp)
    cur=root=TreeNode(temp[0])
    for i in range(1,len(temp)):
        cur.right=TreeNode(temp[i])
        cur=cur.right
    return root

def inorder(self,root,temp):
    if root is None:
        return
    self.inorder(root.left,temp)
    temp.append(root.val)
    self.inorder(root.right,temp)

```

953. Verifying an Alien Dictionary ↗

In an alien language, surprisingly, they also use English lowercase letters, but possibly in a different order . The order of the alphabet is some permutation of lowercase letters.

Given a sequence of words written in the alien language, and the order of the alphabet, return true if and only if the given words are sorted lexicographically in this alien language.

Example 1:

Input: words = ["hello", "leetcode"], order = "hlabcdegijklnopqrstuvwxyz"

Output: true

Explanation: As 'h' comes before 'l' in this language, then the sequence is sorted.

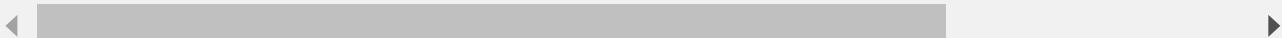


Example 2:

Input: words = ["word", "world", "row"], order = "worldabcefghijklmnopqrstuvwxyz"

Output: false

Explanation: As 'd' comes after 'l' in this language, then words[0] > words[1], hence the sequence is not sorted.

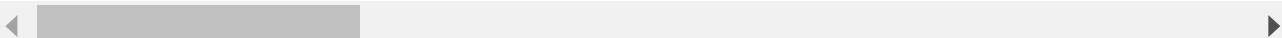


Example 3:

Input: words = ["apple", "app"], order = "abcdefghijklmnopqrstuvwxyz"

Output: false

Explanation: The first three characters "app" match, and the second string is shorter, so it is lexicographically smaller.



Constraints:

- $1 \leq \text{words.length} \leq 100$
- $1 \leq \text{words}[i].length \leq 20$
- $\text{order.length} == 26$
- All characters in `words[i]` and `order` are English lowercase letters.

```
class Solution:
    def isAlienSorted(self, words: List[str], order: str) -> bool:
        ordering = {
            k:v for v, k in enumerate(order)
        }
        def check(a,b):
            for i , j in itertools.zip_longest(a, b, fillvalue="_"):
                if ordering.get(i,-1) < ordering.get(j,-1):
                    return True
                elif ordering.get(i,-1) == ordering.get(j,-1):
                    continue
                else:
                    return False

            for w1, w2 in zip(words[:-1],words[1:]):
                if check(w1,w2) is None or check(w1,w2):
                    continue
                else:
                    return False
        return True
```

976. Largest Perimeter Triangle ↗

Given an integer array `nums` , return *the largest perimeter of a triangle with a non-zero area, formed from three of these lengths*. If it is impossible to form any triangle of a non-zero area, return `0` .

Example 1:

Input: `nums = [2,1,2]`
Output: 5

Example 2:

Input: `nums = [1,2,1]`
Output: 0

Constraints:

- $3 \leq \text{nums.length} \leq 10^4$
- $1 \leq \text{nums}[i] \leq 10^6$

*Initially i thought of this problem was hard *But the Basics, required to solve this one is Triangle Geometry i.e whether a traingle is valid or not **

```
class Solution:
    def largestPerimeter(self, nums: List[int]) -> int:
        nums.sort()
        perimeter = []
        for i in range(2,len(nums)):
            if nums[i-2] + nums[i-1] > nums[i]:
                perimeter.append(nums[i-2]+nums[i-1]+nums[i])
        return max(perimeter) if len(perimeter) > 0 else 0
```

989. Add to Array-Form of Integer ↗



The **array-form** of an integer `num` is an array representing its digits in left to right order.

- For example, for `num = 1321` , the array form is `[1,3,2,1]` .

Given `num` , the **array-form** of an integer, and an integer `k` , return *the array-form of the integer num + k* .

Example 1:

```
Input: num = [1,2,0,0], k = 34
Output: [1,2,3,4]
Explanation: 1200 + 34 = 1234
```

Example 2:

```
Input: num = [2,7,4], k = 181
Output: [4,5,5]
Explanation: 274 + 181 = 455
```

Example 3:

Input: num = [2,1,5], k = 806

Output: [1,0,2,1]

Explanation: 215 + 806 = 1021

Constraints:

- $1 \leq \text{num.length} \leq 10^4$
- $0 \leq \text{num}[i] \leq 9$
- num does not contain any leading zeros except for the zero itself.
- $1 \leq k \leq 10^4$

```
class Solution:
    def addToArrayForm(self, num: List[int], k: int) -> List[int]:
        s = ""
        for val in num:
            s+=str(val)
        s = str(int(s) + k)
        num.clear()
        for char in s:
            num.append(int(char))
        return num
```

1022. Sum of Root To Leaf Binary Numbers ↗ ▾

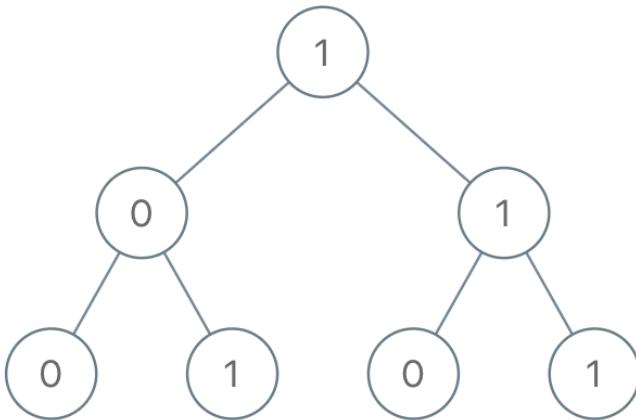
You are given the `root` of a binary tree where each node has a value `0` or `1`. Each root-to-leaf path represents a binary number starting with the most significant bit.

- For example, if the path is `0 -> 1 -> 1 -> 0 -> 1`, then this could represent `01101` in binary, which is `13`.

For all leaves in the tree, consider the numbers represented by the path from the root to that leaf. Return *the sum of these numbers*.

The test cases are generated so that the answer fits in a **32-bits** integer.

Example 1:



Input: root = [1,0,1,0,1,0,1]

Output: 22

Explanation: (100) + (101) + (110) + (111) = 4 + 5 + 6 + 7 = 22

Example 2:

Input: root = [0]

Output: 0

Constraints:

- The number of nodes in the tree is in the range [1, 1000].
 - Node.val is 0 or 1.
-
- Go to depth if null is found then use a string to make a variable and sum it by converting it into binary..

```

# 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 sumRootToLeaf(self, root: Optional[TreeNode]) -> int:
        self.total = 0
        def calculate(val,ptr):
            if ptr is None:
                return
            elif ptr.right is None and ptr.left is None:
                val += str(ptr.val)
                self.total += int(val,2)
                return
            else:
                val += str(ptr.val)
                calculate(val,ptr.left)
                calculate(val,ptr.right)

        calculate('',root)
        return self.total

```

1046. Last Stone Weight ↗

You are given an array of integers `stones` where `stones[i]` is the weight of the i^{th} stone.

We are playing a game with the stones. On each turn, we choose the **heaviest two stones** and smash them together. Suppose the heaviest two stones have weights x and y with $x \leq y$. The result of this smash is:

- If $x == y$, both stones are destroyed, and
- If $x != y$, the stone of weight x is destroyed, and the stone of weight y has new weight $y - x$.

At the end of the game, there is **at most one** stone left.

Return *the weight of the last remaining stone*. If there are no stones left, return `0`.

Example 1:

Input: stones = [2,7,4,1,8,1]

Output: 1

Explanation:

We combine 7 and 8 to get 1 so the array converts to [2,4,1,1,1] then,
we combine 2 and 4 to get 2 so the array converts to [2,1,1,1] then,
we combine 2 and 1 to get 1 so the array converts to [1,1,1] then,
we combine 1 and 1 to get 0 so the array converts to [1] then that's the value of th

Example 2:

Input: stones = [1]

Output: 1

Constraints:

- $1 \leq \text{stones.length} \leq 30$
- $1 \leq \text{stones}[i] \leq 1000$

You are given an array of integers stones where $\text{stones}[i]$ is the weight of the i th stone.

We are playing a game with the stones. On each turn, we choose the heaviest two stones and smash them together. Suppose the heaviest two stones have weights x and y with $x \leq y$. The result of this smash is:

If $x == y$, both stones are destroyed, and

If $x != y$, the stone of weight x is destroyed, and the stone of weight y has new weight $y - x$.

At the end of the game, there is at most one stone left.

Return the smallest possible weight of the left stone. If there are no stones left, return 0.

```

def lastStoneWeight(self, stones):
    while len(stones)>1:
        stones.sort(reverse=True)
        x, y=stones[0], stones[1]
        if x-y==0:
            if len(stones)==2:
                return 0
            else:
                del stones[0]
                del stones[0]
                continue
        else:
            stones.append(x-y)
            print(stones)
            del stones[0]
            del stones[0]
    return stones[-1]

```

1045. Customers Who Bought All Products ↗



Table: Customer

Column Name	Type
customer_id	int
product_key	int

There is no primary key for this table. It may contain duplicates.
 product_key is a foreign key to Product table.

Table: Product

Column Name	Type
product_key	int

product_key is the primary key column for this table.

Write an SQL query to report the customer ids from the Customer table that bought all the products in the Product table.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Customer table:

customer_id	product_key
1	5
2	6
3	5
3	6
1	6

Product table:

product_key
5
6

Output:

customer_id
1
3

Explanation:

The customers who bought all the products (5 and 6) are customers with IDs 1 and 3.

- CTE + Group By Based Solution

```

WITH AGG_COUNT AS
(
  SELECT
    CUSTOMER_ID,
    COUNT(DISTINCT PRODUCT_KEY) AS DCOUNT,
    (SELECT COUNT(*) FROM PRODUCT) AS PCOUNT
  FROM CUSTOMER
  GROUP BY CUSTOMER_ID
)

SELECT CUSTOMER_ID
FROM AGG_COUNT
WHERE DCOUNT = PCOUNT

```

1050. Actors and Directors Who Cooperated At Least Three Times ↗

Table: ActorDirector

Column Name	Type
actor_id	int
director_id	int
timestamp	int

timestamp is the primary key column for this table.

Write a SQL query for a report that provides the pairs `(actor_id, director_id)` where the actor has cooperated with the director at least three times.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

ActorDirector table:

actor_id	director_id	timestamp
1	1	0
1	1	1
1	1	2
1	2	3
1	2	4
2	1	5
2	1	6

Output:

actor_id	director_id
1	1

Explanation: The only pair is (1, 1) where they cooperated exactly 3 times.

- Mysql Solution Using Group By and Count

```
SELECT ACTOR_ID, DIRECTOR_ID
FROM ACTORDIRECTOR
GROUP BY ACTOR_ID, DIRECTOR_ID
HAVING COUNT(*) >= 3
```

- Wrong Solution Using Where condition

```
SELECT ACTOR_ID, DIRECTOR_ID
FROM ACTORDIRECTOR
WHERE ACTOR_ID = DIRECTOR_ID
HAVING COUNT(*) >= 3
```

1068. Product Sales Analysis I ↗



Table: Sales

Column Name	Type
sale_id	int
product_id	int
year	int
quantity	int
price	int

(sale_id, year) is the primary key of this table.

product_id is a foreign key to Product table.

Each row of this table shows a sale on the product product_id in a certain year.

Note that the price is per unit.

Table: Product

Column Name	Type
product_id	int
product_name	varchar

product_id is the primary key of this table.

Each row of this table indicates the product name of each product.

Write an SQL query that reports the `product_name`, `year`, and `price` for each `sale_id` in the `Sales` table.

Return the resulting table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Sales table:

sale_id	product_id	year	quantity	price
1	100	2008	10	5000
2	100	2009	12	5000
7	200	2011	15	9000

Product table:

product_id	product_name
100	Nokia
200	Apple
300	Samsung

Output:

product_name	year	price
Nokia	2008	5000
Nokia	2009	5000
Apple	2011	9000

Explanation:

From sale_id = 1, we can conclude that Nokia was sold for 5000 in the year 2008.

From sale_id = 2, we can conclude that Nokia was sold for 5000 in the year 2009.

From sale_id = 7, we can conclude that Apple was sold for 9000 in the year 2011.

- solution Using Left Join + Group BY

```

SELECT
P.PRODUCT_NAME,
S.YEAR,
S.PRICE
FROM PRODUCT AS P LEFT JOIN SALES AS S
ON P.PRODUCT_ID = S.PRODUCT_ID
WHERE S.PRODUCT_ID IS NOT NULL
GROUP BY S.SALE_ID
    
```

1069. Product Sales Analysis II ↗



Table: Sales

Column Name	Type
sale_id	int
product_id	int
year	int
quantity	int
price	int

(sale_id, year) is the primary key of this table.

product_id is a foreign key to Product table.

Each row of this table shows a sale on the product product_id in a certain year.

Note that the price is per unit.

Table: Product

Column Name	Type
product_id	int
product_name	varchar

product_id is the primary key of this table.

Each row of this table indicates the product name of each product.

Write an SQL query that reports the total quantity sold for every product id.

Return the resulting table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Sales table:

sale_id	product_id	year	quantity	price
1	100	2008	10	5000
2	100	2009	12	5000
7	200	2011	15	9000

Product table:

product_id	product_name
100	Nokia
200	Apple
300	Samsung

Output:

product_id	total_quantity
100	22
200	15

- Simple Solution Using Left join + Group By

/*⭐ Solution Using LeftJoin Runs Super Fast ⭐*/

```
SELECT P.PRODUCT_ID, SUM(S.QUANTITY) AS TOTAL_QUANTITY FROM PRODUCT AS P LEFT JOIN SALES
AS S ON P.PRODUCT_ID = S.PRODUCT_ID WHERE S.PRODUCT_ID IS NOT NULL GROUP BY P.PRODUCT_ID
``
```

1070. Product Sales Analysis III ↗



Table: Sales

Column Name	Type
sale_id	int
product_id	int
year	int
quantity	int
price	int

(sale_id, year) is the primary key of this table.

product_id is a foreign key to Product table.

Each row of this table shows a sale on the product product_id in a certain year.

Note that the price is per unit.

Table: Product

Column Name	Type
product_id	int
product_name	varchar

product_id is the primary key of this table.

Each row of this table indicates the product name of each product.

Write an SQL query that selects the **product id**, **year**, **quantity**, and **price** for the **first year** of every product sold.

Return the resulting table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Sales table:

sale_id	product_id	year	quantity	price
1	100	2008	10	5000
2	100	2009	12	5000
7	200	2011	15	9000

Product table:

product_id	product_name
100	Nokia
200	Apple
300	Samsung

Output:

product_id	first_year	quantity	price
100	2008	10	5000
200	2011	15	9000

- Simple rank based solution

```

SELECT
A.PRODUCT_ID,
A.YEAR AS FIRST_YEAR,
A.QUANTITY,
A.PRICE
FROM
(
    SELECT  *,
    DENSE_RANK() OVER(PARTITION BY PRODUCT_ID ORDER BY YEAR) AS CRANK
    FROM SALES
) A
WHERE A.CRANK = 1

```

1076. Project Employees II ↗



Table: Project

Column Name	Type
project_id	int
employee_id	int

(project_id, employee_id) is the primary key of this table.

employee_id is a foreign key to Employee table.

Each row of this table indicates that the employee with employee_id is working on the

Table: Employee

Column Name	Type
employee_id	int
name	varchar
experience_years	int

employee_id is the primary key of this table.

Each row of this table contains information about one employee.

Write an SQL query that reports all the **projects** that have the most employees.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Project table:

project_id	employee_id
1	1
1	2
1	3
2	1
2	4

Employee table:

employee_id	name	experience_years
1	Khaled	3
2	Ali	2
3	John	1
4	Doe	2

Output:

project_id
1

Explanation: The first project has 3 employees while the second one has 2.

- solution using rank can also use other techniques too.
- i.e using max function.

```
# Solution Using Rank
SELECT T1.PROJECT_ID
FROM
(
    SELECT
        P.PROJECT_ID,
        DENSE_RANK() OVER (ORDER BY COUNT(*) DESC) AS CRANK
    FROM PROJECT AS P
    GROUP BY PROJECT_ID
) AS T1
WHERE T1.CRANK = 1
```

1077. Project Employees III ↗



Table: Project

Column Name	Type
project_id	int
employee_id	int

(project_id, employee_id) is the primary key of this table.

employee_id is a foreign key to Employee table.

Each row of this table indicates that the employee with employee_id is working on the

Table: Employee

Column Name	Type
employee_id	int
name	varchar
experience_years	int

employee_id is the primary key of this table.

Each row of this table contains information about one employee.

Write an SQL query that reports the **most experienced** employees in each project. In case of a tie, report all employees with the maximum number of experience years.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Project table:

project_id	employee_id
1	1
1	2
1	3
2	1
2	4

Employee table:

employee_id	name	experience_years
1	Khaled	3
2	Ali	2
3	John	3
4	Doe	2

Output:

project_id	employee_id
1	1
1	3
2	1

Explanation: Both employees with id 1 and 3 have the most experience among the employees assigned to project 1.

```

SELECT
A.PROJECT_ID,
A.EMPLOYEE_ID
FROM
(
    SELECT
    P.PROJECT_ID,
    P.EMPLOYEE_ID,
    E.EXPERIENCE_YEARS,
    DENSE_RANK() OVER(PARTITION BY P.PROJECT_ID ORDER BY E.EXPERIENCE_YEARS DESC )
AS CRANK
    FROM PROJECT AS P LEFT JOIN EMPLOYEE AS E
    ON P.EMPLOYEE_ID = E.EMPLOYEE_ID
) A
WHERE A.CRANK = 1

```



1082. Sales Analysis I ↗

Table: Product

Column Name	Type
product_id	int
product_name	varchar
unit_price	int

product_id is the primary key of this table.

Each row of this table indicates the name and the price of each product.

Table: Sales

Column Name	Type
seller_id	int
product_id	int
buyer_id	int
sale_date	date
quantity	int
price	int

This table has no primary key, it can have repeated rows.

product_id is a foreign key to the Product table.

Each row of this table contains some information about one sale.

Write an SQL query that reports the best **seller** by total sales price, If there is a tie, report them all.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Product table:

product_id	product_name	unit_price
1	S8	1000
2	G4	800
3	iPhone	1400

Sales table:

seller_id	product_id	buyer_id	sale_date	quantity	price
1	1	1	2019-01-21	2	2000
1	2	2	2019-02-17	1	800
2	2	3	2019-06-02	1	800
3	3	4	2019-05-13	2	2800

Output:

seller_id
1
3

Explanation: Both sellers with id 1 and 3 sold products with the most total price of

- My Solution Using Group BY + Left Join + Rank

```

SELECT
TEMP.SELLER_ID
FROM
(
  SELECT
SSELLER_ID,
SUM(S.PRICE) AS TOTAL,
DENSE_RANK() OVER(ORDER BY SUM(S.PRICE) DESC) AS CRANK
FROM PRODUCT AS P LEFT JOIN SALES AS S
ON P.PRODUCT_ID = S.PRODUCT_ID
WHERE SSELLER_ID IS NOT NULL
GROUP BY SSELLER_ID
) AS TEMP
WHERE TEMP.CRANK = 1

```

1083. Sales Analysis II



Table: Product

Column Name	Type
product_id	int
product_name	varchar
unit_price	int

product_id is the primary key of this table.

Each row of this table indicates the name and the price of each product.

Table: Sales

Column Name	Type
seller_id	int
product_id	int
buyer_id	int
sale_date	date
quantity	int
price	int

This table has no primary key, it can have repeated rows.

product_id is a foreign key to the Product table.

Each row of this table contains some information about one sale.

Write an SQL query that reports the **buyers** who have bought *S8* but not *iPhone*. Note that *S8* and *iPhone* are products present in the Product table.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Product table:

product_id	product_name	unit_price
1	S8	1000
2	G4	800
3	iPhone	1400

Sales table:

seller_id	product_id	buyer_id	sale_date	quantity	price
1	1	1	2019-01-21	2	2000
1	2	2	2019-02-17	1	800
2	1	3	2019-06-02	1	800
3	3	3	2019-05-13	2	2800

Output:

buyer_id
1

Explanation: The buyer with id 1 bought an S8 but did not buy an iPhone. The buyer w

- Using EXCEPT operator to solve the Problem

```

SELECT
S.BUYER_ID
FROM PRODUCT AS P LEFT JOIN SALES AS S
ON P.PRODUCT_ID = S.PRODUCT_ID
WHERE S.BUYER_ID IS NOT NULL AND P.PRODUCT_NAME IN ("S8")
GROUP BY S.BUYER_ID

EXCEPT

SELECT
S.BUYER_ID
FROM PRODUCT AS P LEFT JOIN SALES AS S
ON P.PRODUCT_ID = S.PRODUCT_ID
WHERE S.BUYER_ID IS NOT NULL AND P.PRODUCT_NAME IN ("iPhone")
GROUP BY S.BUYER_ID

```

1084. Sales Analysis III ↗

Table: Product

Column Name	Type
product_id	int
product_name	varchar
unit_price	int

product_id is the primary key of this table.

Each row of this table indicates the name and the price of each product.

Table: Sales

Column Name	Type
seller_id	int
product_id	int
buyer_id	int
sale_date	date
quantity	int
price	int

This table has no primary key, it can have repeated rows.

product_id is a foreign key to the Product table.

Each row of this table contains some information about one sale.

Write an SQL query that reports the **products** that were **only** sold in the first quarter of 2019 . That is, between 2019-01-01 and 2019-03-31 inclusive.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Product table:

product_id	product_name	unit_price
1	S8	1000
2	G4	800
3	iPhone	1400

Sales table:

seller_id	product_id	buyer_id	sale_date	quantity	price
1	1	1	2019-01-21	2	2000
1	2	2	2019-02-17	1	800
2	2	3	2019-06-02	1	800
3	3	4	2019-05-13	2	2800

Output:

product_id	product_name
1	S8

Explanation:

The product with id 1 was only sold in the spring of 2019.

The product with id 2 was sold in the spring of 2019 but was also sold after the spr

The product with id 3 was sold after spring 2019.

We return only product 1 as it is the product that was only sold in the spring of 20

• FAILED QUERY

```

SELECT
    S.PRODUCT_ID,
    P.PRODUCT_NAME
FROM SALES AS S JOIN PRODUCT AS P
    ON S.PRODUCT_ID = P.PRODUCT_ID
WHERE
    SALE_DATE BETWEEN '2019-01-01' AND '2019-03-31'
    AND S.PRODUCT_ID
    NOT IN (
        SELECT
            SS.PRODUCT_ID
        FROM SALES SS JOIN PRODUCT PP
            ON SS.PRODUCT_ID = PP.PRODUCT_ID
        WHERE
            SS.SALE_DATE > '2019-03-31'
    )

```

- Newly Learnt One By using Agg Functions in the having condition

```

SELECT
P.PRODUCT_ID,
P.PRODUCT_NAME
FROM PRODUCT AS P JOIN SALES AS S
ON P.PRODUCT_ID = S.PRODUCT_ID
GROUP BY
P.PRODUCT_ID, P.PRODUCT_NAME
HAVING MIN(LEASE_DATE) >= '2019-01-01' AND MAX(LEASE_DATE) <= '2019-03-31'

```

511. Game Play Analysis I ↗

Table: Activity

Column Name	Type
player_id	int
device_id	int
event_date	date
games_played	int

(player_id, event_date) is the primary key of this table.

This table shows the activity of players of some games.

Each row is a record of a player who logged in and played a number of games (possibly 0).

Write an SQL query to report the **first login date** for each player.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Activity table:

player_id	device_id	event_date	games_played
1	2	2016-03-01	5
1	2	2016-05-02	6
2	3	2017-06-25	1
3	1	2016-03-02	0
3	4	2018-07-03	5

Output:

player_id	first_login
1	2016-03-01
2	2017-06-25
3	2016-03-02

- Here i have used list aggregator to slice the final output..

```
SELECT
    PLAYER_ID,
    SUBSTRING(GROUP_CONCAT(EVENT_DATE ORDER BY EVENT_DATE),1,10) AS FIRST_LOGIN
FROM ACTIVITY
WHERE EVENT_DATE IS NOT NULL
GROUP BY PLAYER_ID
```

512. Game Play Analysis II ↗

Table: Activity

Column Name	Type
player_id	int
device_id	int
event_date	date
games_played	int

(player_id, event_date) is the primary key of this table.

This table shows the activity of players of some games.

Each row is a record of a player who logged in and played a number of games (possibly

Write an SQL query to report the **device** that is first logged in for each player.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Activity table:

player_id	device_id	event_date	games_played
1	2	2016-03-01	5
1	2	2016-05-02	6
2	3	2017-06-25	1
3	1	2016-03-02	0
3	4	2018-07-03	5

Output:

player_id	device_id
1	2
2	3
3	1

- Solution Using Rank + Group By

```
# Write your MySQL query statement below
SELECT
PLAYER_ID,
DEVICE_ID
FROM
(
SELECT
PLAYER_ID, DEVICE_ID, EVENT_DATE,
DENSE_RANK() OVER(PARTITION BY PLAYER_ID ORDER BY EVENT_DATE ASC) AS CRANK
FROM ACTIVITY
GROUP BY PLAYER_ID, EVENT_DATE
) AS OP
WHERE OP.CRANK = 1
```



534. Game Play Analysis III ↗

Table: Activity

Column Name	Type
player_id	int
device_id	int
event_date	date
games_played	int

(player_id, event_date) is the primary key of this table.

This table shows the activity of players of some games.

Each row is a record of a player who logged in and played a number of games (possibly 0).



Write an SQL query to report for each player and date, how many games played **so far** by the player. That is, the total number of games played by the player until that date. Check the example for clarity.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Activity table:

player_id	device_id	event_date	games_played
1	2	2016-03-01	5
1	2	2016-05-02	6
1	3	2017-06-25	1
3	1	2016-03-02	0
3	4	2018-07-03	5

Output:

player_id	event_date	games_played_so_far
1	2016-03-01	5
1	2016-05-02	11
1	2017-06-25	12
3	2016-03-02	0
3	2018-07-03	5

Explanation:For the player with id 1, $5 + 6 = 11$ games played by 2016-05-02, and $5 + 6 + 1 = 12$ For the player with id 3, $0 + 5 = 5$ games played by 2018-07-03.

Note that for each player we only care about the days when the player logged in.



- Wrong Method

```
# Write your MySQL query statement below
SELECT
    PLAYER_ID,
    EVENT_DATE,
    SUM(GAMES_PLAYED) OVER (PARTITION BY PLAYER_ID) GAMES_PLAYED_SO_FAR
FROM ACTIVITY
GROUP BY PLAYER_ID, EVENT_DATE
```

- Wrong Method

```
# Write your MySQL query statement below
SELECT
PLAYER_ID,
EVENT_DATE,
IFNULL(
    LAG(GAMES_PLAYED, 1) OVER (PARTITION BY PLAYER_ID) , 0
) + GAMES_PLAYED

AS GAMES_PLAYED_SO_FAR
FROM ACTIVITY
GROUP BY PLAYER_ID, EVENT_DATE
```

- Correct Approach

```
SELECT
PLAYER_ID,
EVENT_DATE,
SUM(GAMES_PLAYED) OVER (PARTITION BY PLAYER_ID ORDER BY EVENT_DATE)
AS GAMES_PLAYED_SO_FAR
FROM ACTIVITY
GROUP BY PLAYER_ID, EVENT_DATE
```

550. Game Play Analysis IV ↗

Table: Activity

Column Name	Type
player_id	int
device_id	int
event_date	date
games_played	int

(player_id, event_date) is the primary key of this table.

This table shows the activity of players of some games.

Each row is a record of a player who logged in and played a number of games (possibly 0).

Write an SQL query to report the **fraction** of players that logged in again on the day after the day they first logged in, **rounded to 2 decimal places**. In other words, you need to count the number of players that logged in for at least two consecutive days starting from their first login date, then divide that number by the total number of players.

The query result format is in the following example.

Example 1:**Input:**

Activity table:

player_id	device_id	event_date	games_played
1	2	2016-03-01	5
1	2	2016-03-02	6
2	3	2017-06-25	1
3	1	2016-03-02	0
3	4	2018-07-03	5

Output:

fraction
0.33

Explanation:

Only the player with id 1 logged back in after the first day he had logged in so the fraction is 1/3 = 0.33

- doesn't work for all the test cases.

```

SELECT
ROUND (SUM(A.PARTIAL) / (SELECT COUNT(DISTINCT PLAYER_ID) FROM ACTIVITY),2)
AS FRACTION
FROM
(
  SELECT
  COUNT(DISTINCT PLAYER_ID) AS PARTIAL
  FROM ACTIVITY
  GROUP BY PLAYER_ID
  HAVING ABS(MIN(EVENT_DATE) - MAX(EVENT_DATE)) = 1
) AS A
    
```

- Working Code which got completed after 4 days of barinstorming.

```

SELECT
ROUND (SUM(C.TOTAL_PLAYERS) / (SELECT COUNT(DISTINCT PLAYER_ID) FROM ACTIVITY),2)
AS FRACTION
FROM
(
    SELECT COUNT(DISTINCT PLAYER_ID) AS TOTAL_PLAYERS
    FROM
    (
        SELECT
        A.PLAYER_ID,
        A.EVENT_DATE,
        LAG(A.EVENT_DATE,1) OVER (PARTITION BY A.PLAYER_ID) AS DIFF
        FROM
        (
            SELECT
            PLAYER_ID,
            EVENT_DATE,
            DENSE_RANK()
            OVER(PARTITION BY PLAYER_ID ORDER BY EVENT_DATE) as CRANK
            FROM
            ACTIVITY
        ) A
        WHERE A.CRANK IN (1,2)
        ) B
        WHERE ( B.EVENT_DATE - B.DIFF = 1)
    ) C
)

```

1107. New Users Daily Count ↗

Table: Traffic

Column Name	Type
user_id	int
activity	enum
activity_date	date

There is no primary key for this table, it may have duplicate rows.

The activity column is an ENUM type of ('login', 'logout', 'jobs', 'groups', 'homepage')

Write an SQL query to reports for every date within at most 90 days from today, the number of users that logged in for the first time on that date. Assume today is 2019-06-30 .

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Traffic table:

user_id	activity	activity_date
1	login	2019-05-01
1	homepage	2019-05-01
1	logout	2019-05-01
2	login	2019-06-21
2	logout	2019-06-21
3	login	2019-01-01
3	jobs	2019-01-01
3	logout	2019-01-01
4	login	2019-06-21
4	groups	2019-06-21
4	logout	2019-06-21
5	login	2019-03-01
5	logout	2019-03-01
5	login	2019-06-21
5	logout	2019-06-21

Output:

login_date	user_count
2019-05-01	1
2019-06-21	2

Explanation:

Note that we only care about dates with non zero user count.

The user with id 5 first logged in on 2019-03-01 so he's not counted on 2019-06-21.



```

SELECT
ACTIVITY_DATE AS LOGIN_DATE,
COUNT(DISTINCT USER_ID) AS USER_COUNT
FROM
(
    SELECT *,
    DENSE_RANK() OVER(PARTITION BY USER_ID ORDER BY ACTIVITY_DATE ASC) AS CRANK
    FROM TRAFFIC
    WHERE ACTIVITY = 'login'
) AS A
WHERE A.CRANK = 1 AND ACTIVITY_DATE BETWEEN '2019-04-01' AND '2019-06-30'
GROUP BY ACTIVITY_DATE

```

1112. Highest Grade For Each Student ↗

Table: Enrollments

Column Name	Type
student_id	int
course_id	int
grade	int

(student_id, course_id) is the primary key of this table.

Write a SQL query to find the highest grade with its corresponding course for each student. In case of a tie, you should find the course with the smallest `course_id`.

Return the result table ordered by `student_id` in **ascending order**.

The query result format is in the following example.

Example 1:

Input:

Enrollments table:

student_id	course_id	grade
2	2	95
2	3	95
1	1	90
1	2	99
3	1	80
3	2	75
3	3	82

Output:

student_id	course_id	grade
1	2	99
2	2	95
3	3	82

- Working Solution Using RANK

```

SELECT
A.STUDENT_ID,
A.COURSE_ID,
A.GRADE
FROM
(
    SELECT
        STUDENT_ID,
        COURSE_ID,
        GRADE,
        DENSE_RANK()
    OVER(PARTITION BY STUDENT_ID ORDER BY GRADE DESC, COURSE_ID ASC)
    AS RANKING
    FROM ENROLLMENTS
) A
WHERE A.RANKING = 1
ORDER BY A.STUDENT_ID

```

1113. Reported Posts ↗



Table: Actions

Column Name	Type
user_id	int
post_id	int
action_date	date
action	enum
extra	varchar

There is no primary key for this table, it may have duplicate rows.

The action column is an ENUM type of ('view', 'like', 'reaction', 'comment', 'report')

The extra column has optional information about the action, such as a reason for the

Write an SQL query that reports the number of posts reported yesterday for each report reason. Assume today is 2019-07-05 .

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Actions table:

user_id	post_id	action_date	action	extra
1	1	2019-07-01	view	null
1	1	2019-07-01	like	null
1	1	2019-07-01	share	null
2	4	2019-07-04	view	null
2	4	2019-07-04	report	spam
3	4	2019-07-04	view	null
3	4	2019-07-04	report	spam
4	3	2019-07-02	view	null
4	3	2019-07-02	report	spam
5	2	2019-07-04	view	null
5	2	2019-07-04	report	racism
5	5	2019-07-04	view	null
5	5	2019-07-04	report	racism

Output:

report_reason	report_count
spam	1
racism	2

Explanation: Note that we only care about report reasons with non-zero number of rep

- Solution Using Group BY + Where Condition

```

SELECT
EXTRA AS REPORT_REASON,
COUNT(DISTINCT POST_ID) AS REPORT_COUNT
FROM ACTIONS
WHERE ACTION = 'report' AND ACTION_DATE = '2019-07-04'
GROUP BY EXTRA

```

1126. Active Businesses ↗



Table: Events

Column Name	Type
business_id	int
event_type	varchar
occurrences	int

(business_id, event_type) is the primary key of this table.

Each row in the table logs the info that an event of some type occurred at some busi

The **average activity** for a particular event_type is the average occurrences across all companies that have this event.

An **active business** is a business that has **more than one** event_type such that their occurrences is **strictly greater** than the average activity for that event.

Write an SQL query to find all **active businesses**.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Events table:

business_id	event_type	occurrences
1	reviews	7
3	reviews	3
1	ads	11
2	ads	7
3	ads	6
1	page views	3
2	page views	12

Output:

business_id
1

Explanation:

The average activity for each event can be calculated as follows:

- 'reviews': $(7+3)/2 = 5$
- 'ads': $(11+7+6)/3 = 8$
- 'page views': $(3+12)/2 = 7.5$

The business with id=1 has 7 'reviews' events (more than 5) and 11 'ads' events (more than 8).

- Solution Using JOIN + GROUP BY + HAVING

```

SELECT
E.BUSINESS_ID
FROM EVENTS AS E JOIN
(
    SELECT EVENT_TYPE, ROUND(AVG(OCCURENCES),2) AS AVG_OCCR
    FROM EVENTS
    GROUP BY EVENT_TYPE
) AS T
ON E.EVENT_TYPE = T.EVENT_TYPE
WHERE E.OCCURENCES > T.AVG_OCCR
GROUP BY E.BUSINESS_ID
HAVING COUNT(E.BUSINESS_ID) >= 2

```

1132. Reported Posts II

Table: Actions

Column Name	Type
user_id	int
post_id	int
action_date	date
action	enum
extra	varchar

There is no primary key for this table, it may have duplicate rows.

The action column is an ENUM type of ('view', 'like', 'reaction', 'comment', 'report')

The extra column has optional information about the action, such as a reason for the

Table: Removals

Column Name	Type
post_id	int
remove_date	date

post_id is the primary key of this table.

Each row in this table indicates that some post was removed due to being reported or

Write an SQL query to find the average daily percentage of posts that got removed after being reported as spam, **rounded to 2 decimal places**.

The query result format is in the following example.

Example 1:

Input:

Actions table:

user_id	post_id	action_date	action	extra
1	1	2019-07-01	view	null
1	1	2019-07-01	like	null
1	1	2019-07-01	share	null
2	2	2019-07-04	view	null
2	2	2019-07-04	report	spam
3	4	2019-07-04	view	null
3	4	2019-07-04	report	spam
4	3	2019-07-02	view	null
4	3	2019-07-02	report	spam
5	2	2019-07-03	view	null
5	2	2019-07-03	report	racism
5	5	2019-07-03	view	null
5	5	2019-07-03	report	racism

Removals table:

post_id	remove_date
2	2019-07-20
3	2019-07-18

Output:

average_daily_percent
75.00

Explanation:

The percentage for 2019-07-04 is 50% because only one post of two spam reported post

The percentage for 2019-07-02 is 100% because one post was reported as spam and it w

The other days had no spam reports so the average is $(50 + 100) / 2 = 75\%$

Note that the output is only one number and that we do not care about the remove dat



- Inner Join Solution Fails At 3 test case...

```
SELECT
ROUND(SUM(C.PERCENTAGE) / COUNT(*),2) AS AVERAGE_DAILY_PERCENT
FROM
(
    SELECT
        A.ACTION_DATE,
        (A.NO_OF_REMOVED_POSTS / B.TOTAL_SPAM_POSTS) * 100 AS PERCENTAGE
    FROM
        (SELECT
            ACTION_DATE,
            COUNT(DISTINCT POST_ID) AS NO_OF_REMOVED_POSTS
        FROM ACTIONS
        WHERE POST_ID IN (SELECT POST_ID FROM REMOVALS) AND EXTRA = 'spam'
        GROUP BY ACTION_DATE) A
    JOIN
        (SELECT
            ACTION_DATE,
            COUNT(DISTINCT POST_ID) AS TOTAL_SPAM_POSTS
        FROM ACTIONS
        WHERE EXTRA = 'spam'
        GROUP BY ACTION_DATE) B
    ON A.ACTION_DATE = B.ACTION_DATE
) AS C
```

- Right Join Test Case Which Absolutely Runs Faster

```

SELECT
ROUND(SUM(C.PERCENTAGE) / COUNT(*),2) AS AVERAGE_DAILY_PERCENT
FROM
(
    SELECT
        A.ACTION_DATE,
        (A.NO_OF_REMOVED_POSTS / B.TOTAL_SPAM_POSTS) * 100 AS PERCENTAGE
    FROM
        (SELECT
            ACTION_DATE,
            COUNT(DISTINCT POST_ID) AS NO_OF_REMOVED_POSTS
        FROM ACTIONS
        WHERE POST_ID IN (SELECT POST_ID FROM REMOVALS) AND EXTRA = 'spam'
        GROUP BY ACTION_DATE) A

    RIGHT JOIN

        (SELECT
            ACTION_DATE,
            COUNT(DISTINCT POST_ID) AS TOTAL_SPAM_POSTS
        FROM ACTIONS
        WHERE EXTRA = 'spam'
        GROUP BY ACTION_DATE) B

    ON A.ACTION_DATE = B.ACTION_DATE
) AS C

```

1141. User Activity for the Past 30 Days | ↗ ▾

Table: Activity

Column Name	Type
user_id	int
session_id	int
activity_date	date
activity_type	enum

There is no primary key for this table, it may have duplicate rows.

The activity_type column is an ENUM of type ('open_session', 'end_session', 'scroll')

The table shows the user activities for a social media website.

Note that each session belongs to exactly one user.

Write an SQL query to find the daily active user count for a period of 30 days ending 2019-07-27 inclusively. A user was active on someday if they made at least one activity on that day.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Activity table:

user_id	session_id	activity_date	activity_type
1	1	2019-07-20	open_session
1	1	2019-07-20	scroll_down
1	1	2019-07-20	end_session
2	4	2019-07-20	open_session
2	4	2019-07-21	send_message
2	4	2019-07-21	end_session
3	2	2019-07-21	open_session
3	2	2019-07-21	send_message
3	2	2019-07-21	end_session
4	3	2019-06-25	open_session
4	3	2019-06-25	end_session

Output:

day	active_users
2019-07-20	2
2019-07-21	2

Explanation: Note that we do not care about days with zero active users.

- Mysql Server Solution Using Clean Between Statements.

```
SELECT
ACTIVITY_DATE as day,
COUNT(DISTINCT USER_ID) AS active_users
FROM ACTIVITY
WHERE DATE(ACTIVITY_DATE) BETWEEN '2019-06-28' AND '2019-07-27'
GROUP BY ACTIVITY_DATE
```

1142. User Activity for the Past 30 Days II ↗



Table: Activity

Column Name	Type
user_id	int
session_id	int
activity_date	date
activity_type	enum

There is no primary key for this table, it may have duplicate rows.

The activity_type column is an ENUM of type ('open_session', 'end_session', 'scroll_

The table shows the user activities for a social media website.

Note that each session belongs to exactly one user.



Write an SQL query to find the average number of sessions per user for a period of 30 days ending 2019-07-27 inclusively, **rounded to 2 decimal places**. The sessions we want to count for a user are those with at least one activity in that time period.

The query result format is in the following example.

Example 1:

Input:

Activity table:

user_id	session_id	activity_date	activity_type
1	1	2019-07-20	open_session
1	1	2019-07-20	scroll_down
1	1	2019-07-20	end_session
2	4	2019-07-20	open_session
2	4	2019-07-21	send_message
2	4	2019-07-21	end_session
3	2	2019-07-21	open_session
3	2	2019-07-21	send_message
3	2	2019-07-21	end_session
3	5	2019-07-21	open_session
3	5	2019-07-21	scroll_down
3	5	2019-07-21	end_session
4	3	2019-06-25	open_session
4	3	2019-06-25	end_session

Output:

average_sessions_per_user
1.33

Explanation: User 1 and 2 each had 1 session in the past 30 days while user 3 had 2

- i deeply had a misconception of activity_type to be some kinda scroll, send_msg
- i was wrong at that point.

```
SELECT ROUND(IFNULL(AVG(SESSIONS), 0.00),2) AS AVERAGE_SESSIONS_PER_USER
FROM
(
  SELECT
    USER_ID,
    COUNT(DISTINCT SESSION_ID) AS SESSIONS
  FROM ACTIVITY
  WHERE ACTIVITY_DATE BETWEEN '2019-06-28' AND '2019-07-27'
  GROUP BY USER_ID
)
AS T1
```

1148. Article Views I ↗

Table: Views

Column Name	Type
article_id	int
author_id	int
viewer_id	int
view_date	date

There is no primary key for this table, it may have duplicate rows.

Each row of this table indicates that some viewer viewed an article (written by some Note that equal author_id and viewer_id indicate the same person.



Write an SQL query to find all the authors that viewed at least one of their own articles.

Return the result table sorted by `id` in ascending order.

The query result format is in the following example.

Example 1:**Input:**

Views table:

article_id	author_id	viewer_id	view_date
1	3	5	2019-08-01
1	3	6	2019-08-02
2	7	7	2019-08-01
2	7	6	2019-08-02
4	7	1	2019-07-22
3	4	4	2019-07-21
3	4	4	2019-07-21

Output:

id
4
7

- lacking basics will cause huge problem while solving such a kind of problems
- i was trying, left join and all other things which must be used only

```
SELECT DISTINCT AUTHOR_ID AS ID  
FROM VIEWS  
WHERE AUTHOR_ID = VIEWER_ID
```

1149. Article Views II ↗

Table: Views

Column Name	Type
article_id	int
author_id	int
viewer_id	int
view_date	date

There is no primary key for this table, it may have duplicate rows.

Each row of this table indicates that some viewer viewed an article (written by some Note that equal author_id and viewer_id indicate the same person.

Write an SQL query to find all the people who viewed more than one article on the same date.

Return the result table sorted by `id` in ascending order.

The query result format is in the following example.

Example 1:

Input:

Views table:

article_id	author_id	viewer_id	view_date
1	3	5	2019-08-01
3	4	5	2019-08-01
1	3	6	2019-08-02
2	7	7	2019-08-01
2	7	6	2019-08-02
4	7	1	2019-07-22
3	4	4	2019-07-21
3	4	4	2019-07-21

Output:

id
5
6

```

SELECT
DISTINCT
VIEWER_ID AS ID
FROM VIEWS
GROUP BY VIEW_DATE, VIEWER_ID
HAVING COUNT(DISTINCT ARTICLE_ID) >= 2
ORDER BY ID

```

1158. Market Analysis I



Table: Users

Column Name	Type
user_id	int
join_date	date
favorite_brand	varchar

user_id is the primary key of this table.

This table has the info of the users of an online shopping website where users can s

Table: Orders

Column Name	Type
order_id	int
order_date	date
item_id	int
buyer_id	int
seller_id	int

order_id is the primary key of this table.

item_id is a foreign key to the Items table.

buyer_id and seller_id are foreign keys to the Users table.

Table: Items

Column Name	Type
item_id	int
item_brand	varchar

item_id is the primary key of this table.

Write an SQL query to find for each user, the join date and the number of orders they made as a buyer in 2019 .

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Users table:

user_id	join_date	favorite_brand
1	2018-01-01	Lenovo
2	2018-02-09	Samsung
3	2018-01-19	LG
4	2018-05-21	HP

Orders table:

order_id	order_date	item_id	buyer_id	seller_id
1	2019-08-01	4	1	2
2	2018-08-02	2	1	3
3	2019-08-03	3	2	3
4	2018-08-04	1	4	2
5	2018-08-04	1	3	4
6	2019-08-05	2	2	4

Items table:

item_id	item_brand
1	Samsung
2	Lenovo
3	LG
4	HP

Output:

buyer_id	join_date	orders_in_2019
1	2018-01-01	1
2	2018-02-09	2
3	2018-01-19	0
4	2018-05-21	0

```

SELECT
O.BUYER_ID,
U.JOIN_DATE,
CASE
    WHEN COUNT(*) IS NOT NULL THEN COUNT(*)
    ELSE 0
END
AS ORDERS_IN_2019
FROM USERS AS U LEFT JOIN ORDERS AS O
ON U.USER_ID = O.BUYER_ID JOIN ITEMS AS I
ON O.ITEM_ID = I.ITEM_ID
WHERE YEAR(O.ORDER_DATE) = 2019
GROUP BY O.BUYER_ID, U.JOIN_DATE

```

```

# 90 % LOGIC DONE ONLY I NEED TO CRACK THE COUNT BASED ON THE YEAR
SELECT
O.BUYER_ID,
U.JOIN_DATE,
/*
CASE
    WHEN YEAR(O.ORDER_DATE) = 2019 THEN COUNT(*)
    ELSE 0
END AS ORDERS_IN_2019
*/
FROM USERS AS U LEFT JOIN ORDERS AS O
ON U.USER_ID = O.BUYER_ID
GROUP BY O.BUYER_ID, U.JOIN_DATE

```

- Final Solution After Multiple Failures Made Mistake by keeping O.Buyer_id Which is wrong
- Always need to keep the id which is, primary key.

```

SELECT
U.USER_ID AS BUYER_ID,
U.JOIN_DATE,
SUM
(
CASE
    WHEN YEAR(O.ORDER_DATE) = 2019
    THEN 1 ELSE 0
END
) AS ORDERS_IN_2019

FROM USERS AS U LEFT JOIN ORDERS AS O
ON U.USER_ID = O.BUYER_ID
GROUP BY U.USER_ID, U.JOIN_DATE

```

1173. Immediate Food Delivery I ↗

Table: Delivery

Column Name	Type
delivery_id	int
customer_id	int
order_date	date
customer_pref_delivery_date	date

delivery_id is the primary key of this table.

The table holds information about food delivery to customers that make orders at som

If the customer's preferred delivery date is the same as the order date, then the order is called **immediate**; otherwise, it is called **scheduled**.

Write an SQL query to find the percentage of immediate orders in the table, **rounded to 2 decimal places**.

The query result format is in the following example.

Example 1:**Input:**

Delivery table:

delivery_id	customer_id	order_date	customer_pref_delivery_date
1	1	2019-08-01	2019-08-02
2	5	2019-08-02	2019-08-02
3	1	2019-08-11	2019-08-11
4	3	2019-08-24	2019-08-26
5	4	2019-08-21	2019-08-22
6	2	2019-08-11	2019-08-13

Output:

immediate_percentage
33.33

Explanation: The orders with delivery id 2 and 3 are immediate while the others are

- Using Count To Solve the Problem

```

SELECT
ROUND(
    COUNT(*)/(SELECT COUNT(*) FROM DELIVERY)*100,
    2)
AS IMMEDIATE_PERCENTAGE
FROM DELIVERY
WHERE ORDER_DATE = CUSTOMER_PREF_DELIVERY_DATE

```

1174. Immediate Food Delivery II ↗

Table: Delivery

Column Name	Type
delivery_id	int
customer_id	int
order_date	date
customer_pref_delivery_date	date

delivery_id is the primary key of this table.

The table holds information about food delivery to customers that make orders at som

If the customer's preferred delivery date is the same as the order date, then the order is called **immediate**; otherwise, it is called **scheduled**.

The **first order** of a customer is the order with the earliest order date that the customer made. It is guaranteed that a customer has precisely one first order.

Write an SQL query to find the percentage of immediate orders in the first orders of all customers, **rounded to 2 decimal places**.

The query result format is in the following example.

Example 1:

Input:

Delivery table:

delivery_id	customer_id	order_date	customer_pref_delivery_date
1	1	2019-08-01	2019-08-02
2	2	2019-08-02	2019-08-02
3	1	2019-08-11	2019-08-12
4	3	2019-08-24	2019-08-24
5	3	2019-08-21	2019-08-22
6	2	2019-08-11	2019-08-13
7	4	2019-08-09	2019-08-09

Output:

immediate_percentage
50.00

Explanation:

The customer id 1 has a first order with delivery id 1 and it is scheduled.
The customer id 2 has a first order with delivery id 2 and it is immediate.
The customer id 3 has a first order with delivery id 5 and it is scheduled.
The customer id 4 has a first order with delivery id 7 and it is immediate.
Hence, half the customers have immediate first orders.

```

WITH FIRST_ORDER_COUNTS AS
(
    SELECT * FROM
    (
        SELECT *,  

        DENSE_RANK()  

        OVER(PARTITION BY CUSTOMER_ID ORDER BY ORDER_DATE ASC) AS CRANK
    FROM DELIVERY
    ) AS A
    WHERE A.CRANK = 1
)

SELECT
ROUND
(
    100*(COUNT(*)/ (SELECT COUNT(*) FROM FIRST_ORDER_COUNTS)),
    2
) AS IMMEDIATE_PERCENTAGE
FROM
(
    SELECT *
    FROM
    (
        SELECT *,  

        DENSE_RANK()  

        OVER(PARTITION BY CUSTOMER_ID ORDER BY ORDER_DATE ASC) AS CRANK
    FROM DELIVERY
    ) AS A
    WHERE A.CRANK = 1
    ) AS B
WHERE B.ORDER_DATE = B.CUSTOMER_PREF_DELIVERY_DATE

```

1179. Reformat Department Table ↗



Table: Department

Column Name	Type
id	int
revenue	int
month	varchar

(id, month) is the primary key of this table.

The table has information about the revenue of each department per month.

The month has values in ["Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct"]

Write an SQL query to reformat the table such that there is a department id column and a revenue column **for each month**.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Department table:

id	revenue	month
1	8000	Jan
2	9000	Jan
3	10000	Feb
1	7000	Feb
1	6000	Mar

Output:

id	Jan_Revenue	Feb_Revenue	Mar_Revenue	...	Dec_Revenue
1	8000	7000	6000	...	null
2	9000	null	null	...	null
3	null	10000	null	...	null

Explanation: The revenue from Apr to Dec is null.

Note that the result table has 13 columns (1 for the department id + 12 for the mont

```
SELECT
ID,
MAX(CASE
    WHEN MONTH = 'Jan' THEN REVENUE
    ELSE NULL
END )AS JAN_REVENUE,

MAX(CASE
    WHEN MONTH = 'Feb' THEN REVENUE
    ELSE NULL
END )AS FEB_REVENUE,
MAX(
CASE
    WHEN MONTH = 'Mar' THEN REVENUE
    ELSE NULL
END )AS MAR_REVENUE,
MAX(
CASE
    WHEN MONTH = 'Apr' THEN REVENUE
    ELSE NULL
END )AS APR_REVENUE,
MAX(
CASE
    WHEN MONTH = 'May' THEN REVENUE
    ELSE NULL
END ) AS MAY_REVENUE,
MAX(
CASE
    WHEN MONTH = 'Jun' THEN REVENUE
    ELSE NULL
END )AS JUN_REVENUE,
MAX(
CASE
    WHEN MONTH = 'Jul' THEN REVENUE
    ELSE NULL
END ) AS JUL_REVENUE,
MAX(
CASE
    WHEN MONTH = 'Aug' THEN REVENUE
    ELSE NULL
END ) AS AUG_REVENUE,
MAX(
CASE
    WHEN MONTH = 'Sep' THEN REVENUE
    ELSE NULL
END) AS SEP_REVENUE,
MAX(
CASE
    WHEN MONTH = 'Oct' THEN REVENUE
    ELSE NULL
END) AS OCT_REVENUE,
```

```

MAX(
CASE
    WHEN MONTH = 'Nov' THEN REVENUE
    ELSE NULL
END ) AS NOV_REVENUE,
MAX(
CASE
    WHEN MONTH = 'Dec' THEN REVENUE
    ELSE NULL
END)AS DEC_REVENUE
FROM DEPARTMENT
GROUP BY ID

```

1193. Monthly Transactions I ↗

Table: Transactions

Column Name	Type
<code>id</code>	<code>int</code>
<code>country</code>	<code>varchar</code>
<code>state</code>	<code>enum</code>
<code>amount</code>	<code>int</code>
<code>trans_date</code>	<code>date</code>

`id` is the primary key of this table.

The table has information about incoming transactions.

The `state` column is an enum of type ["approved", "declined"].

Write an SQL query to find for each month and country, the number of transactions and their total amount, the number of approved transactions and their total amount.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Transactions table:

id	country	state	amount	trans_date
121	US	approved	1000	2018-12-18
122	US	declined	2000	2018-12-19
123	US	approved	2000	2019-01-01
124	DE	approved	2000	2019-01-07

Output:

month	country	trans_count	approved_count	trans_total_amount	approved_
2018-12	US	2	1	3000	1000
2019-01	US	1	1	2000	2000
2019-01	DE	1	1	2000	2000

```

SELECT
DATE_FORMAT(TRANS_DATE, '%Y-%m') AS MONTH,
COUNTRY,
COUNT(*) AS TRANS_COUNT,
SUM(CASE WHEN STATE = 'approved' THEN 1 ELSE 0 END) AS APPROVED_COUNT,
SUM(AMOUNT) AS TRANS_TOTAL_AMOUNT,
SUM(CASE WHEN STATE = 'approved' THEN AMOUNT ELSE 0 END) AS APPROVED_TOTAL_AMOUNT
FROM TRANSACTIONS
GROUP BY DATE_FORMAT(TRANS_DATE, '%Y-%m'), COUNTRY

```

1204. Last Person to Fit in the Bus ↗

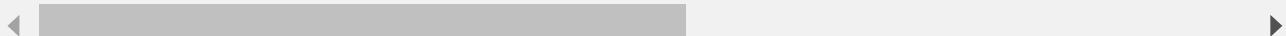
Table: Queue

Column Name	Type
person_id	int
person_name	varchar
weight	int
turn	int

person_id is the primary key column for this table.

This table has the information about all people waiting for a bus.

The person_id and turn columns will contain all numbers from 1 to n, where n is the turn determines the order of which the people will board the bus, where turn=1 denotes weight is the weight of the person in kilograms.



There is a queue of people waiting to board a bus. However, the bus has a weight limit of **1000 kilograms**, so there may be some people who cannot board.

Write an SQL query to find the person_name of the **last person** that can fit on the bus without exceeding the weight limit. The test cases are generated such that the first person does not exceed the weight limit.

The query result format is in the following example.

Example 1:

Input:

Queue table:

person_id	person_name	weight	turn	
5	Alice	250	1	
4	Bob	175	5	
3	Alex	350	2	
6	John Cena	400	3	
1	Winston	500	6	
2	Marie	200	4	

Output:

person_name
John Cena

Explanation: The following table is ordered by the turn for simplicity.

Turn	ID	Name	Weight	Total Weight	
1	5	Alice	250	250	
2	3	Alex	350	600	
3	6	John Cena	400	1000	(last person to board)
4	2	Marie	200	1200	(cannot board)
5	4	Bob	175	__	
6	1	Winston	500	__	

```

SELECT A.PERSON_NAME
FROM
(
  SELECT
    PERSON_NAME,
    SUM(WEIGHT) OVER(ORDER BY TURN ASC) AS TOTAL_WEIGHT
    FROM  QUEUE
) AS A
WHERE A.TOTAL_WEIGHT <= 1000
ORDER BY A.TOTAL_WEIGHT DESC LIMIT 1

```

1205. Monthly Transactions II ↗

Table: Transactions

Column Name	Type
id	int
country	varchar
state	enum
amount	int
trans_date	date

id is the primary key of this table.

The table has information about incoming transactions.

The state column is an enum of type ["approved", "declined"].

Table: Chargebacks

Column Name	Type
trans_id	int
trans_date	date

Chargebacks contains basic information regarding incoming chargebacks from some transactions.

trans_id is a foreign key to the id column of Transactions table.

Each chargeback corresponds to a transaction made previously even if they were not a

Write an SQL query to find for each month and country: the number of approved transactions and their total amount, the number of chargebacks, and their total amount.

Note: In your query, given the month and country, ignore rows with all zeros.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Transactions table:

id	country	state	amount	trans_date
101	US	approved	1000	2019-05-18
102	US	declined	2000	2019-05-19
103	US	approved	3000	2019-06-10
104	US	declined	4000	2019-06-13
105	US	approved	5000	2019-06-15

Chargebacks table:

trans_id	trans_date
102	2019-05-29
101	2019-06-30
105	2019-09-18

Output:

month	country	approved_count	approved_amount	chargeback_count	chargeba
2019-05	US	1	1000	1	2000
2019-06	US	2	8000	1	1000
2019-09	US	0	0	1	5000



```

SELECT
IFNULL(
    DATE_FORMAT(T.TRANS_DATE, '%Y-%m'),
    DATE_FORMAT(CB.TRANS_DATE, '%Y-%m')
),
T.COUNTRY
FROM TRANSACTIONS AS T LEFT JOIN CHARGEBACKS AS CB
ON T.ID = CB.TRANS_ID
GROUP BY
IFNULL(DATE_FORMAT(T.TRANS_DATE, '%Y-%m'),DATE_FORMAT(CB.TRANS_DATE, '%Y-%m')), T.CO
UNTRY

```

```

SELECT
DATE_FORMAT(T.TRANS_DATE, '%Y-%m') AS MONTH,
T.COUNTRY,
SUM(CASE WHEN STATE = 'approved' THEN 1 ELSE 0 END)
AS APPROVED_COUNT,
SUM(CASE WHEN STATE = 'approved' THEN AMOUNT ELSE 0 END)
AS APPROVED_AMOUNT,
SUM(CASE WHEN CB.TRANS_ID IS NOT NULL
THEN 1 ELSE 0 END)
AS CHARGEBACK_COUNT,
SUM(CASE WHEN CB.TRANS_ID IS NOT NULL THEN T.AMOUNT ELSE 0 END)
AS CHARGEBACK_AMOUNT
FROM TRANSACTIONS AS T LEFT JOIN CHARGEBACKS AS CB
ON T.ID = CB.TRANS_ID
GROUP BY
DATE_FORMAT(T.TRANS_DATE, '%Y-%m'), T.COUNTRY

```

- 3rd alternative

```

SELECT
DATE_FORMAT(T.TRANS_DATE, '%Y-%m') AS MONTH,
T.COUNTRY,
SUM(
CASE WHEN T.STATE = 'approved' THEN 1
ELSE 0
END
)
AS APPROVED_COUNT,

SUM(
CASE
WHEN T.STATE = 'approved' THEN AMOUNT
ELSE 0
END
)
AS APPROVED_AMOUNT,
SUM(
CASE
WHEN
DATE_FORMAT(CB.TRANS_DATE, '%Y-%m') = DATE_FORMAT(T.TRANS_DATE, '%Y-%m')      THE
N 1
ELSE 0
END
) AS CHARGEBACK_COUNT
FROM TRANSACTIONS AS T LEFT JOIN CHARGEBACKS AS CB
ON T.ID = CB.TRANS_ID
GROUP BY DATE_FORMAT(T.TRANS_DATE, '%Y-%m'), T.COUNTRY

```

- Finalized Solution

```
WITH CTE AS (
    SELECT *
    FROM TRANSACTIONS

    UNION

    SELECT
        CB.TRANS_ID,
        T.COUNTRY,
        'Chargeback' AS STATE,
        T.AMOUNT,
        CB.TRANS_DATE
    FROM TRANSACTIONS AS T LEFT JOIN CHARGEBACKS AS CB
    ON T.ID = CB.TRANS_ID
    WHERE CB.TRANS_ID IS NOT NULL
),
CTE_2 AS
(
    SELECT
        DATE_FORMAT(TRANS_DATE, '%Y-%m') AS MONTH,
        COUNTRY,
        SUM(
            CASE WHEN STATE = 'approved' THEN 1
            ELSE 0
            END
        )
        AS APPROVED_COUNT,
        SUM(
            CASE
                WHEN STATE = 'approved' THEN AMOUNT
                ELSE 0
                END
        )
        AS APPROVED_AMOUNT,
        SUM(
            CASE WHEN STATE = 'Chargeback' THEN 1
            ELSE 0
            END
        )
        AS CHARGEBACK_COUNT,
        SUM(
            CASE
                WHEN STATE = 'Chargeback' THEN AMOUNT
                ELSE 0
                END
        )
        AS CHARGEBACK_AMOUNT
    FROM CTE
```

```

        GROUP BY DATE_FORMAT(TRANS_DATE, '%Y-%m'), COUNTRY
    )

SELECT *
FROM CTE_2
WHERE (MONTH,COUNTRY) NOT IN
(   SELECT MONTH, COUNTRY
    FROM CTE_2
    WHERE APPROVED_COUNT = 0 AND APPROVED_AMOUNT = 0
        AND CHARGEBACK_COUNT = 0 AND CHARGEBACK_AMOUNT =0
)

```

1211. Queries Quality and Percentage ↗

Table: Queries

Column Name	Type
query_name	varchar
result	varchar
position	int
rating	int

There is no primary key for this table, it may have duplicate rows.

This table contains information collected from some queries on a database.

The position column has a value from 1 to 500.

The rating column has a value from 1 to 5. Query with rating less than 3 is a poor query.

We define query quality as:

The average of the ratio between query rating and its position.

We also define poor query percentage as:

The percentage of all queries with rating less than 3.

Write an SQL query to find each `query_name`, the `quality` and `poor_query_percentage`.

Both `quality` and `poor_query_percentage` should be **rounded to 2 decimal places**.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Queries table:

query_name	result	position	rating
Dog	Golden Retriever	1	5
Dog	German Shepherd	2	5
Dog	Mule	200	1
Cat	Shirazi	5	2
Cat	Siamese	3	3
Cat	Sphynx	7	4

Output:

query_name	quality	poor_query_percentage
Dog	2.50	33.33
Cat	0.66	33.33

Explanation:

Dog queries quality is $((5 / 1) + (5 / 2) + (1 / 200)) / 3 = 2.50$

Dog queries poor_query_percentage is $(1 / 3) * 100 = 33.33$

Cat queries quality equals $((2 / 5) + (3 / 3) + (4 / 7)) / 3 = 0.66$

Cat queries poor_query_percentage is $(1 / 3) * 100 = 33.33$

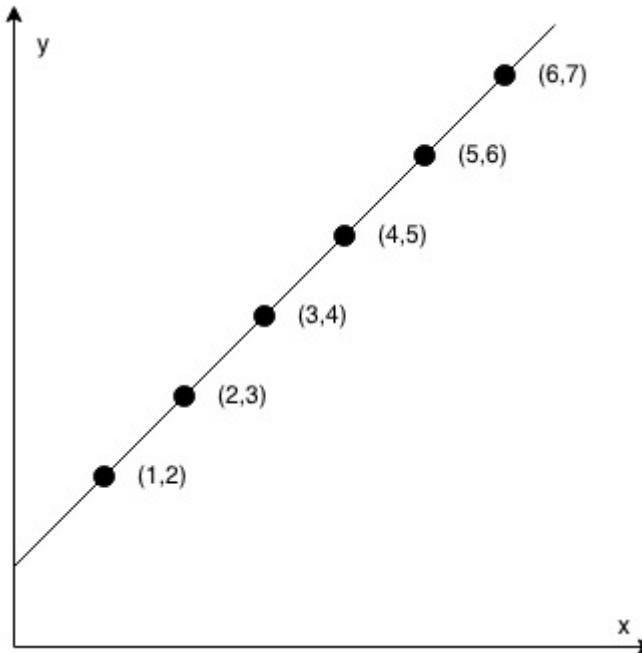
- SQL Solution Using CASE + Group BY

```
SELECT
QUERY_NAME,
ROUND(SUM(RATING/POSITION)/COUNT(*),2) AS QUALITY,
ROUND((
    SUM(CASE WHEN RATING < 3 THEN 1 ELSE 0 END) / COUNT(*)
) * 100, 2)
AS POOR_QUERY_PERCENTAGE
FROM QUERIES
GROUP BY QUERY_NAME
```

1232. Check If It Is a Straight Line ↗

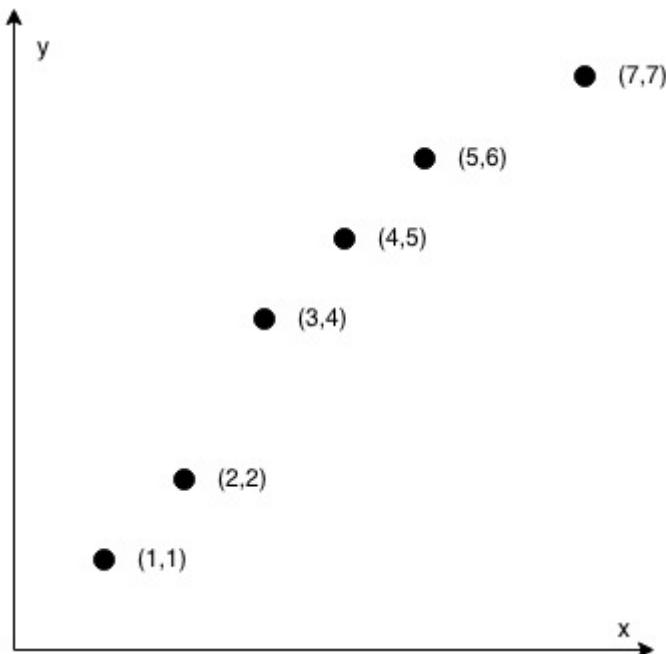


You are given an array `coordinates`, `coordinates[i] = [x, y]`, where `[x, y]` represents the coordinate of a point. Check if these points make a straight line in the XY plane.

Example 1:

Input: `coordinates = [[1,2],[2,3],[3,4],[4,5],[5,6],[6,7]]`

Output: `true`

Example 2:

Input: `coordinates = [[1,1],[2,2],[3,4],[4,5],[5,6],[7,7]]`

Output: `false`

Constraints:

- $2 \leq \text{coordinates.length} \leq 1000$
- $\text{coordinates}[i].length == 2$
- $-10^4 \leq \text{coordinates}[i][0], \text{coordinates}[i][1] \leq 10^4$
- coordinates contains no duplicate point.

```
class Solution:
    def checkStraightLine(self, coordinates: List[List[int]]) -> bool:
        slope = set()
        for i in range(1, len(coordinates)):
            if coordinates[i][0]-coordinates[i-1][0] != 0:
                res = (coordinates[i][1]-coordinates[i-1][1]) / (coordinates[i][0]-coordinates[i-1][0])
                slope.add(res)
            else:
                slope.add('inf')
        if len(slope):
            return True
        return False
```

1249. Minimum Remove to Make Valid Parentheses

Given a string s of ' $($ ' , ' $)$ ' and lowercase English characters.

Your task is to remove the minimum number of parentheses (' $($ ' or ' $)$ ' , in any positions) so that the resulting *parentheses string* is valid and return **any** valid string.

Formally, a *parentheses string* is valid if and only if:

- It is the empty string, contains only lowercase characters, or
- It can be written as AB (A concatenated with B), where A and B are valid strings, or
- It can be written as (A) , where A is a valid string.

Example 1:

Input: $s = \text{"lee(t(c)o)de"}$

Output: "lee(t(c)o)de"

Explanation: "lee(t(co)de)" , "lee(t(c)ode)" would also be accepted.

Example 2:

Input: s = "a)b(c)d"
Output: "ab(c)d"

Example 3:

Input: s = "))((")
Output: ""
Explanation: An empty string is also valid.

Constraints:

- $1 \leq s.length \leq 10^5$
- $s[i]$ is either '(', ')', or lowercase English letter .

```
class Solution:
    from string import digits
    def minRemoveToMakeValid(self, s: str) -> str:
        res = []
        # O(N) to Build the Stack by following the indexes.
        for i in range(len(s)):
            if ord(s[i]) >= 40 and ord(s[i]) <= 41:
                if s[i] == ')':
                    if len(res) > 0:
                        if res[-1][0] == '(':
                            res.pop()
                        else:
                            res.append((s[i],i))
                    else:
                        res.append((s[i],i))
                else:
                    res.append((s[i],i))
        # Converting the String to List and checking it where to remove the extra parathesis.
        s = list(s)
        for val in res:
            s[val[1]] = -1
        return ''.join([char for char in s if char != -1])
```

2217. Find Palindrome With Fixed Length ↗

Given an integer array `queries` and a **positive** integer `intLength`, return an array `answer` where `answer[i]` is either the `queries[i]th` smallest **positive palindrome** of length `intLength` or -1 if no such palindrome exists.

A **palindrome** is a number that reads the same backwards and forwards. Palindromes cannot have leading zeros.

Example 1:

Input: queries = [1,2,3,4,5,90], intLength = 3

Output: [101,111,121,131,141,999]

Explanation:

The first few palindromes of length 3 are:

101, 111, 121, 131, 141, 151, 161, 171, 181, 191, 202, ...

The 90th palindrome of length 3 is 999.

Example 2:

Input: queries = [2,4,6], intLength = 4

Output: [1111,1331,1551]

Explanation:

The first six palindromes of length 4 are:

1001, 1111, 1221, 1331, 1441, and 1551.

Constraints:

- $1 \leq \text{queries.length} \leq 5 * 10^4$
 - $1 \leq \text{queries}[i] \leq 10^9$
 - $1 \leq \text{intLength} \leq 15$
-
- Firstly I have made false assumptions about the problem statement, and haven't seen the upper bounds.
 - And has tried to implement a naive algorithm.

```

class Solution:
def kthPalindrome(self, queries: List[int], intLength: int) -> List[int]:
    upper = 10 ** intLength
    lower = 10 ** (intLength - 1)
    res = [0]
    orig_range = upper-lower

    # Estimating
    # the number of palindromes to be generated
    max_limit = -inf
    for val in queries:
        if val <= orig_range and val >= max_limit :
            max_limit = val

    for val in range(lower, upper):
        if int(str(val)[::-1]) == val and len(res) <= max_limit:
            res.append(val)

    fin = []
    length = len(res)
    print(len(res))
    for val in queries:
        if val > length:
            fin.append(-1)
        else:
            fin.append(res[val])
    return fin

```

1241. Number of Comments per Post ↗



Table: Submissions

Column Name	Type
sub_id	int
parent_id	int

There is no primary key for this table, it may have duplicate rows.

Each row can be a post or comment on the post.

parent_id is null for posts.

parent_id for comments is sub_id for another post in the table.

Write an SQL query to find the number of comments per post. The result table should contain `post_id` and its corresponding `number_of_comments`.

The `Submissions` table may contain duplicate comments. You should count the number of **unique comments** per post.

The `Submissions` table may contain duplicate posts. You should treat them as one post.

The result table should be **ordered** by `post_id` in **ascending order**.

The query result format is in the following example.

Example 1:

Input:

`Submissions` table:

sub_id	parent_id
1	Null
2	Null
1	Null
12	Null
3	1
5	2
3	1
4	1
9	1
10	2
6	7

Output:

post_id	number_of_comments
1	3
2	2
12	0

Explanation:

The post with id 1 has three comments in the table with id 3, 4, and 9. The comment

The post with id 2 has two comments in the table with id 5 and 10.

The post with id 12 has no comments in the table.

The comment with id 6 is a comment on a deleted post with id 7 so we ignored it.



- Using IN + JOIN + CASE + Group BY Works Perfectly Fine

```

SELECT
DISTINCT
T1.SUB_ID AS POST_ID,
CASE
    WHEN T2.COM_COUNT IS NULL THEN 0
    ELSE T2.COM_COUNT
END AS NUMBER_OF_COMMENTS
FROM
(SELECT SUB_ID FROM SUBMISSIONS WHERE PARENT_ID IS NULL) AS T1
LEFT JOIN
(
    SELECT
        PARENT_ID, COUNT(DISTINCT SUB_ID) AS COM_COUNT
    FROM SUBMISSIONS
    WHERE PARENT_ID IN (SELECT SUB_ID FROM SUBMISSIONS
                        WHERE PARENT_ID IS NULL)
    GROUP BY PARENT_ID
) AS T2
ON T1.SUB_ID = T2.PARENT_ID
ORDER BY POST_ID

```

1260. Shift 2D Grid ↗

Given a 2D grid of size $m \times n$ and an integer k . You need to shift the grid k times.

In one shift operation:

- Element at $\text{grid}[i][j]$ moves to $\text{grid}[i][j + 1]$.
- Element at $\text{grid}[i][n - 1]$ moves to $\text{grid}[i + 1][0]$.
- Element at $\text{grid}[m - 1][n - 1]$ moves to $\text{grid}[0][0]$.

Return the 2D grid after applying shift operation k times.

Example 1:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \rightarrow \begin{bmatrix} 9 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix}$$

Input: $\text{grid} = [[1,2,3],[4,5,6],[7,8,9]]$, $k = 1$

Output: $[[9,1,2],[3,4,5],[6,7,8]]$

Example 2:

$$\begin{bmatrix} 3 & 8 & 1 & 9 \\ 19 & 7 & 2 & 5 \\ 4 & 6 & 11 & 10 \\ 12 & 0 & 21 & 13 \end{bmatrix} \rightarrow \begin{bmatrix} 13 & 3 & 8 & 1 \\ 9 & 19 & 7 & 2 \\ 5 & 4 & 6 & 11 \\ 10 & 12 & 0 & 21 \end{bmatrix} \rightarrow \begin{bmatrix} 21 & 13 & 3 & 8 \\ 1 & 9 & 19 & 7 \\ 2 & 5 & 4 & 6 \\ 11 & 10 & 12 & 0 \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} 0 & 21 & 13 & 3 \\ 8 & 1 & 9 & 19 \\ 7 & 2 & 5 & 4 \\ 6 & 11 & 10 & 12 \end{bmatrix} \rightarrow \begin{bmatrix} 12 & 0 & 21 & 13 \\ 3 & 8 & 1 & 9 \\ 19 & 7 & 2 & 5 \\ 4 & 6 & 11 & 10 \end{bmatrix}$$

Input: grid = [[3,8,1,9],[19,7,2,5],[4,6,11,10],[12,0,21,13]], k = 4

Output: [[12,0,21,13],[3,8,1,9],[19,7,2,5],[4,6,11,10]]

Example 3:

Input: grid = [[1,2,3],[4,5,6],[7,8,9]], k = 9

Output: [[1,2,3],[4,5,6],[7,8,9]]

Constraints:

- m == grid.length
- n == grid[i].length
- 1 <= m <= 50
- 1 <= n <= 50
- -1000 <= grid[i][j] <= 1000
- 0 <= k <= 100

```
class Solution:
    def shiftGrid(self, grid: List[List[int]], k: int) -> List[List[int]]:
        m = length
        count = 1
        while count <= k:
            for i in range(m):
                if i == 0:
                    grid[i] = [grid[m-1][-1]] + grid[i]
                else:
                    grid[i] = [grid[i-1][-1]] + grid[i]
            for j in grid:
                del j[-1]

            count += 1

        return grid
```

1251. Average Selling Price ↗

Table: Prices

Column Name	Type
product_id	int
start_date	date
end_date	date
price	int

(product_id, start_date, end_date) is the primary key for this table.

Each row of this table indicates the price of the product_id in the period from start_date to end_date.

For each product_id there will be no two overlapping periods. That means there will

Table: UnitsSold

Column Name	Type
product_id	int
purchase_date	date
units	int

There is no primary key for this table, it may contain duplicates.

Each row of this table indicates the date, units, and product_id of each product sold.

Write an SQL query to find the average selling price for each product. `average_price` should be **rounded to 2 decimal places**.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Prices table:

product_id	start_date	end_date	price	
1	2019-02-17	2019-02-28	5	
1	2019-03-01	2019-03-22	20	
2	2019-02-01	2019-02-20	15	
2	2019-02-21	2019-03-31	30	

UnitsSold table:

product_id	purchase_date	units
1	2019-02-25	100
1	2019-03-01	15
2	2019-02-10	200
2	2019-03-22	30

Output:

product_id	average_price
1	6.96
2	16.96

Explanation:

Average selling price = Total Price of Product / Number of products sold.

Average selling price for product 1 = $((100 * 5) + (15 * 20)) / 115 = 6.96$ Average selling price for product 2 = $((200 * 15) + (30 * 30)) / 230 = 16.96$

- Using CTE + WHERE + Group BY Solves the Problem

```

WITH SUMMARY AS
(
    SELECT
        PR.PRODUCT_ID,
        PR.PRICE,
        U.UNITS AS UNITS
    FROM PRICES AS PR LEFT JOIN UNITSSOLD AS U
    ON PR.PRODUCT_ID = U.PRODUCT_ID
    WHERE U.PURCHASE_DATE BETWEEN PR.START_DATE AND PR.END_DATE
    GROUP BY PR.PRODUCT_ID, PR.START_DATE, PR.END_DATE, U.PURCHASE_DATE
)
SELECT
    PRODUCT_ID,
    ROUND(SUM(PRICE*UNITS)/ SUM(UNITS),2) AS AVERAGE_PRICE
FROM SUMMARY
GROUP BY PRODUCT_ID

```

1270. All People Report to the Given Manager ↗ ▾

Table: Employees

Column Name	Type
employee_id	int
employee_name	varchar
manager_id	int

employee_id is the primary key for this table.

Each row of this table indicates that the employee with ID employee_id and name employee_name reports directly to manager with ID manager_id. The head of the company is the employee with employee_id = 1.

Write an SQL query to find employee_id of all employees that directly or indirectly report their work to the head of the company.

The indirect relation between managers **will not exceed three managers** as the company is small.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Employees table:

employee_id	employee_name	manager_id
1	Boss	1
3	Alice	3
2	Bob	1
4	Daniel	2
7	Luis	4
8	Jhon	3
9	Angela	8
77	Robert	1

Output:

employee_id
2
77
4
7

Explanation:

The head of the company is the employee with employee_id 1.

The employees with employee_id 2 and 77 report their work directly to the head of the company.

The employee with employee_id 4 reports their work indirectly to the head of the company.

The employee with employee_id 7 reports their work indirectly to the head of the company.

The employees with employee_id 3, 8, and 9 do not report their work to the head of the company.



```

SELECT EMPLOYEE_ID
FROM EMPLOYEES
WHERE MANAGER_ID = 1 AND EMPLOYEE_ID != MANAGER_ID

UNION

SELECT EMPLOYEE_ID
FROM EMPLOYEES
WHERE MANAGER_ID IN (SELECT EMPLOYEE_ID FROM EMPLOYEES
                      WHERE MANAGER_ID = 1
                      AND EMPLOYEE_ID != MANAGER_ID)

UNION

(
    SELECT EMPLOYEE_ID
    FROM EMPLOYEES
    WHERE MANAGER_ID IN
    (
        SELECT EMPLOYEE_ID
        FROM EMPLOYEES
        WHERE MANAGER_ID IN (SELECT EMPLOYEE_ID FROM EMPLOYEES
                              WHERE MANAGER_ID = 1
                              AND EMPLOYEE_ID != MANAGER_ID)
    )
)

```

1281. Subtract the Product and Sum of Digits of an Integer ↗

Given an integer number n , return the difference between the product of its digits and the sum of its digits.

Example 1:

Input: $n = 234$
Output: 15
Explanation:
Product of digits = $2 * 3 * 4 = 24$
Sum of digits = $2 + 3 + 4 = 9$
Result = $24 - 9 = 15$

Example 2:

Input: n = 4421

Output: 21

Explanation:

Product of digits = $4 * 4 * 2 * 1 = 32$

Sum of digits = $4 + 4 + 2 + 1 = 11$

Result = $32 - 11 = 21$

Constraints:

- $1 \leq n \leq 10^5$

...

O(1) Space

O(N) Tc

```
class Solution:
    def calculate(self, n):
        prod, sum_ = 1, 0
        while n > 0:
            prod = prod * (n % 10)
            sum_ = sum_ + (n % 10)
            n = n // 10
        return prod - sum_
    def subtractProductAndSum(self, n: int) -> int:
        return self.calculate(n)
```

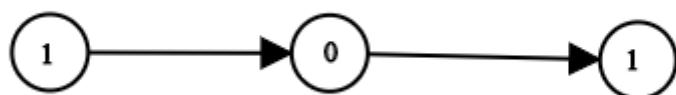
1290. Convert Binary Number in a Linked List to Integer ↴

Given `head` which is a reference node to a singly-linked list. The value of each node in the linked list is either `0` or `1`. The linked list holds the binary representation of a number.

Return the *decimal value* of the number in the linked list.

The **most significant bit** is at the head of the linked list.

Example 1:



Input: head = [1,0,1]

Output: 5

Explanation: (101) in base 2 = (5) in base 10

Example 2:

Input: head = [0]

Output: 0

Constraints:

- The Linked List is not empty.
- Number of nodes will not exceed 30 .
- Each node's value is either 0 or 1 .

```
# Definition for singly-linked list.
class Solution:
    def getDecimalValue(self, head: ListNode) -> int:
        root = head
        val = ""
        while root:
            val+= str(root.val)
            root = root.next
        return int(val,2)
```

1280. Students and Examinations ↗



Table: Students

Column Name	Type
student_id	int
student_name	varchar

student_id is the primary key for this table.

Each row of this table contains the ID and the name of one student in the school.

Table: Subjects

Column Name	Type
subject_name	varchar

subject_name is the primary key for this table.

Each row of this table contains the name of one subject in the school.

Table: Examinations

Column Name	Type
student_id	int
subject_name	varchar

There is no primary key for this table. It may contain duplicates.

Each student from the Students table takes every course from the Subjects table.

Each row of this table indicates that a student with ID student_id attended the exam

Write an SQL query to find the number of times each student attended each exam.

Return the result table ordered by `student_id` and `subject_name`.

The query result format is in the following example.

Example 1:

Input:

Students table:

student_id	student_name
1	Alice
2	Bob
13	John
6	Alex

Subjects table:

subject_name
Math
Physics
Programming

Examinations table:

student_id	subject_name
1	Math
1	Physics
1	Programming
2	Programming
1	Physics
1	Math
13	Math
13	Programming
13	Physics
2	Math
1	Math

Output:

student_id	student_name	subject_name	attended_exams	
1	Alice	Math	3	
1	Alice	Physics	2	
1	Alice	Programming	1	
2	Bob	Math	1	
2	Bob	Physics	0	
2	Bob	Programming	1	
6	Alex	Math	0	
6	Alex	Physics	0	
6	Alex	Programming	0	
13	John	Math	1	
13	John	Physics	1	
13	John	Programming	1	

Explanation:

The result table should contain all students and all subjects.

Alice attended the Math exam 3 times, the Physics exam 2 times, and the Programming exam 1 time. Bob attended the Math exam 1 time, the Programming exam 1 time, and did not attend the Physics exam. Alex did not attend any exams.

John attended the Math exam 1 time, the Physics exam 1 time, and the Programming exam 1 time.

```
SELECT
S.STUDENT_ID,
S.STUDENT_NAME,
SB.SUBJECT_NAME,
COUNT(*) AS ATTENDED_EXAMS
FROM STUDENTS AS S JOIN EXAMINATIONS AS E
ON S.STUDENT_ID = E.STUDENT_ID RIGHT JOIN SUBJECTS AS SB
ON E.SUBJECT_NAME = SB.SUBJECT_NAME
GROUP BY S.STUDENT_ID, S.STUDENT_NAME, SB.SUBJECT_NAME
ORDER BY S.STUDENT_ID
```

final Solution after trying for 25 days..

```
WITH TEMP_TABLE AS
(
SELECT S.STUDENT_ID, S.STUDENT_NAME, SB.SUBJECT_NAME
FROM STUDENTS AS S JOIN SUBJECTS AS SB
)

SELECT
T.STUDENT_ID,
T.STUDENT_NAME,
T.SUBJECT_NAME,
CASE
    WHEN COUNT(E.STUDENT_ID) IS NULL THEN 0
    ELSE COUNT(E.STUDENT_ID)
END AS ATTENDED_EXAMS
FROM TEMP_TABLE AS T LEFT JOIN EXAMINATIONS AS E
ON T.SUBJECT_NAME = E.SUBJECT_NAME AND T.STUDENT_ID = E.STUDENT_ID
GROUP BY T.STUDENT_ID, T.STUDENT_NAME, T.SUBJECT_NAME
ORDER BY T.STUDENT_ID, T.SUBJECT_NAME
```

1294. Weather Type in Each Country

Table: Countries

Column Name	Type
country_id	int
country_name	varchar

country_id is the primary key for this table.

Each row of this table contains the ID and the name of one country.

Table: Weather

Column Name	Type
country_id	int
weather_state	int
day	date

(country_id, day) is the primary key for this table.

Each row of this table indicates the weather state in a country for one day.

Write an SQL query to find the type of weather in each country for **November 2019**.

The type of weather is:

- **Cold** if the average weather_state is less than or equal 15 ,
- **Hot** if the average weather_state is greater than or equal to 25 , and
- **Warm** otherwise.

Return result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Countries table:

country_id	country_name
2	USA
3	Australia
7	Peru
5	China
8	Morocco
9	Spain

Weather table:

country_id	weather_state	day
2	15	2019-11-01
2	12	2019-10-28
2	12	2019-10-27
3	-2	2019-11-10
3	0	2019-11-11
3	3	2019-11-12
5	16	2019-11-07
5	18	2019-11-09
5	21	2019-11-23
7	25	2019-11-28
7	22	2019-12-01
7	20	2019-12-02
8	25	2019-11-05
8	27	2019-11-15
8	31	2019-11-25
9	7	2019-10-23
9	3	2019-12-23

Output:

country_name	weather_type
USA	Cold
Australia	Cold
Peru	Hot
Morocco	Hot
China	Warm

Explanation:

Average weather_state in USA in November is $(15) / 1 = 15$ so weather type is Cold.

Average weather_state in Australia in November is $(-2 + 0 + 3) / 3 = 0.333$ so weather type is Cold.

Average weather_state in Peru in November is $(25) / 1 = 25$ so the weather type is Hot.

Average weather_state in China in November is $(16 + 18 + 21) / 3 = 18.333$ so weather type is Warm.

Average weather_state in Morocco in November is $(25 + 27 + 31) / 3 = 27.667$ so weather type is Hot.

We know nothing about the average weather_state in Spain in November so we do not include it in the output.

- Using LEFT Join + GROUP BY solves the problem

```

SELECT
C.COUNTRY_NAME,
CASE
    WHEN AVG(WEATHER_STATE) <= 15 THEN 'Cold'
    WHEN AVG(WEATHER_STATE) >= 25 THEN 'Hot'
    ELSE 'Warm'
END AS WEATHER_TYPE
FROM COUNTRIES AS C LEFT JOIN WEATHER AS W
ON C.COUNTRY_ID = W.COUNTRY_ID
WHERE YEAR(DAY) = 2019 AND MONTH(DAY) = 11
GROUP BY C.COUNTRY_ID

```

1309. Decrypt String from Alphabet to Integer Mapping

You are given a string `s` formed by digits and '#' . We want to map `s` to English lowercase characters as follows:

- Characters ('a' to 'i') are represented by ('1' to '9') respectively.
- Characters ('j' to 'z') are represented by ('10#' to '26#') respectively.

Return *the string formed after mapping*.

The test cases are generated so that a unique mapping will always exist.

Example 1:

```

Input: s = "10#11#12"
Output: "jkab"
Explanation: "j" -> "10#" , "k" -> "11#" , "a" -> "1" , "b" -> "2".

```

Example 2:

```

Input: s = "1326#"
Output: "acz"

```

Constraints:

- `1 <= s.length <= 1000`
- `s` consists of digits and the '#' letter.

- `s` will be a valid string such that mapping is always possible.

```
class Solution:
    # O(N) Solution
    def freqAlphabets(self, s: str) -> str:
        res = []
        for ind in range(len(s)):
            if s[ind] == '#':
                val = res[-2] + res[-1]
                res.pop()
                res.pop()
                res.append(val)
            else:
                res.append(s[ind])
        output = ""
        for val in res:
            output+=chr(96 + int(val))
        return output
```

1308. Running Total for Different Genders ↗



Table: Scores

Column Name	Type
player_name	varchar
gender	varchar
day	date
score_points	int

(gender, day) is the primary key for this table.

A competition is held between the female team and the male team.

Each row of this table indicates that a player_name and with gender has scored score
Gender is 'F' if the player is in the female team and 'M' if the player is in the ma



Write an SQL query to find the total score for each gender on each day.

Return the result table ordered by gender and day in **ascending order**.

The query result format is in the following example.

Example 1:**Input:**

Scores table:

player_name	gender	day	score_points
Aron	F	2020-01-01	17
Alice	F	2020-01-07	23
Bajrang	M	2020-01-07	7
Khali	M	2019-12-25	11
Slaman	M	2019-12-30	13
Joe	M	2019-12-31	3
Jose	M	2019-12-18	2
Priya	F	2019-12-31	23
Priyanka	F	2019-12-30	17

Output:

gender	day	total
F	2019-12-30	17
F	2019-12-31	40
F	2020-01-01	57
F	2020-01-07	80
M	2019-12-18	2
M	2019-12-25	13
M	2019-12-30	26
M	2019-12-31	29
M	2020-01-07	36

Explanation:

For the female team:

The first day is 2019-12-30, Priyanka scored 17 points and the total score for the team is 17.

The second day is 2019-12-31, Priya scored 23 points and the total score for the team is 40.

The third day is 2020-01-01, Aron scored 17 points and the total score for the team is 57.

The fourth day is 2020-01-07, Alice scored 23 points and the total score for the team is 80.

For the male team:

The first day is 2019-12-18, Jose scored 2 points and the total score for the team is 2.

The second day is 2019-12-25, Khali scored 11 points and the total score for the team is 13.

The third day is 2019-12-30, Slaman scored 13 points and the total score for the team is 26.

The fourth day is 2019-12-31, Joe scored 3 points and the total score for the team is 29.

The fifth day is 2020-01-07, Bajrang scored 7 points and the total score for the team is 36.



```

SELECT
GENDER,
DAY,
SUM(SCORE_POINTS)
OVER(PARTITION BY GENDER ORDER BY DAY) AS TOTAL
FROM SCORES
GROUP BY GENDER, DAY
ORDER BY GENDER, DAY

```

1342. Number of Steps to Reduce a Number to Zero ↗

Given an integer `num`, return *the number of steps to reduce it to zero*.

In one step, if the current number is even, you have to divide it by 2, otherwise, you have to subtract 1 from it.

Example 1:

Input: num = 14
Output: 6
Explanation:
Step 1) 14 is even; divide by 2 and obtain 7.
Step 2) 7 is odd; subtract 1 and obtain 6.
Step 3) 6 is even; divide by 2 and obtain 3.
Step 4) 3 is odd; subtract 1 and obtain 2.
Step 5) 2 is even; divide by 2 and obtain 1.
Step 6) 1 is odd; subtract 1 and obtain 0.

Example 2:

Input: num = 8
Output: 4
Explanation:
Step 1) 8 is even; divide by 2 and obtain 4.
Step 2) 4 is even; divide by 2 and obtain 2.
Step 3) 2 is even; divide by 2 and obtain 1.
Step 4) 1 is odd; subtract 1 and obtain 0.

Example 3:

Input: num = 123
Output: 12

Constraints:

- $0 \leq num \leq 10^6$
- By followiong the above mentioned steps we can solve this problem.

```
class Solution:
    def numberOfSteps(self, num: int) -> int:
        count = 0
        while num > 0:
            if num == 0:
                break
            if num % 2 == 0:
                num = num / 2
                count +=1
            else:
                num = num - 1
                count += 1
        return count
```

1321. Restaurant Growth

Table: Customer

Column Name	Type
customer_id	int
name	varchar
visited_on	date
amount	int

(customer_id, visited_on) is the primary key for this table.

This table contains data about customer transactions in a restaurant.

visited_on is the date on which the customer with ID (customer_id) has visited the restaurant.
amount is the total paid by a customer.

You are the restaurant owner and you want to analyze a possible expansion (there will be at least one customer every day).

Write an SQL query to compute the moving average of how much the customer paid in a seven days window (i.e., current day + 6 days before). average_amount should be **rounded to two decimal places**.

Return result table ordered by `visited_on` in ascending order.

The query result format is in the following example.

Example 1:

Input:

Customer table:

customer_id	name	visited_on	amount
1	Jhon	2019-01-01	100
2	Daniel	2019-01-02	110
3	Jade	2019-01-03	120
4	Khaled	2019-01-04	130
5	Winston	2019-01-05	110
6	Elvis	2019-01-06	140
7	Anna	2019-01-07	150
8	Maria	2019-01-08	80
9	Jaze	2019-01-09	110
1	Jhon	2019-01-10	130
3	Jade	2019-01-10	150

Output:

visited_on	amount	average_amount
2019-01-07	860	122.86
2019-01-08	840	120
2019-01-09	840	120
2019-01-10	1000	142.86

Explanation:

1st moving average from 2019-01-01 to 2019-01-07 has an average_amount of $(100 + 110) / 7 = 122.86$
 2nd moving average from 2019-01-02 to 2019-01-08 has an average_amount of $(110 + 120) / 7 = 120$
 3rd moving average from 2019-01-03 to 2019-01-09 has an average_amount of $(120 + 130) / 7 = 120$
 4th moving average from 2019-01-04 to 2019-01-10 has an average_amount of $(130 + 110) / 7 = 142.86$



```

WITH CTE AS
(
    SELECT
        DISTINCT
        VISITED_ON,
        SUM(AMOUNT) OVER(PARTITION BY VISITED_ON) AS AMT
    FROM CUSTOMER
)
SELECT
    VISITED_ON,
    FROM CTE
WHERE VISITED_ON BETWEEN

```

- Partially Executed Solution

```

WITH CTE AS
(
    SELECT
        DISTINCT
        VISITED_ON,
        SUM(AMOUNT) OVER(PARTITION BY VISITED_ON) AS AMT
    FROM CUSTOMER
)

SELECT
    A.VISITED_ON,
    A.AMOUNT,
    A.AVERAGE_AMOUNT
FROM
(
    SELECT
        ID,
        VISITED_ON,
        SUM(AMT) OVER(ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS AMOUNT,
        ROUND(AVG(AMT) OVER(ROWS BETWEEN 6 PRECEDING AND CURRENT ROW),2) AS AVERAGE_AMOUNT
    FROM CTE
    GROUP BY VISITED_ON
) AS A
WHERE A.ID > 6
ORDER BY A.VISITED_ON

```

- Final Working Code

```

WITH CTE AS
(
    SELECT
        DISTINCT
        VISITED_ON,
        ROW_NUMBER() OVER() AS ID,
        SUM(AMOUNT) OVER(PARTITION BY VISITED_ON) AS AMT
    FROM CUSTOMER
)

SELECT
A.VISITED_ON,
A.AMOUNT,
A.AVERAGE_AMOUNT
FROM
(
    SELECT
        ROW_NUMBER() OVER() AS ID,
        VISITED_ON,
        SUM(AMT) OVER(ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS AMOUNT,
        ROUND(AVG(AMT) OVER(ROWS BETWEEN 6 PRECEDING AND CURRENT ROW),2)
        AS AVERAGE_AMOUNT
    FROM CTE
    GROUP BY VISITED_ON
    ORDER BY VISITED_ON
) AS A
WHERE A.ID >= 7

```

1322. Ads Performance ↗

Table: Ads

Column Name	Type
ad_id	int
user_id	int
action	enum

(ad_id, user_id) is the primary key for this table.

Each row of this table contains the ID of an Ad, the ID of a user, and the action taken. The action column is an ENUM type of ('Clicked', 'Viewed', 'Ignored').

A company is running Ads and wants to calculate the performance of each Ad.

Performance of the Ad is measured using Click-Through Rate (CTR) where:

$$CTR = \begin{cases} 0, & \text{if Ad total clicks + Ad total views} = 0 \\ \frac{\text{Ad total clicks}}{\text{Ad total clicks} + \text{Ad total views}} \times 100, & \text{otherwise} \end{cases}$$

Write an SQL query to find the `ctr` of each Ad. **Round** `ctr` to **two decimal points**.

Return the result table ordered by `ctr` in **descending order** and by `ad_id` in **ascending order** in case of a tie.

The query result format is in the following example.

Example 1:

Input:

Ads table:

ad_id	user_id	action
1	1	Clicked
2	2	Clicked
3	3	Viewed
5	5	Ignored
1	7	Ignored
2	7	Viewed
3	5	Clicked
1	4	Viewed
2	11	Viewed
1	2	Clicked

Output:

ad_id	ctr
1	66.67
3	50.00
2	33.33
5	0.00

Explanation:

for `ad_id = 1`, $ctr = (2/(2+1)) * 100 = 66.67$

for `ad_id = 2`, $ctr = (1/(1+2)) * 100 = 33.33$

for `ad_id = 3`, $ctr = (1/(1+1)) * 100 = 50.00$

for `ad_id = 5`, $ctr = 0.00$, Note that `ad_id = 5` has no clicks or views.

Note that we do not care about Ignored Ads.

- Solution Using CTE + UNION + Where + CASE + Group BY + Right Join

```
WITH SUMMARY AS
(
    SELECT
        AD_ID,
        ACTION,
        COUNT(*) AS IMPRESSIONS
    FROM ADS
    WHERE ACTION IN ('Viewed','Clicked')
    GROUP BY AD_ID, ACTION

    UNION

    SELECT
        AD_ID,
        ACTION,
        0 AS IMPRESSIONS
    FROM ADS
    WHERE ACTION NOT IN ('Viewed','Clicked')
    GROUP BY AD_ID
)

SELECT
    T2.AD_ID,
    CASE
        WHEN T1.AD_ID IS NULL THEN 0.00
        ELSE ROUND((T1.TOTAL_CLICKS / T2.CNIEWS) * 100 , 2)
    END AS CTR
FROM
(
    SELECT
        AD_ID,
        SUM(IMPRESSIONS) AS TOTAL_CLICKS
    FROM SUMMARY
    WHERE ACTION = 'Clicked'
    GROUP BY AD_ID
) AS T1
RIGHT JOIN
(
    SELECT
        AD_ID,
        SUM(IMPRESSIONS) AS CNIEWS
    FROM SUMMARY
    GROUP BY AD_ID
) T2
ON T2.AD_ID = T1.AD_ID
ORDER BY CTR DESC, AD_ID ASC
```

1356. Sort Integers by The Number of 1 Bits ↗



You are given an integer array `arr`. Sort the integers in the array in ascending order by the number of `1`'s in their binary representation and in case of two or more integers have the same number of `1`'s you have to sort them in ascending order.

Return *the array after sorting it.*

Example 1:

```
Input: arr = [0,1,2,3,4,5,6,7,8]
Output: [0,1,2,4,8,3,5,6,7]
Explanation: [0] is the only integer with 0 bits.
[1,2,4,8] all have 1 bit.
[3,5,6] have 2 bits.
[7] has 3 bits.
The sorted array by bits is [0,1,2,4,8,3,5,6,7]
```

Example 2:

```
Input: arr = [1024,512,256,128,64,32,16,8,4,2,1]
Output: [1,2,4,8,16,32,64,128,256,512,1024]
Explanation: All integers have 1 bit in the binary representation, you should just so
```

Constraints:

- $1 \leq \text{arr.length} \leq 500$
 - $0 \leq \text{arr}[i] \leq 10^4$
- Used Bisect inosrt to store the values in a ascending order.
 - TC O(Log N) Memory: O(N)

```
class Solution:
    import bisect
    def sortByBits(self, arr: List[int]) -> List[int]:
        res = []
        for val in arr:
            bisect.insort_left(res,(val,bin(val)[2:].count('1')),key=lambda r:(r[1],r[0]))
        return [val[0] for val in res]
```

...

1341. Movie Rating ↗



Table: Movies

Column Name	Type
movie_id	int
title	varchar

movie_id is the primary key for this table.
title is the name of the movie.

Table: Users

Column Name	Type
user_id	int
name	varchar

user_id is the primary key for this table.

Table: MovieRating

Column Name	Type
movie_id	int
user_id	int
rating	int
created_at	date

(movie_id, user_id) is the primary key for this table.
This table contains the rating of a movie by a user in their review.
created_at is the user's review date.

Write an SQL query to:

- Find the name of the user who has rated the greatest number of movies. In case of a tie, return the lexicographically smaller user name.
- Find the movie name with the **highest average** rating in February 2020 . In case of a tie, return the lexicographically smaller movie name.

The query result format is in the following example.

Example 1:**Input:**

Movies table:

movie_id	title
1	Avengers
2	Frozen 2
3	Joker

Users table:

user_id	name
1	Daniel
2	Monica
3	Maria
4	James

MovieRating table:

movie_id	user_id	rating	created_at
1	1	3	2020-01-12
1	2	4	2020-02-11
1	3	2	2020-02-12
1	4	1	2020-01-01
2	1	5	2020-02-17
2	2	2	2020-02-01
2	3	2	2020-03-01
3	1	3	2020-02-22
3	2	4	2020-02-25

Output:

results
Daniel
Frozen 2

Explanation:

Daniel and Monica have rated 3 movies ("Avengers", "Frozen 2" and "Joker") but Daniel and Monica have a rating average of 3.5 in February but Frozen 2 is smaller.



- Solution Using RANK + SORTING

```

(
SELECT
    A.NAME AS RESULTS
FROM
(
    SELECT
        MR.USER_ID,
        U.NAME,
        DENSE_RANK() OVER(ORDER BY COUNT(*) DESC) AS CRANK
    FROM MOVIES AS M LEFT JOIN MOVIERATING AS MR
    ON M.MOVIE_ID = MR.MOVIE_ID RIGHT JOIN USERS AS U
    ON MR.USER_ID = U.USER_ID
    GROUP BY MR.USER_ID , U.NAME
) AS A
WHERE A.CRANK = 1
ORDER BY A.NAME ASC LIMIT 1
)

UNION

SELECT TITLE AS RESULTS
FROM
(
    SELECT
        DISTINCT
        M.TITLE,
        MONTH(CREATED_AT) AS MONTH,
        AVG(MR.RATING) OVER(PARTITION BY M.TITLE)
        AS AVG_RATING
    FROM MOVIES AS M LEFT JOIN MOVIERATING AS MR
    ON M.MOVIE_ID = MR.MOVIE_ID RIGHT JOIN USERS AS U
    ON MR.USER_ID = U.USER_ID
    WHERE DATE_FORMAT(CREATED_AT, '%Y-%m') = '2020-02'
    ORDER BY AVG_RATING DESC, TITLE ASC LIMIT 1
)

) AS A

```

- Using only SORTING

```
(  
SELECT  
    NAME AS RESULTS  
FROM  
(  
    SELECT  
        U.NAME,  
        COUNT(*) AS RATED_MOVIES  
    FROM MOVIES AS M LEFT JOIN MOVIERATING AS MR  
    ON M.MOVIE_ID = MR.MOVIE_ID RIGHT JOIN USERS AS U  
    ON MR.USER_ID = U.USER_ID  
    GROUP BY MR.USER_ID , U.NAME  
    ORDER BY RATED_MOVIES DESC, NAME ASC LIMIT 1  
) AS A  
)  
  
UNION  
  
SELECT TITLE AS RESULTS  
FROM  
(  
    SELECT  
        DISTINCT  
        M.TITLE,  
        MONTH(CREATED_AT) AS MONTH,  
        AVG(MR.RATING) OVER(PARTITION BY M.TITLE)  
        AS AVG_RATING  
    FROM MOVIES AS M LEFT JOIN MOVIERATING AS MR  
    ON M.MOVIE_ID = MR.MOVIE_ID RIGHT JOIN USERS AS U  
    ON MR.USER_ID = U.USER_ID  
    WHERE DATE_FORMAT(CREATED_AT, '%Y-%m') = '2020-02'  
    ORDER BY AVG_RATING DESC, TITLE ASC LIMIT 1  
) AS A
```

1350. Students With Invalid Departments ↗

Table: Departments

Column Name	Type
id	int
name	varchar

id is the primary key of this table.

The table has information about the id of each department of a university.

Table: Students

Column Name	Type
id	int
name	varchar
department_id	int

id is the primary key of this table.

The table has information about the id of each student at a university and the id of

Write an SQL query to find the id and the name of all students who are enrolled in departments that no longer exist.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Departments table:

id name
1 Electrical Engineering
7 Computer Engineering
13 Bussiness Administration

Students table:

id name department_id
23 Alice 1
1 Bob 7
5 Jennifer 13
2 John 14
4 Jasmine 77
3 Steve 74
6 Luis 1
8 Jonathan 7
7 Daiana 33
11 Madelynn 1

Output:

id name
2 John
7 Daiana
4 Jasmine
3 Steve

Explanation:

John, Daiana, Steve, and Jasmine are enrolled in departments 14, 33, 74, and 77 respectively.

- Right Join + Handling Nulls Solves the Problem

```

SELECT
S.ID,
S.NAME
FROM DEPARTMENTS AS D RIGHT JOIN STUDENTS AS S
ON D.ID = S.DEPARTMENT_ID
WHERE D.ID IS NULL
    
```



1355. Activity Participants ↗

Table: Friends

Column Name	Type
id	int
name	varchar
activity	varchar

id is the id of the friend and primary key for this table.

name is the name of the friend.

activity is the name of the activity which the friend takes part in.

Table: Activities

Column Name	Type
id	int
name	varchar

id is the primary key for this table.

name is the name of the activity.

Write an SQL query to find the names of all the activities with neither the maximum nor the minimum number of participants.

Each activity in the Activities table is performed by any person in the table Friends.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Friends table:

id	name	activity
1	Jonathan D.	Eating
2	Jade W.	Singing
3	Victor J.	Singing
4	Elvis Q.	Eating
5	Daniel A.	Eating
6	Bob B.	Horse Riding

Activities table:

id	name
1	Eating
2	Singing
3	Horse Riding

Output:

activity
Singing

Explanation:

Eating activity is performed by 3 friends, maximum number of participants, (Jonathan Horse Riding activity is performed by 1 friend, minimum number of participants, (Bob Singing is performed by 2 friends (Victor J. and Jade W.)



```

WITH CTE AS
(
    SELECT
        ACTIVITY,
        COUNT(*) AS TOTAL
    FROM FRIENDS
    GROUP BY ACTIVITY
),

CTE2 AS
(
    SELECT
        MAX(TOTAL),
        MIN(TOTAL)
    FROM CTE
)

SELECT *
FROM CTE
WHERE TOTAL IN (SELECT   FROM CTE2)

```

1364. Number of Trusted Contacts of a Customer ↗



Table: Customers

Column Name	Type
customer_id	int
customer_name	varchar
email	varchar

customer_id is the primary key for this table.

Each row of this table contains the name and the email of a customer of an online sh



Table: Contacts

Column Name	Type
user_id	id
contact_name	varchar
contact_email	varchar

(user_id, contact_email) is the primary key for this table.

Each row of this table contains the name and email of one contact of customer with user_id.

This table contains information about people each customer trust. The contact may or

Table: Invoices

Column Name	Type
invoice_id	int
price	int
user_id	int

invoice_id is the primary key for this table.

Each row of this table indicates that user_id has an invoice with invoice_id and a price.

Write an SQL query to find the following for each invoice_id :

- **customer_name** : The name of the customer the invoice is related to.
- **price** : The price of the invoice.
- **contacts_cnt** : The number of contacts related to the customer.
- **trusted_contacts_cnt** : The number of contacts related to the customer and at the same time they are customers to the shop. (i.e their email exists in the Customers table.)

Return the result table **ordered** by invoice_id .

The query result format is in the following example.

Example 1:

Input:

Customers table:

customer_id	customer_name	email
1	Alice	alice@leetcode.com
2	Bob	bob@leetcode.com
13	John	john@leetcode.com
6	Alex	alex@leetcode.com

Contacts table:

user_id	contact_name	contact_email
1	Bob	bob@leetcode.com
1	John	john@leetcode.com
1	Jal	jal@leetcode.com
2	Omar	omar@leetcode.com
2	Meir	meir@leetcode.com
6	Alice	alice@leetcode.com

Invoices table:

invoice_id	price	user_id
77	100	1
88	200	1
99	300	2
66	400	2
55	500	13
44	60	6

Output:

invoice_id	customer_name	price	contacts_cnt	trusted_contacts_cnt
44	Alex	60	1	1
55	John	500	0	0
66	Bob	400	2	0
77	Alice	100	3	2
88	Alice	200	3	2
99	Bob	300	2	0

Explanation:

Alice has three contacts, two of them are trusted contacts (Bob and John).

Bob has two contacts, none of them is a trusted contact.

Alex has one contact and it is a trusted contact (Alice).

John doesn't have any contacts.

```

WITH CTE1 AS
(
    SELECT
        I.INVOICE_ID,
        C.CUSTOMER_NAME,
        I.PRICE,
        CASE
            WHEN COUNT(CT.USER_ID) IS NOT NULL THEN COUNT(CT.USER_ID)
            ELSE 0
        END AS CONTACTS_CNT
        # N:1 relationship
        FROM INVOICES AS I LEFT JOIN CUSTOMERS AS C
        # M:N RELATION AFTER RESULT FROM ABOVE ON CONTACTS
        ON I.USER_ID = C.CUSTOMER_ID LEFT JOIN CONTACTS AS CT
        ON C.CUSTOMER_ID = CT.USER_ID
        GROUP BY
            I.INVOICE_ID,
            C.CUSTOMER_NAME,
            I.PRICE
),
CTE2 AS
(
    SELECT
        I.INVOICE_ID,
        C.CUSTOMER_NAME,
        I.PRICE,
        CASE
            WHEN COUNT(CT.USER_ID) IS NOT NULL THEN COUNT(CT.USER_ID)
            ELSE 0
        END AS TRUSTED_CONTACTS_CNT
        # N:1 relationship
        FROM INVOICES AS I LEFT JOIN CUSTOMERS AS C
        # M:N RELATION AFTER RESULT FROM ABOVE ON CONTACTS
        ON I.USER_ID = C.CUSTOMER_ID LEFT JOIN
        (SELECT * FROM CONTACTS WHERE CONTACT_EMAIL IN
        (SELECT EMAIL FROM CUSTOMERS WHERE EMAIL IN
        (SELECT CONTACT_EMAIL FROM CONTACTS))) AS CT
        ON C.CUSTOMER_ID = CT.USER_ID
        GROUP BY
            I.INVOICE_ID,
            C.CUSTOMER_NAME,
            I.PRICE
)
SELECT
    CT1.INVOICE_ID,
    CT1.CUSTOMER_NAME,
    CT1.PRICE,
    CT1.CONTACTS_CNT,
    CT2.TRUSTED_CONTACTS_CNT
    FROM CTE1 AS CT1 JOIN CTE2 AS CT2

```

```
ON CT1.INVOICE_ID = CT2.INVOICE_ID AND CT1.CUSTOMER_NAME = CT2.CUSTOMER_NAME
ORDER BY CT1.INVOICE_ID
```

1378. Replace Employee ID With The Unique Identifier ↗



Table: Employees

Column Name	Type
<code>id</code>	<code>int</code>
<code>name</code>	<code>varchar</code>

`id` is the primary key for this table.

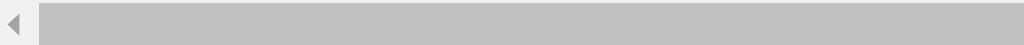
Each row of this table contains the `id` and the name of an employee in a company.

Table: EmployeeUNI

Column Name	Type
<code>id</code>	<code>int</code>
<code>unique_id</code>	<code>int</code>

(`id`, `unique_id`) is the primary key for this table.

Each row of this table contains the `id` and the corresponding unique id of an employee.



Write an SQL query to show the **unique ID** of each user. If a user does not have a unique ID replace just show `null`.

Return the result table in **any** order.

The query result format is in the following example.

Example 1:

Input:

Employees table:

id	name
1	Alice
7	Bob
11	Meir
90	Winston
3	Jonathan

EmployeeUNI table:

id	unique_id
3	1
11	2
90	3

Output:

unique_id	name
null	Alice
null	Bob
2	Meir
3	Winston
1	Jonathan

Explanation:

Alice and Bob do not have a unique ID, We will show null instead.

The unique ID of Meir is 2.

The unique ID of Winston is 3.

The unique ID of Jonathan is 1.

- Actually Written a Correct Query at First Itself But due to lack of sleep

```

SELECT
EU.UNIQUE_ID,
E.NAME
FROM EMPLOYEES AS E LEFT JOIN EMPLOYEEUNI AS EU
ON E.ID = EU.ID

```

1393. Capital Gain/Loss ↗



Table: Stocks

Column Name	Type
stock_name	varchar
operation	enum
operation_day	int
price	int

(stock_name, operation_day) is the primary key for this table.

The operation column is an ENUM of type ('Sell', 'Buy')

Each row of this table indicates that the stock which has stock_name had an operation on day operation_day at price price.

It is guaranteed that each 'Sell' operation for a stock has a corresponding 'Buy' operation for it on some day.

Write an SQL query to report the **Capital gain/loss** for each stock.

The **Capital gain/loss** of a stock is the total gain or loss after buying and selling the stock one or many times.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Stocks table:

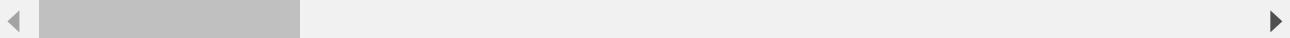
stock_name	operation	operation_day	price
Leetcode	Buy	1	1000
Corona Masks	Buy	2	10
Leetcode	Sell	5	9000
Handbags	Buy	17	30000
Corona Masks	Sell	3	1010
Corona Masks	Buy	4	1000
Corona Masks	Sell	5	500
Corona Masks	Buy	6	1000
Handbags	Sell	29	7000
Corona Masks	Sell	10	10000

Output:

stock_name	capital_gain_loss
Corona Masks	9500
Leetcode	8000
Handbags	-23000

Explanation:

Leetcode stock was bought at day 1 for 1000\$ and was sold at day 5 for 9000\$. Capital gain is 9000 - 1000 = 8000\$.
 Handbags stock was bought at day 17 for 30000\$ and was sold at day 29 for 7000\$. Capital loss is 7000 - 30000 = -23000\$.
 Corona Masks stock was bought at day 1 for 10\$ and was sold at day 3 for 1010\$. It was bought at day 4 for 1000\$ and was sold at day 5 for 1010\$. Capital gain is 1010 - 10 = 1000\$.



- MYSql Solution for the above problem Statement using CTE and SUM Functions

```

WITH STOCK_SUMMARY AS
(
    SELECT
        STOCK_NAME,
        CASE
            WHEN OPERATION = 'Buy' THEN SUM(PRICE)
        END AS BUYING_PRICE,
        CASE
            WHEN OPERATION = 'Sell' THEN SUM(PRICE)
        END AS SELLING_PRICE
    FROM STOCKS
    GROUP BY STOCK_NAME, OPERATION
)
SELECT
    STOCK_NAME,
    SUM(SELLING_PRICE) - SUM(BUYING_PRICE) AS CAPITAL_GAIN_LOSS
FROM STOCK_SUMMARY
GROUP BY STOCK_NAME

```

1398. Customers Who Bought Products A and B but Not C ↴

Table: Customers

Column Name	Type
customer_id	int
customer_name	varchar

customer_id is the primary key for this table.
customer_name is the name of the customer.

Table: Orders

Column Name	Type
order_id	int
customer_id	int
product_name	varchar

order_id is the primary key for this table.
customer_id is the id of the customer who bought the product "product_name".

Write an SQL query to report the customer_id and customer_name of customers who bought products "A", "B" but did not buy the product "C" since we want to recommend them to purchase this product.

Return the result table **ordered** by customer_id .

The query result format is in the following example.

Example 1:

Input:

Customers table:

customer_id	customer_name
1	Daniel
2	Diana
3	Elizabeth
4	Jhon

Orders table:

order_id	customer_id	product_name
10	1	A
20	1	B
30	1	D
40	1	C
50	2	A
60	3	A
70	3	B
80	3	D
90	4	C

Output:

customer_id	customer_name
3	Elizabeth

Explanation: Only the customer_id with id 3 bought the product A and B but not the p

```

SELECT B.CUSTOMER_ID, B.CUSTOMER_NAME
FROM
(
    SELECT
        A.CUSTOMER_ID,
        A.CUSTOMER_NAME,
        MAX(PRODUCT_A) AS PROD_A,
        MAX(PRODUCT_B) AS PROD_B,
        MAX(PRODUCT_C) AS PROD_C
    FROM
    (
        SELECT
            C.CUSTOMER_ID,
            C.CUSTOMER_NAME,
            CASE WHEN O.PRODUCT_NAME = 'A' THEN 'A' ELSE NULL END AS PRODUCT_A,
            CASE WHEN O.PRODUCT_NAME = 'B' THEN 'B' ELSE NULL END AS PRODUCT_B,
            CASE WHEN O.PRODUCT_NAME = 'C' THEN 'C' ELSE NULL END AS PRODUCT_C
        FROM CUSTOMERS AS C LEFT JOIN ORDERS AS O
        ON C.CUSTOMER_ID = O.CUSTOMER_ID
    ) AS A
    GROUP BY A.CUSTOMER_ID, A.CUSTOMER_NAME
) AS B
WHERE (B.PROD_A IS NOT NULL AND B.PROD_B IS NOT NULL) AND B.PROD_C IS NULL
ORDER BY B.CUSTOMER_ID ASC

```

1407. Top Travellers ↗

Table: Users

Column Name	Type
<code>id</code>	<code>int</code>
<code>name</code>	<code>varchar</code>

`id` is the primary key for this table.
`name` is the name of the user.

Table: Rides

Column Name	Type
id	int
user_id	int
distance	int

`id` is the primary key for this table.

`user_id` is the id of the user who traveled the distance "distance".

Write an SQL query to report the distance traveled by each user.

Return the result table ordered by `travelled_distance` in **descending order**, if two or more users traveled the same distance, order them by their `name` in **ascending order**.

The query result format is in the following example.

Example 1:

Input:

Users table:

id	name
1	Alice
2	Bob
3	Alex
4	Donald
7	Lee
13	Jonathan
19	Elvis

Rides table:

id	user_id	distance
1	1	120
2	2	317
3	3	222
4	7	100
5	13	312
6	19	50
7	7	120
8	19	400
9	7	230

Output:

name	travelled_distance
Elvis	450
Lee	450
Bob	317
Jonathan	312
Alex	222
Alice	120
Donald	0

Explanation:

Elvis and Lee traveled 450 miles, Elvis is the top traveler as his name is alphabetical.
 Bob, Jonathan, Alex, and Alice have only one ride and we just order them by the total distance traveled.
 Donald did not have any rides, the distance traveled by him is 0.



- Mysql Solution Using Left Join and Group By

```

SELECT
U.NAME,
CASE
    WHEN SUM(R.DISTANCE) IS NULL THEN 0
    ELSE SUM(R.DISTANCE)
END AS TRAVELED_DISTANCE
FROM USERS AS U LEFT JOIN RIDES AS R
ON U.ID = R.USER_ID
GROUP BY U.NAME, U.ID
ORDER BY TRAVELED_DISTANCE DESC, NAME ASC

```

1421. NPV Queries ↗



Table: NPV

Column Name	Type
id	int
year	int
npv	int

(id, year) is the primary key of this table.

The table has information about the id and the year of each inventory and the corres

Table: Queries

Column Name	Type
id	int
year	int

(id, year) is the primary key of this table.

The table has information about the id and the year of each inventory query.

Write an SQL query to find the npv of each query of the Queries table.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:**Input:**

NPV table:

id	year	npv
1	2018	100
7	2020	30
13	2019	40
1	2019	113
2	2008	121
3	2009	12
11	2020	99
7	2019	0

Queries table:

id	year
1	2019
2	2008
3	2009
7	2018
7	2019
7	2020
13	2019

Output:

id	year	npv
1	2019	113
2	2008	121
3	2009	12
7	2018	0
7	2019	0
7	2020	30
13	2019	40

Explanation:

The npv value of (7, 2018) is not present in the NPV table, we consider it 0.

The npv values of all other queries can be found in the NPV table.

- Right Join + CASE Solves the problem

```

SELECT
Q.ID,
Q.YEAR,
CASE
    WHEN N.NPV IS NULL THEN 0
    ELSE N.NPV
END AS NPV
FROM NPV AS N RIGHT JOIN QUERIES AS Q
ON N.ID = Q.ID AND N.YEAR = Q.YEAR

```

1435. Create a Session Bar Chart ↗



Table: Sessions

Column Name	Type
session_id	int
duration	int

session_id is the primary key for this table.

duration is the time in seconds that a user has visited the application.

You want to know how long a user visits your application. You decided to create bins of "[0-5>" , "[5-10>" , "[10-15>" , and "15 minutes or more" and count the number of sessions on it.

Write an SQL query to report the (bin, total) .

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Sessions table:

session_id	duration
1	30
2	199
3	299
4	580
5	1000

Output:

bin	total
[0-5>	3
[5-10>	1
[10-15>	0
15 or more	1

Explanation:

For session_id 1, 2, and 3 have a duration greater or equal than 0 minutes and less than 5 minutes.

For session_id 4 has a duration greater or equal than 5 minutes and less than 10 minutes.

There is no session with a duration greater than or equal to 10 minutes and less than 15 minutes.

For session_id 5 has a duration greater than or equal to 15 minutes.



```

SELECT
T1.BIN,
COUNT(*) AS TOTAL
FROM SESSIONS AS S RIGHT JOIN
(SELECT
CASE
WHEN (DURATION/60) < 5 THEN "[0-5]"
WHEN (DURATION/60) >= 5 OR (DURATION/60) < 10 THEN "[5-10]"
WHEN (DURATION/60) >= 10 OR (DURATION/60) < 15 THEN "[10-15]"
ELSE "15 or more"
END AS BIN, SESSION_ID
FROM SESSIONS
) AS T1
ON S.SESSION_ID = T1.SESSION_ID
GROUP BY T1.BIN

```

```

SELECT
CASE
WHEN (DURATION/60) < 5 THEN "[0-5]"
WHEN (DURATION/60) BETWEEN 5 AND 9.9999 THEN "[5-10]"
WHEN (DURATION/60) BETWEEN 10 AND 14.999 THEN "[10-15]"
ELSE "15 or more"
END AS BIN, SESSION_ID
FROM SESSIONS

```

1440. Evaluate Boolean Expression ↗



Table Variables :

Column Name	Type
name	varchar
value	int

name is the primary key for this table.

This table contains the stored variables and their values.

Table Expressions :

Column Name	Type
left_operand	varchar
operator	enum
right_operand	varchar

(left_operand, operator, right_operand) is the primary key for this table.

This table contains a boolean expression that should be evaluated.

operator is an enum that takes one of the values ('<', '>', '=')

The values of left_operand and right_operand are guaranteed to be in the Variables table.



Write an SQL query to evaluate the boolean expressions in Expressions table.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Variables table:

name	value
x	66
y	77

Expressions table:

left_operand	operator	right_operand
x	>	y
x	<	y
x	=	y
y	>	x
y	<	x
x	=	x

Output:

left_operand	operator	right_operand	value
x	>	y	false
x	<	y	true
x	=	y	false
y	>	x	true
y	<	x	false
x	=	x	true

Explanation:

As shown, you need to find the value of each boolean expression in the table using the variables' values.



```

SELECT
LFT.LEFT_OPERAND,
LFT.OPERATOR,
RGT.RIGHT_OPERAND,
CASE
    WHEN LFT.OPERATOR = '>' THEN
        CASE
            WHEN LFT.VALUE > RGT.VALUE THEN 'true'
            ELSE 'false'
        END
    WHEN LFT.OPERATOR = '<' THEN
        CASE
            WHEN LFT.VALUE < RGT.VALUE THEN 'true'
            ELSE 'false'
        END
    WHEN LFT.OPERATOR = '=' THEN
        CASE
            WHEN LFT.VALUE = RGT.VALUE THEN 'true'
            ELSE 'false'
        END
END AS VALUE
FROM
(
(
    SELECT
        ROW_NUMBER() OVER() AS ID,
        V.VALUE,
        E.LEFT_OPERAND,
        E.OPERATOR
        FROM VARIABLES AS V JOIN EXPRESSIONS AS E
        ON V.NAME = E.LEFT_OPERAND
    ) AS LFT
    JOIN
    (
    SELECT
        ROW_NUMBER() OVER() AS ID,
        V.VALUE,
        E.RIGHT_OPERAND,
        E.OPERATOR
        FROM VARIABLES AS V JOIN EXPRESSIONS AS E
        ON V.NAME = E.RIGHT_OPERAND
    ) AS RGT
    ON LFT.ID = RGT.ID AND LFT.OPERATOR = RGT.OPERATOR
)

```

1445. Apples & Oranges ↗



Table: Sales

Column Name	Type
sale_date	date
fruit	enum
sold_num	int

(sale_date, fruit) is the primary key for this table.

This table contains the sales of "apples" and "oranges" sold each day.

Write an SQL query to report the difference between the number of **apples** and **oranges** sold each day.

Return the result table **ordered** by `sale_date`.

The query result format is in the following example.

Example 1:

Input:

Sales table:

sale_date	fruit	sold_num
2020-05-01	apples	10
2020-05-01	oranges	8
2020-05-02	apples	15
2020-05-02	oranges	15
2020-05-03	apples	20
2020-05-03	oranges	0
2020-05-04	apples	15
2020-05-04	oranges	16

Output:

sale_date	diff
2020-05-01	2
2020-05-02	0
2020-05-03	20
2020-05-04	-1

Explanation:

Day 2020-05-01, 10 apples and 8 oranges were sold (Difference $10 - 8 = 2$).

Day 2020-05-02, 15 apples and 15 oranges were sold (Difference $15 - 15 = 0$).

Day 2020-05-03, 20 apples and 0 oranges were sold (Difference $20 - 0 = 20$).

Day 2020-05-04, 15 apples and 16 oranges were sold (Difference $15 - 16 = -1$).

- CTE + Group BY Sums up the Solution.

```

WITH SOLD_SUMMARY AS
(
    SELECT
        SALE_DATE,
        CASE
            WHEN FRUIT = 'apples' THEN SOLD_NUM
        END AS APPLES_SOLD,
        CASE
            WHEN FRUIT = 'oranges' THEN SOLD_NUM
        END AS ORANGES_SOLD
    FROM SALES
    GROUP BY SALE_DATE, FRUIT
)

SELECT
    SALE_DATE,
    SUM(APPLES_SOLD) - SUM(ORANGES_SOLD) AS DIFF
FROM SOLD_SUMMARY
GROUP BY SALE_DATE
ORDER BY SALE_DATE

```

1491. Average Salary Excluding the Minimum and Maximum Salary ↗

You are given an array of **unique** integers `salary` where `salary[i]` is the salary of the i^{th} employee.

Return *the average salary of employees excluding the minimum and maximum salary*. Answers within 10^{-5} of the actual answer will be accepted.

Example 1:

```

Input: salary = [4000,3000,1000,2000]
Output: 2500.00000
Explanation: Minimum salary and maximum salary are 1000 and 4000 respectively.
Average salary excluding minimum and maximum salary is (2000+3000) / 2 = 2500

```

Example 2:

```

Input: salary = [1000,2000,3000]
Output: 2000.00000
Explanation: Minimum salary and maximum salary are 1000 and 3000 respectively.
Average salary excluding minimum and maximum salary is (2000) / 1 = 2000

```

Constraints:

- $3 \leq \text{salary.length} \leq 100$
- $1000 \leq \text{salary}[i] \leq 10^6$
- All the integers of `salary` are **unique**.

Method 1

```
class Solution:
    def average(self, salary: List[int]) -> float:
        salary.remove(min(salary))
        salary.remove(max(salary))
        return sum(salary)/ len(salary)
```

Method 2

```
class Solution:
    def average(self, salary: List[int]) -> float:
        length = len(salary)
        return sum(sorted(salary)[1:length-1])/ (length-2)
```

1480. Running Sum of 1d Array ↗

Given an array `nums`. We define a running sum of an array as `runningSum[i] = sum(nums[0]...nums[i])`.

Return the running sum of `nums`.

Example 1:

Input: `nums = [1,2,3,4]`
Output: `[1,3,6,10]`
Explanation: Running sum is obtained as follows: `[1, 1+2, 1+2+3, 1+2+3+4]`.

Example 2:

Input: `nums = [1,1,1,1,1]`
Output: `[1,2,3,4,5]`
Explanation: Running sum is obtained as follows: `[1, 1+1, 1+1+1, 1+1+1+1, 1+1+1+1+1]`

Example 3:

Input: nums = [3,1,2,10,1]

Output: [3,4,6,16,17]

Constraints:

- $1 \leq \text{nums.length} \leq 1000$
- $-10^6 \leq \text{nums}[i] \leq 10^6$

```
# Code Author Naveen Kumar Vadlamudi
class Solution:
    def runningSum(self, nums: List[int]) -> List[int]:
        res = []
        running_sum = 0
        # O(N) Time Complexity
        for val in nums:
            running_sum += val
            res.append(running_sum)
        return res
```

1468. Calculate Salaries ↗

Table Salaries :

Column Name	Type
company_id	int
employee_id	int
employee_name	varchar
salary	int

(company_id, employee_id) is the primary key for this table.

This table contains the company id, the id, the name, and the salary for an employee

Write an SQL query to find the salaries of the employees after applying taxes. Round the salary to **the nearest integer**.

The tax rate is calculated for each company based on the following criteria:

- 0% If the max salary of any employee in the company is less than \$1000 .
- 24% If the max salary of any employee in the company is in the range [1000, 10000] inclusive.

- 49% If the max salary of any employee in the company is greater than \$10000 .

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Salaries table:

company_id	employee_id	employee_name	salary
1	1	Tony	2000
1	2	Pronub	21300
1	3	Tyrrox	10800
2	1	Pam	300
2	7	Bassem	450
2	9	Hermione	700
3	7	Bocaben	100
3	2	Ognjen	2200
3	13	Nyancat	3300
3	15	Morninngcat	7777

Output:

company_id	employee_id	employee_name	salary
1	1	Tony	1020
1	2	Pronub	10863
1	3	Tyrrox	5508
2	1	Pam	300
2	7	Bassem	450
2	9	Hermione	700
3	7	Bocaben	76
3	2	Ognjen	1672
3	13	Nyancat	2508
3	15	Morninngcat	5911

Explanation:

For company 1, Max salary is 21300. Employees in company 1 have taxes = 49%

For company 2, Max salary is 700. Employees in company 2 have taxes = 0%

For company 3, Max salary is 7777. Employees in company 3 have taxes = 24%

The salary after taxes = salary - (taxes percentage / 100) * salary

For example, Salary for Morninngcat (3, 15) after taxes = 7777 - 7777 * (24 / 100) =



```

SELECT
T.COMPANY_ID,
S.EMPLOYEE_ID,
S.EMPLOYEE_NAME,
ROUND(S.SALARY - S.SALARY * T.PERCENTAGE,0) AS SALARY
FROM SALARIES AS S
RIGHT JOIN
(
    SELECT
COMPANY_ID,
CASE
    WHEN MAX(SALARY) < 1000 THEN 0
    WHEN MAX(SALARY) >= 1000 AND MAX(SALARY) <= 10000 THEN 0.24
    ELSE 0.49
END AS PERCENTAGE
FROM SALARIES
GROUP BY COMPANY_ID
) AS T
ON S.COMPANY_ID = T.COMPANY_ID

```

1484. Group Sold Products By The Date ↗



Table Activities :

Column Name	Type
sell_date	date
product	varchar

There is no primary key for this table, it may contain duplicates.

Each row of this table contains the product name and the date it was sold in a market.

Write an SQL query to find for each date the number of different products sold and their names.

The sold products names for each date should be sorted lexicographically.

Return the result table ordered by `sell_date`.

The query result format is in the following example.

Example 1:

Input:

Activities table:

sell_date	product
2020-05-30	Headphone
2020-06-01	Pencil
2020-06-02	Mask
2020-05-30	Basketball
2020-06-01	Bible
2020-06-02	Mask
2020-05-30	T-Shirt

Output:

sell_date	num_sold	products
2020-05-30	3	Basketball,Headphone,T-shirt
2020-06-01	2	Bible,Pencil
2020-06-02	1	Mask

Explanation:

For 2020-05-30, Sold items were (Headphone, Basketball, T-shirt), we sort them lexicographically and return them as a single string.

For 2020-06-01, Sold items were (Pencil, Bible), we sort them lexicographically and return them as a single string.

For 2020-06-02, the Sold item is (Mask), we just return it.



- MYSQL Based Solution For The above mentioned problem statement.

```
SELECT sell_date,
       count(distinct product) num_sold,
       GROUP_CONCAT(distinct PRODUCT order by PRODUCT ASC) AS products
  FROM ACTIVITIES
 GROUP BY SELL_DATE
 ORDER BY SELL_DATE, PRODUCTS ASC, NUM SOLD DESC
```

- Oracle based Solution for the above problem.

```

SELECT
  to_char(SELL_DATE, 'YYYY-MM-DD') as sell_date,
  count(*) as num_sold,
  LISTAGG(PRODUCT, ',') WITHIN GROUP (ORDER BY PRODUCT) as products
FROM
(
  SELECT
    DISTINCT SELL_DATE, PRODUCT
    FROM ACTIVITIES
)
GROUP BY SELL_DATE
ORDER BY SELL_DATE

```

1502. Can Make Arithmetic Progression From Sequence ↗

A sequence of numbers is called an **arithmetic progression** if the difference between any two consecutive elements is the same.

Given an array of numbers `arr`, return `true` if the array can be rearranged to form an **arithmetic progression**. Otherwise, return `false`.

Example 1:

Input: arr = [3,5,1]

Output: true

Explanation: We can reorder the elements as [1,3,5] or [5,3,1] with differences 2 and



Example 2:

Input: arr = [1,2,4]

Output: false

Explanation: There is no way to reorder the elements to obtain an arithmetic progression.



Constraints:

- $2 \leq \text{arr.length} \leq 1000$
- $-10^6 \leq \text{arr}[i] \leq 10^6$

```
class Solution:
    def canMakeArithmetricProgression(self, arr: List[int]) -> bool:
        flag = 0
        diff = []
        arr.sort()
        diff = [abs(arr[i-1] - arr[i]) for i in range(1, len(arr))]
        if len(set(diff)) == 1:
            return True
        return False
```

```
class Solution:
    def canMakeArithmetricProgression(self, arr: List[int]) -> bool:
        arr.sort()
        return True if len(set([abs(arr[i-1] - arr[i]) for i in range(1, len(arr))])) == 1 else False
```

1523. Count Odd Numbers in an Interval Range

Given two non-negative integers `low` and `high`. Return the *count of odd numbers between low and high (inclusive)*.

Example 1:

Input: `low = 3, high = 7`
Output: 3
Explanation: The odd numbers between 3 and 7 are [3,5,7].

Example 2:

Input: `low = 8, high = 10`
Output: 1
Explanation: The odd numbers between 8 and 10 are [9].

Constraints:

- $0 \leq \text{low} \leq \text{high} \leq 10^9$

```

class Solution:
    import math
    def countOdds(self, low: int, high: int) -> int:
        count = 0
        # both are even
        if low % 2 == 0 and high % 2 == 0:
            return math.floor((high-low+1)/2)
        # one is even and other is odd
        elif low % 2 == 0 and high % 2 !=0:
            return math.floor((high - low + 1) / 2)
        # both are odd
        elif low % 2 != 0 and high % 2 !=0 :
            return math.ceil((high-low+1)/2)
        # one is odd and other is even
        else:
            return math.ceil((high-low+1)/2)

```

1511. Customer Order Frequency ↗



Table: Customers

Column Name	Type
customer_id	int
name	varchar
country	varchar

customer_id is the primary key for this table.

This table contains information about the customers in the company.

Table: Product

Column Name	Type
product_id	int
description	varchar
price	int

product_id is the primary key for this table.

This table contains information on the products in the company.

price is the product cost.

Table: Orders

Column Name	Type
order_id	int
customer_id	int
product_id	int
order_date	date
quantity	int

order_id is the primary key for this table.

This table contains information on customer orders.

customer_id is the id of the customer who bought "quantity" products with id "product_id".

Order_date is the date in format ('YYYY-MM-DD') when the order was shipped.

◀ ▶

Write an SQL query to report the customer_id and customer_name of customers who have spent at least \$100 in each month of **June and July 2020**.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Customers table:

customer_id	name	country
1	Winston	USA
2	Jonathan	Peru
3	Moustafa	Egypt

Product table:

product_id	description	price
10	LC Phone	300
20	LC T-Shirt	10
30	LC Book	45
40	LC Keychain	2

Orders table:

order_id	customer_id	product_id	order_date	quantity
1	1	10	2020-06-10	1
2	1	20	2020-07-01	1
3	1	30	2020-07-08	2
4	2	10	2020-06-15	2
5	2	40	2020-07-01	10
6	3	20	2020-06-24	2
7	3	30	2020-06-25	2
9	3	30	2020-05-08	3

Output:

customer_id	name
1	Winston

Explanation:Winston spent \$300 ($300 * 1$) in June and \$100 ($10 * 1 + 45 * 2$) in July 2020.Jonathan spent \$600 ($300 * 2$) in June and \$20 ($2 * 10$) in July 2020.Moustafa spent \$110 ($10 * 2 + 45 * 2$) in June and \$0 in July 2020.

```

WITH CTE AS
(
    SELECT
        C.CUSTOMER_ID,
        C.NAME,
        SUM(O.QUANTITY * P.PRICE) AS AMT,
        MONTH(O.ORDER_DATE) AS MONTH
    FROM CUSTOMERS AS C LEFT JOIN ORDERS AS O
    ON C.CUSTOMER_ID = O.CUSTOMER_ID LEFT JOIN PRODUCT AS P
    ON O.PRODUCT_ID = P.PRODUCT_ID
    WHERE YEAR(O.ORDER_DATE) = 2020
    GROUP BY C.CUSTOMER_ID, C.NAME, MONTH(O.ORDER_DATE)
)

SELECT CUSTOMER_ID, NAME
FROM CTE
WHERE AMT >= 100 AND MONTH IN (06,07)
GROUP BY CUSTOMER_ID, NAME
HAVING COUNT(*) = 2

```

1517. Find Users With Valid E-Mails ↗

Table: Users

Column Name	Type
user_id	int
name	varchar
mail	varchar

user_id is the primary key for this table.

This table contains information of the users signed up in a website. Some e-mails are

Write an SQL query to find the users who have **valid emails**.

A valid e-mail has a prefix name and a domain where:

- **The prefix name** is a string that may contain letters (upper or lower case), digits, underscore '_', period '.', and/or dash '-'. The prefix name **must** start with a letter.
- **The domain** is '@leetcode.com' .

Return the result table in **any order**.

The query result format is in the following example.

Example 1:**Input:**

Users table:

user_id	name	mail
1	Winston	winston@leetcode.com
2	Jonathan	jonathanisgreat
3	Annabelle	bella-@leetcode.com
4	Sally	sally.come@leetcode.com
5	Marwan	quarz#2020@leetcode.com
6	David	david69@gmail.com
7	Shapiro	.shapo@leetcode.com

Output:

user_id	name	mail
1	Winston	winston@leetcode.com
3	Annabelle	bella-@leetcode.com
4	Sally	sally.come@leetcode.com

Explanation:

The mail of user 2 does not have a domain.

The mail of user 5 has the # sign which is not allowed.

The mail of user 6 does not have the leetcode domain.

The mail of user 7 starts with a period.

```
SELECT *
FROM USERS
WHERE
SUBSTRING(MAIL,1) RLIKE ('[a-zA-Z]{1}')
AND
SUBSTRING(MAIL,2,CHAR_LENGTH(MAIL))
RLIKE ('[a-zA-Z0-9_.-]+@leetcode.com')
```

```
# Learnt From Other Solutions
SELECT USER_ID,NAME, MAIL
FROM USERS
WHERE REGEXP_LIKE(mail ,'^[a-zA-Z][A-Za-z0-9\_\.\_\.\\/-]*@leetcode[.]com')
```

1527. Patients With a Condition ↗

Table: Patients

Column Name	Type
patient_id	int
patient_name	varchar
conditions	varchar

patient_id is the primary key for this table.

'conditions' contains 0 or more code separated by spaces.

This table contains information of the patients in the hospital.

Write an SQL query to report the patient_id, patient_name all conditions of patients who have Type I Diabetes. Type I Diabetes always starts with DIAB1 prefix

Return the result table in **any order**.

The query result format is in the following example.

Example 1:**Input:**

Patients table:

patient_id	patient_name	conditions
1	Daniel	YFEV COUGH
2	Alice	
3	Bob	DIAB100 MYOP
4	George	ACNE DIAB100
5	Alain	DIAB201

Output:

patient_id	patient_name	conditions
3	Bob	DIAB100 MYOP
4	George	ACNE DIAB100

Explanation: Bob and George both have a condition that starts with DIAB1.

- Using 2 Like Operator Conditions in Oracle

```
SELECT *
FROM patients
WHERE CONDITIONS LIKE '% DIAB1%'
OR CONDITIONS LIKE 'DIAB1%'
```

1532. The Most Recent Three Orders ↗

Table: Customers

Column Name	Type
customer_id	int
name	varchar

customer_id is the primary key for this table.
This table contains information about customers.

Table: Orders

Column Name	Type
order_id	int
order_date	date
customer_id	int
cost	int

order_id is the primary key for this table.
This table contains information about the orders made by customer_id.
Each customer has **one order per day**.

Write an SQL query to find the most recent three orders of each user. If a user ordered less than three orders, return all of their orders.

Return the result table ordered by `customer_name` in **ascending order** and in case of a tie by the `customer_id` in **ascending order**. If there is still a tie, order them by `order_date` in **descending order**.

The query result format is in the following example.

Example 1:

Input:

Customers table:

customer_id	name
1	Winston
2	Jonathan
3	Annabelle
4	Marwan
5	Khaled

Orders table:

order_id	order_date	customer_id	cost
1	2020-07-31	1	30
2	2020-07-30	2	40
3	2020-07-31	3	70
4	2020-07-29	4	100
5	2020-06-10	1	1010
6	2020-08-01	2	102
7	2020-08-01	3	111
8	2020-08-03	1	99
9	2020-08-07	2	32
10	2020-07-15	1	2

Output:

customer_name	customer_id	order_id	order_date
Annabelle	3	7	2020-08-01
Annabelle	3	3	2020-07-31
Jonathan	2	9	2020-08-07
Jonathan	2	6	2020-08-01
Jonathan	2	2	2020-07-30
Marwan	4	4	2020-07-29
Winston	1	8	2020-08-03
Winston	1	1	2020-07-31
Winston	1	10	2020-07-15

Explanation:

Winston has 4 orders, we discard the order of "2020-06-10" because it is the oldest
 Annabelle has only 2 orders, we return them.

Jonathan has exactly 3 orders.

Marwan ordered only one time.

We sort the result table by customer_name in ascending order, by customer_id in ascending order.

Follow up: Could you write a general solution for the most recent n orders?

- Didn't Know To Use Rank for Limiting the Data By 3 rows.

```
# This Query Lists all the data Correctly But limiting to
# the 3 most recent orders is not done.

SELECT
C.NAME AS CUSTOMER_NAME,
C.CUSTOMER_ID,
O.ORDER_ID,
O.ORDER_DATE
FROM CUSTOMERS AS C INNER JOIN ORDERS AS O
ON C.CUSTOMER_ID = O.CUSTOMER_ID
ORDER BY CUSTOMER_NAME, CUSTOMER_ID ASC, ORDER_DATE DESC
```

- Rank Vaadalani telisindhi

```
WITH ORDER_SUMMARY AS
(
SELECT
C.NAME AS CUSTOMER_NAME,
C.CUSTOMER_ID,
O.ORDER_ID,
O.ORDER_DATE,
RANK() OVER (PARTITION BY C.CUSTOMER_ID ORDER BY O.ORDER_DATE DESC) AS ORDER_RANK
FROM CUSTOMERS AS C INNER JOIN ORDERS AS O
ON C.CUSTOMER_ID = O.CUSTOMER_ID
ORDER BY CUSTOMER_NAME, CUSTOMER_ID ASC, ORDER_DATE DESC
)

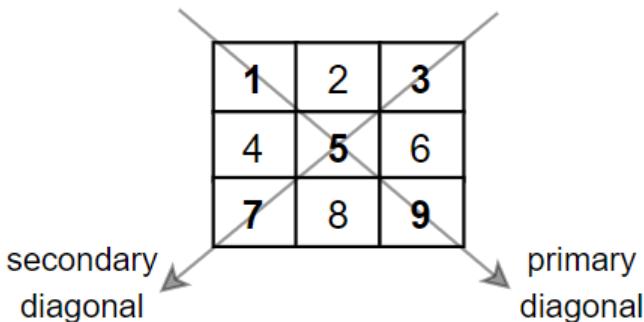
SELECT CUSTOMER_NAME, CUSTOMER_ID, ORDER_ID, ORDER_DATE
FROM ORDER_SUMMARY
WHERE ORDER_RANK < 4
```

1572. Matrix Diagonal Sum

Given a square matrix `mat`, return the sum of the matrix diagonals.

Only include the sum of all the elements on the primary diagonal and all the elements on the secondary diagonal that are not part of the primary diagonal.

Example 1:



Input: mat = [[1,2,3],
[4,5,6],
[7,8,9]]

Output: 25

Explanation: Diagonals sum: 1 + 5 + 9 + 3 + 7 = 25

Notice that element mat[1][1] = 5 is counted only once.

Example 2:

Input: mat = [[1,1,1,1],
[1,1,1,1],
[1,1,1,1],
[1,1,1,1]]

Output: 8

Example 3:

Input: mat = [[5]]

Output: 5

Constraints:

- $n == \text{mat.length} == \text{mat[i].length}$
- $1 \leq n \leq 100$
- $1 \leq \text{mat}[i][j] \leq 100$

```
class Solution:
    # O(N^2) To find the Sum of the Diagonals...
    def diagonalSum(self, mat: List[List[int]]) -> int:
        total = 0
        for i in range(len(mat)):
            for j in range(len(mat)):
                if i == j :
                    total+= mat[i][j]
                elif (j == len(mat)-i-1 and j != i):
                    total+= mat[i][j]
        return total
```

1603. Design Parking System ↗



Design a parking system for a parking lot. The parking lot has three kinds of parking spaces: big, medium, and small, with a fixed number of slots for each size.

Implement the `ParkingSystem` class:

- `ParkingSystem(int big, int medium, int small)` Initializes object of the `ParkingSystem` class. The number of slots for each parking space are given as part of the constructor.
- `bool addCar(int carType)` Checks whether there is a parking space of `carType` for the car that wants to get into the parking lot. `carType` can be of three kinds: big, medium, or small, which are represented by 1, 2, and 3 respectively. **A car can only park in a parking space of its carType**. If there is no space available, return `false`, else park the car in that size space and return `true`.

Example 1:

Input

```
[ "ParkingSystem", "addCar", "addCar", "addCar", "addCar" ]  
[[1, 1, 0], [1], [2], [3], [1]]
```

Output

```
[null, true, true, false, false]
```

Explanation

```
ParkingSystem parkingSystem = new ParkingSystem(1, 1, 0);  
parkingSystem.addCar(1); // return true because there is 1 available slot for a big  
parkingSystem.addCar(2); // return true because there is 1 available slot for a medi  
parkingSystem.addCar(3); // return false because there is no available slot for a sm  
parkingSystem.addCar(1); // return false because there is no available slot for a bi
```



Constraints:

- $0 \leq \text{big}, \text{medium}, \text{small} \leq 1000$
- `carType` is 1, 2, or 3
- At most 1000 calls will be made to `addCar`

```

class ParkingSystem:

    def __init__(self, big: int, medium: int, small: int):
        self.big = big
        self.medium = medium
        self.small = small

    def addCar(self, carType: int) -> bool:
        if carType == 1:
            if self.big == 0:
                return False
            else:
                self.big -= 1
                return True
        elif carType == 2:
            if self.medium == 0:
                return False
            else:
                self.medium -= 1
                return True
        else:
            if self.small == 0:
                return False
            else:
                self.small -= 1
                return True

# Your ParkingSystem object will be instantiated and called as such:
# obj = ParkingSystem(big, medium, small)
# param_1 = obj.addCar(carType)

```

1565. Unique Orders and Customers Per Month ↗ ▾

Table: Orders

Column Name	Type
order_id	int
order_date	date
customer_id	int
invoice	int

order_id is the primary key for this table.

This table contains information about the orders made by customer_id.

Write an SQL query to find the number of **unique orders** and the number of **unique customers** with invoices > **\$20** for each **different month**.

Return the result table sorted in **any order**.

The query result format is in the following example.

Example 1:

Input:

Orders table:

order_id	order_date	customer_id	invoice
1	2020-09-15	1	30
2	2020-09-17	2	90
3	2020-10-06	3	20
4	2020-10-20	3	21
5	2020-11-10	1	10
6	2020-11-21	2	15
7	2020-12-01	4	55
8	2020-12-03	4	77
9	2021-01-07	3	31
10	2021-01-15	2	20

Output:

month	order_count	customer_count
2020-09	2	2
2020-10	1	1
2020-12	2	1
2021-01	1	1

Explanation:

In September 2020 we have two orders from 2 different customers with invoices > \$20.

In October 2020 we have two orders from 1 customer, and only one of the two orders h

In November 2020 we have two orders from 2 different customers but invoices < \$20, s

In December 2020 we have two orders from 1 customer both with invoices > \$20.

In January 2021 we have two orders from 2 different customers, but only one of them



```

SELECT
DATE_FORMAT(ORDER_DATE, '%Y-%m') AS MONTH,
COUNT(ORDER_ID) AS ORDER_COUNT,
COUNT(DISTINCT CUSTOMER_ID) AS CUSTOMER_COUNT
FROM ORDERS
WHERE INVOICE > 20
GROUP BY MONTH(ORDER_DATE), YEAR(ORDER_DATE)

```

1571. Warehouse Manager ↗

Table: Warehouse

Column Name	Type
name	varchar
product_id	int
units	int

(name, product_id) is the primary key for this table.

Each row of this table contains the information of the products in each warehouse.

Table: Products

Column Name	Type
product_id	int
product_name	varchar
Width	int
Length	int
Height	int

product_id is the primary key for this table.

Each row of this table contains information about the product dimensions (Width, Len

Write an SQL query to report the number of cubic feet of **volume** the inventory occupies in each warehouse.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:**Input:**

Warehouse table:

name	product_id	units
LCHouse1	1	1
LCHouse1	2	10
LCHouse1	3	5
LCHouse2	1	2
LCHouse2	2	2
LCHouse3	4	1

Products table:

product_id	product_name	Width	Length	Height
1	LC-TV	5	50	40
2	LC-KeyChain	5	5	5
3	LC-Phone	2	10	10
4	LC-T-Shirt	4	10	20

Output:

warehouse_name	volume
LCHouse1	12250
LCHouse2	20250
LCHouse3	800

Explanation:Volume of product_id = 1 (LC-TV), $5 \times 50 \times 40 = 10000$ Volume of product_id = 2 (LC-KeyChain), $5 \times 5 \times 5 = 125$ Volume of product_id = 3 (LC-Phone), $2 \times 10 \times 10 = 200$ Volume of product_id = 4 (LC-T-Shirt), $4 \times 10 \times 20 = 800$

LCHouse1: 1 unit of LC-TV + 10 units of LC-KeyChain + 5 units of LC-Phone.

Total volume: $1 \times 10000 + 10 \times 125 + 5 \times 200 = 12250$ cubic feet

LCHouse2: 2 units of LC-TV + 2 units of LC-KeyChain.

Total volume: $2 \times 10000 + 2 \times 125 = 20250$ cubic feet

LCHouse3: 1 unit of LC-T-Shirt.

Total volume: $1 \times 800 = 800$ cubic feet.

```

SELECT
W.NAME AS WAREHOUSE_NAME,
SUM((P.WIDTH * P.LENGTH * P.HEIGHT)* W.UNITS) AS VOLUME
FROM PRODUCTS AS P JOIN WAREHOUSE AS W
ON P.PRODUCT_ID = W.PRODUCT_ID
GROUP BY W.NAME

```

1581. Customer Who Visited but Did Not Make Any Transactions ▼

Table: Visits

Column Name	Type
visit_id	int
customer_id	int

visit_id is the primary key for this table.

This table contains information about the customers who visited the mall.

Table: Transactions

Column Name	Type
transaction_id	int
visit_id	int
amount	int

transaction_id is the primary key for this table.

This table contains information about the transactions made during the visit_id.

Write an SQL query to find the IDs of the users who visited without making any transactions and the number of times they made these types of visits.

Return the result table sorted in **any order**.

The query result format is in the following example.

Example 1:

Input:

Visits

visit_id	customer_id
1	23
2	9
4	30
5	54
6	96
7	54
8	54

Transactions

transaction_id	visit_id	amount
2	5	310
3	5	300
9	5	200
12	1	910
13	2	970

Output:

customer_id	count_no_trans
54	2
30	1
96	1

Explanation:

Customer with id = 23 visited the mall once and made one transaction during the visit.

Customer with id = 9 visited the mall once and made one transaction during the visit.

Customer with id = 30 visited the mall once and did not make any transactions.

Customer with id = 54 visited the mall three times. During 2 visits they did not mak

Customer with id = 96 visited the mall once and did not make any transactions.

As we can see, users with IDs 30 and 96 visited the mall one time without making any



```
SELECT V.CUSTOMER_ID, COUNT(*) AS COUNT_NO_TRANS
FROM VISITS AS V LEFT JOIN TRANSACTIONS AS T
ON V.VISIT_ID = T.VISIT_ID
WHERE T.VISIT_ID IS NULL
GROUP BY V.CUSTOMER_ID
```

1587. Bank Account Summary II ↗

Table: Users

Column Name	Type
account	int
name	varchar

account is the primary key for this table.

Each row of this table contains the account number of each user in the bank.

There will be no two users having the same name in the table.

Table: Transactions

Column Name	Type
trans_id	int
account	int
amount	int
transacted_on	date

trans_id is the primary key for this table.

Each row of this table contains all changes made to all accounts.

amount is positive if the user received money and negative if they transferred money

All accounts start with a balance of 0.

Write an SQL query to report the name and balance of users with a balance higher than 10000 . The balance of an account is equal to the sum of the amounts of all transactions involving that account.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Users table:

account	name
900001	Alice
900002	Bob
900003	Charlie

Transactions table:

trans_id	account	amount	transacted_on
1	900001	7000	2020-08-01
2	900001	7000	2020-09-01
3	900001	-3000	2020-09-02
4	900002	1000	2020-09-12
5	900003	6000	2020-08-07
6	900003	6000	2020-09-07
7	900003	-4000	2020-09-11

Output:

name	balance
Alice	11000

Explanation:Alice's balance is $(7000 + 7000 - 3000) = 11000$.

Bob's balance is 1000.

Charlie's balance is $(6000 + 6000 - 4000) = 8000$.

- Sql Solution Using Group BY and Having

```

SELECT
U.NAME,
SUM(T.AMOUNT) AS BALANCE
FROM USERS AS U LEFT JOIN TRANSACTIONS AS T
ON U.ACCT = T.ACCT
GROUP BY U.ACCT
HAVING SUM(T.AMOUNT) > 10000
    
```

1596. The Most Frequently Ordered Products for Each Customer ↗



Table: Customers

Column Name	Type
customer_id	int
name	varchar

customer_id is the primary key for this table.

This table contains information about the customers.

Table: Orders

Column Name	Type
order_id	int
order_date	date
customer_id	int
product_id	int

order_id is the primary key for this table.

This table contains information about the orders made by customer_id.

No customer will order the same product **more than once** in a single day.

Table: Products

Column Name	Type
product_id	int
product_name	varchar
price	int

product_id is the primary key for this table.

This table contains information about the products.

Write an SQL query to find the most frequently ordered product(s) for each customer.

The result table should have the `product_id` and `product_name` for each `customer_id` who ordered at least one order.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Customers table:

customer_id	name
1	Alice
2	Bob
3	Tom
4	Jerry
5	John

Orders table:

order_id	order_date	customer_id	product_id
1	2020-07-31	1	1
2	2020-07-30	2	2
3	2020-08-29	3	3
4	2020-07-29	4	1
5	2020-06-10	1	2
6	2020-08-01	2	1
7	2020-08-01	3	3
8	2020-08-03	1	2
9	2020-08-07	2	3
10	2020-07-15	1	2

Products table:

product_id	product_name	price
1	keyboard	120
2	mouse	80
3	screen	600
4	hard disk	450

Output:

customer_id	product_id	product_name
1	2	mouse
2	1	keyboard
2	2	mouse
2	3	screen
3	3	screen
4	1	keyboard

Explanation:

Alice (customer 1) ordered the mouse three times and the keyboard one time, so the most frequent product for Alice is the mouse.

Bob (customer 2) ordered the keyboard, the mouse, and the screen one time, so those are tied as the most frequent products for Bob.

Tom (customer 3) only ordered the screen (two times), so that is the most frequently ordered product for Tom.

Jerry (customer 4) only ordered the keyboard (one time), so that is the most frequent product.
 John (customer 5) did not order anything, so we do not include them in the result table.

- Solution Using Rank ...

```

SELECT
A.CUSTOMER_ID,
A.PRODUCT_ID,
A.PRODUCT_NAME
FROM
(
SELECT
O.CUSTOMER_ID,
P.PRODUCT_ID,
P.PRODUCT_NAME,
DENSE_RANK() OVER(PARTITION BY O.CUSTOMER_ID ORDER BY COUNT(*) DESC) AS CRANK
FROM ORDERS AS O JOIN PRODUCTS AS P
ON O.PRODUCT_ID = P.PRODUCT_ID
GROUP BY O.CUSTOMER_ID, P.PRODUCT_ID
) A
WHERE A.CRANK = 1
  
```

1636. Sort Array by Increasing Frequency ↗

Given an array of integers `nums`, sort the array in **increasing** order based on the frequency of the values.
 If multiple values have the same frequency, sort them in **decreasing** order.

Return the *sorted array*.

Example 1:

Input: `nums = [1,1,2,2,2,3]`
Output: `[3,1,1,2,2,2]`
Explanation: '3' has a frequency of 1, '1' has a frequency of 2, and '2' has a frequency of 2.

Example 2:

Input: `nums = [2,3,1,3,2]`
Output: `[1,3,3,2,2]`
Explanation: '2' and '3' both have a frequency of 2, so they are sorted in decreasing order.

Example 3:

Input: nums = [-1,1,-6,4,5,-6,1,4,1]
Output: [5,-1,4,4,-6,-6,1,1,1]

Constraints:

- $1 \leq \text{nums.length} \leq 100$
- $-100 \leq \text{nums}[i] \leq 100$

```
class Solution:
    def frequencySort(self, nums: List[int]) -> List[int]:
        # O(N) time to build the Hash Table.
        freq = {}
        for val in nums:
            if val not in freq:
                freq[val] = 1
            else:
                freq[val] += 1

        # Once built we are sorting the data based on frequency Asc
        # And The Value Desc when sorted.
        # So i have used the, lambda function
        # to Solve the problem.
        # This following code takes O(NLogN)
        freq = {k:v
                for k,v in sorted(freq.items(),
                                  key = lambda items: (items[1], 100 - items[0]))}

        nums.clear()
        # building the Array takes O(N)
        for k,v in freq.items():
            val = [k]*v
            nums = nums + val
    return nums
```

1607. Sellers With No Sales ↗

Table: Customer

Column Name	Type
customer_id	int
customer_name	varchar

customer_id is the primary key for this table.

Each row of this table contains the information of each customer in the WebStore.

Table: Orders

Column Name	Type
order_id	int
sale_date	date
order_cost	int
customer_id	int
seller_id	int

order_id is the primary key for this table.

Each row of this table contains all orders made in the webstore.

sale_date is the date when the transaction was made between the customer (customer_i

Table: Seller

Column Name	Type
seller_id	int
seller_name	varchar

seller_id is the primary key for this table.

Each row of this table contains the information of each seller.

Write an SQL query to report the names of all sellers who did not make any sales in 2020 .

Return the result table ordered by seller_name in **ascending order**.

The query result format is in the following example.

Example 1:

Input:

Customer table:

customer_id	customer_name
101	Alice
102	Bob
103	Charlie

Orders table:

order_id	sale_date	order_cost	customer_id	seller_id
1	2020-03-01	1500	101	1
2	2020-05-25	2400	102	2
3	2019-05-25	800	101	3
4	2020-09-13	1000	103	2
5	2019-02-11	700	101	2

Seller table:

seller_id	seller_name
1	Daniel
2	Elizabeth
3	Frank

Output:

seller_name
Frank

Explanation:

Daniel made 1 sale in March 2020.

Elizabeth made 2 sales in 2020 and 1 sale in 2019.

Frank made 1 sale in 2019 but no sales in 2020.

```

SELECT SELLER_NAME
FROM SELLER
WHERE SELLER_ID NOT IN (
SELECT DISTINCT SSELLER_ID
FROM ORDERS AS O JOIN SELLER AS S
ON O.SELLER_ID = S.SELLER_ID
WHERE YEAR(O.SALE_DATE) = 2020
)
ORDER BY SELLER_NAME

```

1623. All Valid Triplets That Can Represent a Country ↴

Table: SchoolA

Column Name	Type
student_id	int
student_name	varchar

student_id is the primary key for this table.

Each row of this table contains the name and the id of a student in school A.

All student_name are distinct.

Table: SchoolB

Column Name	Type
student_id	int
student_name	varchar

student_id is the primary key for this table.

Each row of this table contains the name and the id of a student in school B.

All student_name are distinct.

Table: SchoolC

Column Name	Type
student_id	int
student_name	varchar

student_id is the primary key for this table.

Each row of this table contains the name and the id of a student in school C.

All student_name are distinct.

There is a country with three schools, where each student is enrolled in **exactly one** school. The country is joining a competition and wants to select one student from each school to represent the country such that:

- member_A is selected from SchoolA ,
- member_B is selected from SchoolB ,
- member_C is selected from SchoolC , and
- The selected students' names and IDs are pairwise distinct (i.e. no two students share the same name, and no two students share the same ID).

Write an SQL query to find all the possible triplets representing the country under the given constraints.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

SchoolA table:

student_id	student_name
1	Alice
2	Bob

SchoolB table:

student_id	student_name
3	Tom

SchoolC table:

student_id	student_name
3	Tom
2	Jerry
10	Alice

Output:

member_A	member_B	member_C
Alice	Tom	Jerry
Bob	Tom	Alice

Explanation:

Let us see all the possible triplets.

- (Alice, Tom, Tom) --> Rejected because member_B and member_C have the same name and
- (Alice, Tom, Jerry) --> Valid triplet.
- (Alice, Tom, Alice) --> Rejected because member_A and member_C have the same name.
- (Bob, Tom, Tom) --> Rejected because member_B and member_C have the same name and
- (Bob, Tom, Jerry) --> Rejected because member_A and member_C have the same ID.
- (Bob, Tom, Alice) --> Valid triplet.



```

SELECT
SA.STUDENT_NAME AS MEMBER_A,
SB.STUDENT_NAME AS MEMBER_B ,
SC.STUDENT_NAME AS MEMBER_C
FROM SCHOOLA AS SA JOIN SCHOOLB AS SB
ON (SA.STUDENT_NAME != SB.STUDENT_NAME AND SA.STUDENT_ID != SB.STUDENT_ID) JOIN SC
HOOLC AS SC
ON (
(
    SB.STUDENT_NAME != SC.STUDENT_NAME
    AND SA.STUDENT_NAME != SC.STUDENT_NAME
)
AND
(
    SB.STUDENT_ID != SC.STUDENT_ID
    AND SA.STUDENT_ID != SC.STUDENT_ID
)
)
)

```

1633. Percentage of Users Attended a Contest ↗

Table: Users

Column Name	Type
user_id	int
user_name	varchar

user_id is the primary key for this table.

Each row of this table contains the name and the id of a user.

Table: Register

Column Name	Type
contest_id	int
user_id	int

(contest_id, user_id) is the primary key for this table.

Each row of this table contains the id of a user and the contest they registered into.

Write an SQL query to find the percentage of the users registered in each contest rounded to **two decimals**.

Return the result table ordered by `percentage` in **descending order**. In case of a tie, order it by `contest_id` in **ascending order**.

The query result format is in the following example.

Example 1:

Input:

Users table:

user_id	user_name
6	Alice
2	Bob
7	Alex

Register table:

contest_id	user_id
215	6
209	2
208	2
210	6
208	6
209	7
209	6
215	7
208	7
210	2
207	2
210	7

Output:

contest_id	percentage
208	100.0
209	100.0
210	100.0
215	66.67
207	33.33

Explanation:

All the users registered in contests 208, 209, and 210. The percentage is 100% and we can ignore them.
Alice and Alex registered in contest 215 and the percentage is $((2/3) * 100) = 66.67\%$
Bob registered in contest 207 and the percentage is $((1/3) * 100) = 33.33\%$



```

SELECT
R.CONTEST_ID,
ROUND(COUNT(DISTINCT R.USER_ID)/(SELECT COUNT(*) FROM USERS)*100,2) AS PERCENTAGE
FROM USERS AS U LEFT JOIN REGISTER AS R
ON U.USER_ID = R.USER_ID
WHERE R.USER_ID IS NOT NULL
GROUP BY R.CONTEST_ID
ORDER BY PERCENTAGE DESC, CONTEST_ID ASC

```

1635. Hopper Company Queries I ↗

Table: Drivers

Column Name	Type
driver_id	int
join_date	date

driver_id is the primary key for this table.

Each row of this table contains the driver's ID and the date they joined the Hopper

◀ ▶

Table: Rides

Column Name	Type
ride_id	int
user_id	int
requested_at	date

ride_id is the primary key for this table.

Each row of this table contains the ID of a ride, the user's ID that requested it, a
There may be some ride requests in this table that were not accepted.

◀ ▶

Table: AcceptedRides

Column Name	Type
ride_id	int
driver_id	int
ride_distance	int
ride_duration	int

ride_id is the primary key for this table.

Each row of this table contains some information about an accepted ride.

It is guaranteed that each accepted ride exists in the Rides table.

Write an SQL query to report the following statistics for each month of **2020**:

- The number of drivers currently with the Hopper company by the end of the month (`active_drivers`).
- The number of accepted rides in that month (`accepted_rides`).

Return the result table ordered by `month` in ascending order, where `month` is the month's number (January is `1`, February is `2`, etc.).

The query result format is in the following example.

Example 1:

Input:

Drivers table:

driver_id	join_date
10	2019-12-10
8	2020-1-13
5	2020-2-16
7	2020-3-8
4	2020-5-17
1	2020-10-24
6	2021-1-5

Rides table:

ride_id	user_id	requested_at
6	75	2019-12-9
1	54	2020-2-9
10	63	2020-3-4
19	39	2020-4-6
3	41	2020-6-3
13	52	2020-6-22
7	69	2020-7-16
17	70	2020-8-25
20	81	2020-11-2
5	57	2020-11-9
2	42	2020-12-9
11	68	2021-1-11
15	32	2021-1-17
12	11	2021-1-19
14	18	2021-1-27

AcceptedRides table:

ride_id	driver_id	ride_distance	ride_duration
10	10	63	38
13	10	73	96
7	8	100	28
17	7	119	68
20	1	121	92
5	7	42	101
2	4	6	38
11	8	37	43
15	8	108	82
12	8	38	34
14	1	90	74

Output:

ride_id	driver_id	ride_distance	ride_duration
6	75	2019-12-9	38
1	54	2020-2-9	96
10	63	2020-3-4	28
19	39	2020-4-6	68
3	41	2020-6-3	92
13	52	2020-6-22	101
7	69	2020-7-16	34
17	70	2020-8-25	74
20	81	2020-11-2	43
5	57	2020-11-9	82
2	42	2020-12-9	38
11	68	2021-1-11	119
15	32	2021-1-17	121
12	11	2021-1-19	42
14	18	2021-1-27	108

month	active_drivers	accepted_rides
1	2	0
2	3	0
3	4	1
4	4	0
5	5	0
6	5	1
7	5	1
8	5	1
9	5	0
10	6	0
11	6	2
12	6	1

Explanation:

By the end of January --> two active drivers (10, 8) and no accepted rides.

By the end of February --> three active drivers (10, 8, 5) and no accepted rides.

By the end of March --> four active drivers (10, 8, 5, 7) and one accepted ride (10)

By the end of April --> four active drivers (10, 8, 5, 7) and no accepted rides.

By the end of May --> five active drivers (10, 8, 5, 7, 4) and no accepted rides.

By the end of June --> five active drivers (10, 8, 5, 7, 4) and one accepted ride (1)

By the end of July --> five active drivers (10, 8, 5, 7, 4) and one accepted ride (7)

By the end of August --> five active drivers (10, 8, 5, 7, 4) and one accepted ride

By the end of September --> five active drivers (10, 8, 5, 7, 4) and no accepted ride

By the end of October --> six active drivers (10, 8, 5, 7, 4, 1) and no accepted ride

By the end of November --> six active drivers (10, 8, 5, 7, 4, 1) and two accepted ride

By the end of December --> six active drivers (10, 8, 5, 7, 4, 1) and one accepted ride



```
WITH END_OF_MONTH AS
(
    SELECT
        JOINED_YEAR,
        MONTH,
        SUM(MONTHLY_TOTAL) OVER(ORDER BY JOINED_YEAR, MONTH) AS TOTAL RIDERS
    FROM
    (
        SELECT
            YEAR(JOIN_DATE) AS JOINED_YEAR,
            MONTH(JOIN_DATE) AS MONTH,
            COUNT(*) OVER(PARTITION BY MONTH(JOIN_DATE)),
            YEAR(JOIN_DATE)) AS MONTHLY_TOTAL
        FROM DRIVERS
    ) AS A
    ORDER BY JOINED_YEAR, MONTH
)
,
ACCEPTED RIDES AS
(
    SELECT MONTH(R.REQUESTED_AT) AS MONTH , COUNT(*) AS ACCEPTED RIDES
    FROM RIDES AS R JOIN ACCEPTEDRIDES AS AR
    ON R.RIDE_ID = AR.RIDE_ID
    WHERE YEAR(R.REQUESTED_AT) = 2020
    GROUP BY MONTH(R.REQUESTED_AT)
)
SELECT * FROM END_OF_MONTH
```

- Final Working and Code Which Passes all the Test Cases.

```
WITH MONTHS AS
(
    SELECT 1 AS MONTH, 2020 AS YEAR
    UNION
    SELECT 2 AS MONTH, 2020 AS YEAR
    UNION
    SELECT 3 AS MONTH, 2020 AS YEAR
    UNION
    SELECT 4 AS MONTH, 2020 AS YEAR
    UNION
    SELECT 5 AS MONTH, 2020 AS YEAR
    UNION
    SELECT 6 AS MONTH, 2020 AS YEAR
    UNION
    SELECT 7 AS MONTH, 2020 AS YEAR
    UNION
    SELECT 8 AS MONTH, 2020 AS YEAR
    UNION
    SELECT 9 AS MONTH, 2020 AS YEAR
    UNION
    SELECT 10 AS MONTH, 2020 AS YEAR
    UNION
    SELECT 11 AS MONTH, 2020 AS YEAR
    UNION
    SELECT 12 AS MONTH, 2020 AS YEAR
),
ACTIVE_DRIVERS AS
(
    SELECT
        DISTINCT
        D.MONTH,
        FIRST_VALUE(D.TOTAL RIDERS)
        OVER(PARTITION BY D.TR
            ORDER BY D.MONTH
            ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
        AS TOTAL RIDERS
        FROM
        (
            SELECT
                M.MONTH,
                CASE
                    WHEN C.TOTAL RIDERS IS NULL THEN 0
                    ELSE C.TOTAL RIDERS
                END AS TOTAL RIDERS,
                SUM(CASE WHEN C.TOTAL RIDERS IS NULL THEN 0 ELSE 1 END)
                OVER(ORDER BY M.MONTH) AS TR
                FROM
                (
                    SELECT
                        *
```

```

        FROM
        (
            SELECT
            DISTINCT
            A.JOINED_YEAR,
            A.MONTH,
            SUM(A.MONTHLY_TOTAL)
            OVER(ORDER BY A.JOINED_YEAR, A.MONTH) AS TOTAL
    _RIDERS
        FROM
        (
            SELECT DISTINCT
            YEAR(JOIN_DATE) AS JOINED_YEAR,
            MONTH(JOIN_DATE) AS MONTH,
            COUNT(*)
            OVER
            (
                PARTITION BY
                MONTH(JOIN_DATE),YEAR(JOIN_DATE)
            )
            AS MONTHLY_TOTAL
            FROM DRIVERS
        ) AS A
        WHERE A.JOINED_YEAR IN (2019,2020)

        ) AS B
        WHERE B.JOINED_YEAR = 2020
    ) AS C RIGHT JOIN MONTHS AS M
    ON C.MONTH = M.MONTH

    ) AS D
)
,
ACCEPTED RIDES AS
(
    SELECT MONTH(R.REQUESTED_AT) AS MONTH , COUNT(*) AS ACCEPTED RIDES
    FROM RIDES AS R JOIN ACCEPTEDRIDES AS AR
    ON R.RIDE_ID = AR.RIDE_ID
    WHERE YEAR(R.REQUESTED_AT) = 2020
    GROUP BY MONTH(R.REQUESTED_AT)
),
ACCEPTED_RIDE_RESULT AS
(
    SELECT M.MONTH,
    CASE
        WHEN AR.ACCEPTED_RIDES IS NOT NULL THEN AR.ACCEPTED_RIDES
        ELSE 0
    END AS ACCEPTED_RIDES
    FROM MONTHS AS M LEFT JOIN ACCEPTED RIDES AS AR

```

```

    ON AR.MONTH = M.MONTH
)
SELECT
AD.MONTH,
AD.TOTAL RIDERS AS ACTIVE_DRIVERS,
ARR.ACCEPTED RIDES
FROM ACTIVE_DRIVERS AS AD JOIN ACCEPTED_RIDE_RESULT ARR
ON AD.MONTH = ARR.MONTH
ORDER BY AD.MONTH

```

1672. Richest Customer Wealth ↗

You are given an $m \times n$ integer grid `accounts` where `accounts[i][j]` is the amount of money the i^{th} customer has in the j^{th} bank. Return *the wealth that the richest customer has*.

A customer's **wealth** is the amount of money they have in all their bank accounts. The richest customer is the customer that has the maximum **wealth**.

Example 1:

Input: `accounts = [[1,2,3],[3,2,1]]`
Output: 6
Explanation:
1st customer has wealth = $1 + 2 + 3 = 6$
2nd customer has wealth = $3 + 2 + 1 = 6$
Both customers are considered the richest with a wealth of 6 each, so return 6.

Example 2:

Input: `accounts = [[1,5],[7,3],[3,5]]`
Output: 10
Explanation:
1st customer has wealth = 6
2nd customer has wealth = 10
3rd customer has wealth = 8
The 2nd customer is the richest with a wealth of 10.

Example 3:

Input: `accounts = [[2,8,7],[7,1,3],[1,9,5]]`
Output: 17

Constraints:

- $m == \text{accounts.length}$
- $n == \text{accounts}[i].length$
- $1 \leq m, n \leq 50$
- $1 \leq \text{accounts}[i][j] \leq 100$

```
# O(N) * O(N) => O(N**2)
class Solution:
    def maximumWealth(self, accounts: List[List[int]]) -> int:
        amt = -float('inf')
        for i in range(len(accounts)):
            amt_in_bank = sum(accounts[i])
            if amt < amt_in_bank:
                amt = amt_in_bank

        return amt
```

1678. Goal Parser Interpretation

You own a **Goal Parser** that can interpret a string `command`. The `command` consists of an alphabet of "G", "()" and/or "(al)" in some order. The Goal Parser will interpret "G" as the string "G", "()" as the string "o", and "(al)" as the string "al". The interpreted strings are then concatenated in the original order.

Given the string `command`, return *the Goal Parser's interpretation of command*.

Example 1:

Input: command = "G()()al"
Output: "Goal"
Explanation: The Goal Parser interprets the command as follows:
 $G \rightarrow G$
 $() \rightarrow o$
 $(al) \rightarrow al$
The final concatenated result is "Goal".

Example 2:

Input: command = "G()()()()()al"
Output: "Gooooal"

Example 3:

Input: command = "(al)G(al)()()G"

Output: "alGalooG"

Constraints:

- $1 \leq \text{command.length} \leq 100$
- command consists of "G" , "()" , and/or "(al)" in some order.

```
class Solution:
    def interpret(self, command: str) -> str:
        stk = []
        res = ""
        for ind in range(len(command)):

            # if the parathesis closes

            if command[ind] == ')':
                if len(stk) > 0 and stk[-1] == '(':
                    # if the n-2 index value is character then pop the
                    # value from stk
                    if ord(command[ind-1]) >= 97 and ord(command[ind-1]) <= 122:
                        stk.pop()
                    # if n-2 index value is not a character then
                    else:
                        res += 'o'
                        stk.pop()

            # if it is opened then push to stack

            elif command[ind] == '(':
                stk.append(command[ind])
            else:
                res += command[ind]
        return res
```

1661. Average Time of Process per Machine ↗

Table: Activity

Column Name	Type
machine_id	int
process_id	int
activity_type	enum
timestamp	float

The table shows the user activities for a factory website.

(`machine_id`, `process_id`, `activity_type`) is the primary key of this table.

`machine_id` is the ID of a machine.

`process_id` is the ID of a process running on the machine with ID `machine_id`.

`activity_type` is an ENUM of type ('start', 'end').

`timestamp` is a float representing the current time in seconds.

'start' means the machine starts the process at the given timestamp and 'end' means

The 'start' timestamp will always be before the 'end' timestamp for every (`machine_i`

There is a factory website that has several machines each running the **same number of processes**. Write an SQL query to find the **average time** each machine takes to complete a process.

The time to complete a process is the 'end' timestamp minus the 'start' timestamp. The average time is calculated by the total time to complete every process on the machine divided by the number of processes that were run.

The resulting table should have the `machine_id` along with the **average time** as `processing_time`, which should be **rounded to 3 decimal places**.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Activity table:

machine_id	process_id	activity_type	timestamp
0	0	start	0.712
0	0	end	1.520
0	1	start	3.140
0	1	end	4.120
1	0	start	0.550
1	0	end	1.550
1	1	start	0.430
1	1	end	1.420
2	0	start	4.100
2	0	end	4.512
2	1	start	2.500
2	1	end	5.000

Output:

machine_id	processing_time
0	0.894
1	0.995
2	1.456

Explanation:

There are 3 machines running 2 processes each.

Machine 0's average time is $((1.520 - 0.712) + (4.120 - 3.140)) / 2 = 0.894$

Machine 1's average time is $((1.550 - 0.550) + (1.420 - 0.430)) / 2 = 0.995$

Machine 2's average time is $((4.512 - 4.100) + (5.000 - 2.500)) / 2 = 1.456$

```

WITH CTE AS (
    SELECT
        MACHINE_ID,
        PROCESS_ID,
        TIMESTAMP AS START_TIME,
        NULL AS END_TIME
    FROM ACTIVITY
    WHERE ACTIVITY_TYPE = 'start'
    GROUP BY MACHINE_ID, PROCESS_ID

    UNION ALL

    SELECT
        MACHINE_ID,
        PROCESS_ID,
        NULL AS START_TIME,
        TIMESTAMP AS END_TIME
    FROM ACTIVITY
    WHERE ACTIVITY_TYPE = 'end'
    GROUP BY MACHINE_ID, PROCESS_ID
)

SELECT MACHINE_ID,
ROUND
(
    (SUM(END_TIME)-SUM(START_TIME)) /
    COUNT(DISTINCT PROCESS_ID), 3
) AS PROCESSING_TIME
FROM CTE
GROUP BY MACHINE_ID

```

1667. Fix Names in a Table ↗

Table: Users

Column Name	Type
user_id	int
name	varchar

user_id is the primary key for this table.

This table contains the ID and the name of the user. The name consists of only lower

Write an SQL query to fix the names so that only the first character is uppercase and the rest are lowercase.

Return the result table ordered by `user_id`.

The query result format is in the following example.

Example 1:

Input:

Users table:

user_id	name
1	aLice
2	bOB

Output:

user_id	name
1	Alice
2	Bob

- Mysql Version of Solution
- Using Concat Function

```
select user_id, concat(upper(substr(name,1,1)),lower(substr(name,2,length(name)))) ) as name
from users
order by user_id
```

- Oracle Version of Solution
- Using Pipe Operator

```
SELECT USER_ID,
UPPER(SUBSTR(NAME,1,1)) || LOWER(SUBSTR(NAME,2,LENGTH(NAME))) AS NAME
FROM USERS
ORDER BY USER_ID
```

1677. Product's Worth Over Invoices ↗

Table: Product

Column Name	Type
product_id	int
name	varchar

product_id is the primary key for this table.

This table contains the ID and the name of the product. The name consists of only lowercase English letters.

Table: Invoice

Column Name	Type
invoice_id	int
product_id	int
rest	int
paid	int
canceled	int
refunded	int

invoice_id is the primary key for this table and the id of this invoice.

product_id is the id of the product for this invoice.

rest is the amount left to pay for this invoice.

paid is the amount paid for this invoice.

canceled is the amount canceled for this invoice.

refunded is the amount refunded for this invoice.

Write an SQL query that will, for all products, return each product name with the total amount due, paid, canceled, and refunded across all invoices.

Return the result table ordered by product_name .

The query result format is in the following example.

Example 1:

Input:

Product table:

product_id	name
0	ham
1	bacon

Invoice table:

invoice_id	product_id	rest	paid	canceled	refunded	
23	0	2	0	5	0	
12	0	0	4	0	3	
1	1	1	1	0	1	
2	1	1	0	1	1	
3	1	0	1	1	1	
4	1	1	1	1	0	

Output:

name	rest	paid	canceled	refunded	
bacon	3	3	3	3	
ham	2	4	5	3	

Explanation:

- The amount of money left to pay for bacon is $1 + 1 + 0 + 1 = 3$
- The amount of money paid for bacon is $1 + 0 + 1 + 1 = 3$
- The amount of money canceled for bacon is $0 + 1 + 1 + 1 = 3$
- The amount of money refunded for bacon is $1 + 1 + 1 + 0 = 3$
- The amount of money left to pay for ham is $2 + 0 = 2$
- The amount of money paid for ham is $0 + 4 = 4$
- The amount of money canceled for ham is $5 + 0 = 5$
- The amount of money refunded for ham is $0 + 3 = 3$

```

SELECT
P.NAME,
CASE WHEN SUM(I.REST) IS NOT NULL THEN SUM(I.REST)
     ELSE 0
END
AS REST,
CASE
    WHEN SUM(I.PAID) IS NOT NULL THEN SUM(I.PAID)
    ELSE 0
END AS PAID,
CASE
    WHEN SUM(I.CANCELED) IS NOT NULL THEN SUM(I.CANCELED)
    ELSE 0
END AS CANCELED,
CASE
    WHEN SUM(I.REFUNDED) IS NOT NULL THEN SUM(I.REFUNDED)
    ELSE 0
END AS REFUNDED
FROM INVOICE AS I RIGHT JOIN PRODUCT AS P
ON I.PRODUCT_ID = P.PRODUCT_ID
GROUP BY P.NAME
ORDER BY NAME

```

1683. Invalid Tweets ↗

Table: Tweets

Column Name	Type
tweet_id	int
content	varchar

tweet_id is the primary key for this table.
This table contains all the tweets in a social media app.

Write an SQL query to find the IDs of the invalid tweets. The tweet is invalid if the number of characters used in the content of the tweet is **strictly greater** than 15 .

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Tweets table:

tweet_id	content
1	Vote for Biden
2	Let us make America great again!

Output:

tweet_id
2

Explanation:

Tweet 1 has length = 14. It is a valid tweet.

Tweet 2 has length = 32. It is an invalid tweet.

- MySql Solution to Find the Invalid Tweets
- But As Mentioned By one of the OP in comments section LENGTH Gives The Space Occupied by the string whereas CHAR_LENGTH strictly gives us no of characters.

```
SELECT
TWEET_ID
FROM TWEETS
WHERE LENGTH(CONTENT) > 15
```

1693. Daily Leads and Partners ↗



Table: DailySales

Column Name	Type
date_id	date
make_name	varchar
lead_id	int
partner_id	int

This table does not have a primary key.

This table contains the date and the name of the product sold and the IDs of the lead. The name consists of only lowercase English letters.

Write an SQL query that will, for each `date_id` and `make_name`, return the number of **`distinct`** `lead_id`'s and **`distinct`** `partner_id`'s.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

DailySales table:

date_id	make_name	lead_id	partner_id
2020-12-8	toyota	0	1
2020-12-8	toyota	1	0
2020-12-8	toyota	1	2
2020-12-7	toyota	0	2
2020-12-7	toyota	0	1
2020-12-8	honda	1	2
2020-12-8	honda	2	1
2020-12-7	honda	0	1
2020-12-7	honda	1	2
2020-12-7	honda	2	1

Output:

date_id	make_name	unique_leads	unique_partners
2020-12-8	toyota	2	3
2020-12-7	toyota	1	2
2020-12-8	honda	2	2
2020-12-7	honda	3	2

Explanation:

For 2020-12-8, toyota gets leads = [0, 1] and partners = [0, 1, 2] while honda gets leads = [1, 2].
 For 2020-12-7, toyota gets leads = [0] and partners = [1, 2] while honda gets leads = [1, 2].



- MYSQL Solution Using Group By And Distinct Count

```

SELECT
DATE_ID,
MAKE_NAME,
COUNT(DISTINCT LEAD_ID) AS UNIQUE_LEADS,
COUNT(DISTINCT PARTNER_ID) AS UNIQUE_PARTNERS

FROM DAILYSALES

GROUP BY
DATE_ID,
MAKE_NAME

```

1699. Number of Calls Between Two Persons ↗



Table: Calls

Column Name	Type
from_id	int
to_id	int
duration	int

This table does not have a primary key, it may contain duplicates.

This table contains the duration of a phone call between from_id and to_id.

from_id != to_id

Write an SQL query to report the number of calls and the total call duration between each pair of distinct persons (person1, person2) where person1 < person2 .

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Calls table:

from_id	to_id	duration
1	2	59
2	1	11
1	3	20
3	4	100
3	4	200
3	4	200
4	3	499

Output:

person1	person2	call_count	total_duration
1	2	2	70
1	3	1	20
3	4	4	999

Explanation:

Users 1 and 2 had 2 calls and the total duration is 70 (59 + 11).

Users 1 and 3 had 1 call and the total duration is 20.

Users 3 and 4 had 4 calls and the total duration is 999 (100 + 200 + 200 + 499).

```

SELECT
A.FROM_ID AS PERSON1,
A.TO_ID AS PERSON2,
SUM(A.CALL_COUNT) AS CALL_COUNT,
SUM(A.TOTAL_DURATION) AS TOTAL_DURATION
FROM
(
SELECT
FROM_ID,
TO_ID,
COUNT(*) AS CALL_COUNT,
SUM(DURATION) AS TOTAL_DURATION
FROM CALLS
GROUP BY FROM_ID, TO_ID

UNION

SELECT
TO_ID,
FROM_ID,
COUNT(*) AS CALL_COUNT,
SUM(DURATION) AS TOTAL_DURATION
FROM CALLS
GROUP BY FROM_ID, TO_ID
) A
WHERE A.FROM_ID < A.TO_ID
GROUP BY A.FROM_ID, A.TO_ID

```

1715. Count Apples and Oranges ↗



Table: Boxes

Column Name	Type
box_id	int
chest_id	int
apple_count	int
orange_count	int

box_id is the primary key for this table.

chest_id is a foreign key of the chests table.

This table contains information about the boxes and the number of oranges and apples

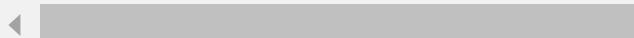


Table: Chests

Column Name	Type
chest_id	int
apple_count	int
orange_count	int

chest_id is the primary key for this table.

This table contains information about the chests and the corresponding number of ora

Write an SQL query to count the number of apples and oranges in all the boxes. If a box contains a chest, you should also include the number of apples and oranges it has.

The query result format is in the following example.

Example 1:

Input:

Boxes table:

box_id	chest_id	apple_count	orange_count	
2	null	6	15	
18	14	4	15	
19	3	8	4	
12	2	19	20	
20	6	12	9	
8	6	9	9	
3	14	16	7	

Chests table:

chest_id	apple_count	orange_count	
6	5	6	
14	20	10	
2	8	8	
3	19	4	
16	19	19	

Output:

apple_count	orange_count	
151	123	

Explanation:

box 2 has 6 apples and 15 oranges.

box 18 has $4 + 20$ (from the chest) = 24 apples and $15 + 10$ (from the chest) = 25 orangebox 19 has $8 + 19$ (from the chest) = 27 apples and $4 + 4$ (from the chest) = 8 orangebox 12 has $19 + 8$ (from the chest) = 27 apples and $20 + 8$ (from the chest) = 28 orangebox 20 has $12 + 5$ (from the chest) = 17 apples and $9 + 6$ (from the chest) = 15 orangesbox 8 has $9 + 5$ (from the chest) = 14 apples and $9 + 6$ (from the chest) = 15 orangesbox 3 has $16 + 20$ (from the chest) = 36 apples and $7 + 10$ (from the chest) = 17 orangesTotal number of apples = $6 + 24 + 27 + 27 + 17 + 14 + 36 = 151$ Total number of oranges = $15 + 25 + 8 + 28 + 15 + 15 + 17 = 123$ 

```

SELECT
SUM(APPLE_COUNT) AS APPLE_COUNT,
SUM(ORANGE_COUNT) AS ORANGE_COUNT
FROM
(
    SELECT
        SUM(B.ORANGE_COUNT) + IFNULL(SUM(C.ORANGE_COUNT),0) AS ORANGE_COUNT,
        SUM(B.APPLE_COUNT) + IFNULL(SUM(C.APPLE_COUNT),0) AS APPLE_COUNT
    FROM BOXES AS B LEFT JOIN CHESTS AS C
    ON B.CHEST_ID = C.CHEST_ID
    GROUP BY B.CHEST_ID
) A

```

1729. Find Followers Count ↗

Table: Followers

Column Name	Type
user_id	int
follower_id	int

(user_id, follower_id) is the primary key for this table.
This table contains the IDs of a user and a follower in a social media app where the

Write an SQL query that will, for each user, return the number of followers.

Return the result table ordered by `user_id`.

The query result format is in the following example.

Example 1:

Input:

Followers table:

user_id	follower_id
0	1
1	0
2	0
2	1

Output:

user_id	followers_count
0	1
1	1
2	2

Explanation:

The followers of 0 are {1}

The followers of 1 are {0}

The followers of 2 are {0,1}

- MYSQL SOLUTION using Group By and Count Functions

```
SELECT
USER_ID,
COUNT(FOLLOWER_ID) AS FOLLOWERS_COUNT
FROM FOLLOWERS
GROUP BY USER_ID
ORDER BY USER_ID
```

1731. The Number of Employees Which Report to Each Employee ↴

Table: Employees

Column Name	Type
employee_id	int
name	varchar
reports_to	int
age	int

employee_id is the primary key for this table.

This table contains information about the employees and the id of the manager they report directly to.

For this problem, we will consider a **manager** an employee who has at least 1 other employee reporting to them.

Write an SQL query to report the ids and the names of all **managers**, the number of employees who report **directly** to them, and the average age of the reports rounded to the nearest integer.

Return the result table ordered by `employee_id`.

The query result format is in the following example.

Example 1:

Input:

Employees table:

employee_id	name	reports_to	age
9	Hercy	null	43
6	Alice	9	41
4	Bob	9	36
2	Winston	null	37

Output:

employee_id	name	reports_count	average_age
9	Hercy	2	39

Explanation: Hercy has 2 people report directly to him, Alice and Bob. Their average age is (41 + 36) / 2 = 39.

```

SELECT
E.EMPLOYEE_ID,
E.NAME,
COUNT(*) AS REPORTS_COUNT,
CASE
    WHEN
        ABS(SUM(EE.AGE)/COUNT(*)) - FLOOR(ABS(SUM(EE.AGE)/COUNT(*))) >= 0.5
    THEN CEILING(SUM(EE.AGE)/COUNT(*))
    ELSE FLOOR(SUM(EE.AGE)/COUNT(*))
END AS AVERAGE_AGE
FROM EMPLOYEES AS E JOIN EMPLOYEES AS EE
ON E.EMPLOYEE_ID = EE.REPORTS_TO
GROUP BY E.EMPLOYEE_ID, E.NAME
HAVING AVERAGE_AGE IS NOT NULL
ORDER BY E.EMPLOYEE_ID

```

1779. Find Nearest Point That Has the Same X or Y Coordinate ↗

You are given two integers, x and y , which represent your current location on a Cartesian grid: (x, y) . You are also given an array `points` where each `points[i] = [ai, bi]` represents that a point exists at (a_i, b_i) . A point is **valid** if it shares the same x-coordinate or the same y-coordinate as your location.

Return the index (**0-indexed**) of the **valid** point with the smallest **Manhattan distance** from your current location. If there are multiple, return the valid point with the **smallest** index. If there are no valid points, return `-1`.

The **Manhattan distance** between two points (x_1, y_1) and (x_2, y_2) is $\text{abs}(x_1 - x_2) + \text{abs}(y_1 - y_2)$.

Example 1:

Input: $x = 3$, $y = 4$, `points = [[1,2],[3,1],[2,4],[2,3],[4,4]]`

Output: 2

Explanation: Of all the points, only $[3,1]$, $[2,4]$ and $[4,4]$ are valid. Of the valid

◀ ➡

Example 2:

Input: $x = 3$, $y = 4$, `points = [[3,4]]`

Output: 0

Explanation: The answer is allowed to be on the same location as your current locati

◀ ➡

Example 3:

Input: x = 3, y = 4, points = [[2,3]]
Output: -1
Explanation: There are no valid points.

Constraints:

- $1 \leq \text{points.length} \leq 10^4$
- $\text{points[i].length} == 2$
- $1 \leq x, y, a_i, b_i \leq 10^4$

```
class Solution:
    def nearestValidPoint(self, x: int, y: int, points: List[List[int]]) -> int:
        res = []
        for i in range(len(points)):
            if x == points[i][0] or y == points[i][1]:
                res.append((i, abs(x-points[i][0]) + abs(y-points[i][1])))
        res = sorted(res, key = lambda x: x[1])
        if len(res) > 0:
            return res[0][0]
        else:
            return -1
```

1741. Find Total Time Spent by Each Employee

Table: Employees

Column Name	Type
emp_id	int
event_day	date
in_time	int
out_time	int

(emp_id, event_day, in_time) is the primary key of this table.
The table shows the employees' entries and exits in an office.
event_day is the day at which this event happened, in_time is the minute at which the in_time and out_time are between 1 and 1440.
It is guaranteed that no two events on the same day intersect in time, and in_time <

Write an SQL query to calculate the total time **in minutes** spent by each employee on each day at the office. Note that within one day, an employee can enter and leave more than once. The time spent in the office for a single entry is `out_time - in_time`.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Employees table:

emp_id	event_day	in_time	out_time
1	2020-11-28	4	32
1	2020-11-28	55	200
1	2020-12-03	1	42
2	2020-11-28	3	33
2	2020-12-09	47	74

Output:

day	emp_id	total_time
2020-11-28	1	173
2020-11-28	2	30
2020-12-03	1	41
2020-12-09	2	27

Explanation:

Employee 1 has three events: two on day 2020-11-28 with a total of $(32 - 4) + (200 - 55) = 173$ minutes.
 Employee 2 has two events: one on day 2020-11-28 with a total of $(33 - 3) = 30$, and one on day 2020-12-09 with a total of $(74 - 47) = 27$ minutes.

- Using Group by And Subtract Functions to Arrive at the Solution .

```
SELECT
EVENT_DAY AS DAY,
EMP_ID,
SUM(OUT_TIME - IN_TIME) AS TOTAL_TIME
FROM EMPLOYEES
GROUP BY EVENT_DAY, EMP_ID
```

1768. Merge Strings Alternately ↗



You are given two strings `word1` and `word2`. Merge the strings by adding letters in alternating order, starting with `word1`. If a string is longer than the other, append the additional letters onto the end of the merged string.

Return *the merged string*.

Example 1:

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

Example 2:

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

Example 3:

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

Constraints:

- $1 \leq \text{word1.length}, \text{word2.length} \leq 100$
- `word1` and `word2` consist of lowercase English letters.

```

class Solution:
    def mergeAlternately(self, word1: str, word2: str) -> str:
        max_len = max(len(word1), len(word2))
        res = ""
        if len(word1) == max_len:
            word2 = list(word2)
            for i in range(max_len):
                if len(word2) != 0:
                    res += word1[i]
                    res += word2.pop(0)
                else:
                    res += word1[i]
        else:
            word1 = list(word1)
            for i in range(max_len):
                if len(word1) > 0:
                    res += word1.pop(0)
                    res += word2[i]
                else:
                    res += word2[i]

        return res

```

1757. Recyclable and Low Fat Products ↗

Table: Products

Column Name	Type
product_id	int
low_fats	enum
recyclable	enum

product_id is the primary key for this table.

low_fats is an ENUM of type ('Y', 'N') where 'Y' means this product is low fat and 'recyclable' is an ENUM of types ('Y', 'N') where 'Y' means this product is recyclable

Write an SQL query to find the ids of products that are both low fat and recyclable.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:**Input:**

Products table:

product_id	low_fats	recyclable
0	Y	N
1	Y	Y
2	N	Y
3	Y	Y
4	N	N

Output:

product_id
1
3

Explanation: Only products 1 and 3 are both low fat and recyclable.

- The Question Simply Asks us to find out which contains both LOW_FAT and RECYLABLE, product_ids.
- This a quite normal one where i need to apply both conditions to fetch the data.

```
select product_id
from products
where low_fats = 'Y' and recyclable = 'Y'
```

1790. Check if One String Swap Can Make Strings Equal ↴

You are given two strings `s1` and `s2` of equal length. A **string swap** is an operation where you choose two indices in a string (not necessarily different) and swap the characters at these indices.

Return `true` if it is possible to make both strings equal by performing **at most one string swap** on **exactly one** of the strings. Otherwise, return `false`.

Example 1:

Input: s1 = "bank", s2 = "kanb"

Output: true

Explanation: For example, swap the first character with the last character of s2 to

Example 2:

Input: s1 = "attack", s2 = "defend"

Output: false

Explanation: It is impossible to make them equal with one string swap.

Example 3:

Input: s1 = "kelb", s2 = "kelb"

Output: true

Explanation: The two strings are already equal, so no string swap operation is required.

Constraints:

- $1 \leq s1.length, s2.length \leq 100$
- $s1.length == s2.length$
- $s1$ and $s2$ consist of only lowercase English letters.

Initial Solution

```
# Fails at Last Test Case.
class Solution:
    def areAlmostEqual(self, s1: str, s2: str) -> bool:
        count = 0
        if set(s1) == set(s2):
            for i in range(len(s1)):
                if s1[i] != s2[i]:
                    count += 1
            print(count)
            if count > 2:
                return False
            else:
                return True
        else:
            return False
```

```

class Solution:
    def areAlmostEqual(self, s1: str, s2: str) -> bool:
        count = 0
        d1, d2 = {}, {}
        for i in range(len(s1)):

            if s1[i] not in d1:
                d1[s1[i]] = 1

            else:
                d1[s1[i]] += 1

            if s2[i] not in d2:
                d2[s2[i]] = 1

            else:
                d2[s2[i]] += 1

        if d1 == d2:
            for i in range(len(s1)):
                if s1[i] != s2[i]:
                    count += 1

        return False if count > 2 else True

    else:
        return False
    ...

```

1777. Product's Price for Each Store ↗

Table: Products

Column Name	Type
product_id	int
store	enum
price	int

(product_id, store) is the primary key for this table.

store is an ENUM of type ('store1', 'store2', 'store3') where each represents the store.
price is the price of the product at this store.

Write an SQL query to find the price of each product in each store.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Products table:

product_id	store	price
0	store1	95
0	store3	105
0	store2	100
1	store1	70
1	store3	80

Output:

product_id	store1	store2	store3
0	95	100	105
1	70	null	80

Explanation:

Product 0 price's are 95 for store1, 100 for store2 and, 105 for store3.

Product 1 price's are 70 for store1, 80 for store3 and, it's not sold in store2.

- The Reason for useng max over here to avoid wrong answer after first column result.

```
SELECT
PRODUCT_ID,
MAX(CASE WHEN STORE = 'store1' THEN PRICE ELSE NULL END) AS STORE1,
MAX(CASE WHEN STORE = 'store2' THEN PRICE ELSE NULL END) AS STORE2,
MAX(CASE WHEN STORE = 'store3' THEN PRICE ELSE NULL END) AS STORE3
FROM PRODUCTS
GROUP BY PRODUCT_ID
```

1783. Grand Slam Titles ↗

Table: Players

Column Name	Type
player_id	int
player_name	varchar

player_id is the primary key for this table.

Each row in this table contains the name and the ID of a tennis player.

Table: Championships

Column Name	Type
year	int
Wimbledon	int
Fr_open	int
US_open	int
Au_open	int

year is the primary key for this table.

Each row of this table contains the IDs of the players who won one each tennis tourna

Write an SQL query to report the number of grand slam tournaments won by each player. Do not include the players who did not win any tournament.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Players table:

player_id	player_name
1	Nadal
2	Federer
3	Novak

Championships table:

year	Wimbledon	Fr_open	US_open	Au_open
2018	1	1	1	1
2019	1	1	2	2
2020	2	1	2	2

Output:

player_id	player_name	grand_slams_count
2	Federer	5
1	Nadal	7

Explanation:

Player 1 (Nadal) won 7 titles: Wimbledon (2018, 2019), Fr_open (2018, 2019, 2020), US_open (2019, 2020), and Au_open (2020).

Player 2 (Federer) won 5 titles: Wimbledon (2020), US_open (2019, 2020), and Au_open (2018, 2019).

Player 3 (Novak) did not win anything, we did not include them in the result table.

- Simple Most solution Using Union ALL

```
SELECT
TEMP.PLAYER_ID,
TEMP.PLAYER_NAME,
SUM(TEMP.GRAND_SLAMS_COUNT) AS GRAND_SLAMS_COUNT
FROM
(
    SELECT
P.PLAYER_ID,
P.PLAYER_NAME,
COUNT(*) AS GRAND_SLAMS_COUNT
FROM PLAYERS AS P JOIN CHAMPIONSHIPS AS CS
ON P.PLAYER_ID = CS.WIMBLEDON
GROUP BY P.PLAYER_ID, P.PLAYER_NAME

UNION ALL

SELECT
P.PLAYER_ID,
P.PLAYER_NAME,
COUNT(*) AS GRAND_SLAMS_COUNT
FROM PLAYERS AS P JOIN CHAMPIONSHIPS AS CS
ON P.PLAYER_ID = CS.FR_OPEN
GROUP BY P.PLAYER_ID, P.PLAYER_NAME

UNION ALL

SELECT
P.PLAYER_ID,
P.PLAYER_NAME,
COUNT(*) AS GRAND_SLAMS_COUNT
FROM PLAYERS AS P JOIN CHAMPIONSHIPS AS CS
ON P.PLAYER_ID = CS.US_OPEN
GROUP BY P.PLAYER_ID, P.PLAYER_NAME

UNION ALL

SELECT
P.PLAYER_ID,
P.PLAYER_NAME,
COUNT(*) AS GRAND_SLAMS_COUNT
FROM PLAYERS AS P JOIN CHAMPIONSHIPS AS CS
ON P.PLAYER_ID = CS.AU_OPEN
GROUP BY P.PLAYER_ID, P.PLAYER_NAME
) AS TEMP GROUP BY TEMP.PLAYER_ID, TEMP.PLAYER_NAME'''
```

1789. Primary Department for Each Employee ↗



Table: Employee

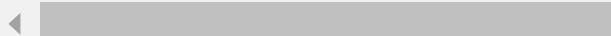
Column Name	Type
employee_id	int
department_id	int
primary_flag	varchar

(employee_id, department_id) is the primary key for this table.

employee_id is the id of the employee.

department_id is the id of the department to which the employee belongs.

primary_flag is an ENUM of type ('Y', 'N'). If the flag is 'Y', the department is th



Employees can belong to multiple departments. When the employee joins other departments, they need to decide which department is their primary department. Note that when an employee belongs to only one department, their primary column is 'N' .

Write an SQL query to report all the employees with their primary department. For employees who belong to one department, report their only department.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Employee table:

employee_id	department_id	primary_flag
1	1	N
2	1	Y
2	2	N
3	3	N
4	2	N
4	3	Y
4	4	N

Output:

employee_id	department_id
1	1
2	1
3	3
4	3

Explanation:

- The Primary department for employee 1 is 1.
- The Primary department for employee 2 is 1.
- The Primary department for employee 3 is 3.
- The Primary department for employee 4 is 3.

- Solution Using Union solves the Problem

```

SELECT EMPLOYEE_ID , DEPARTMENT_ID
FROM EMPLOYEE
WHERE PRIMARY_FLAG = 'Y'

UNION
(
    SELECT EMPLOYEE_ID, DEPARTMENT_ID
    FROM EMPLOYEE
    WHERE EMPLOYEE_ID NOT IN (SELECT EMPLOYEE_ID
                                FROM EMPLOYEE
                                WHERE PRIMARY_FLAG = 'Y')
)

```

1795. Rearrange Products Table ↗



Table: Products

Column Name	Type
product_id	int
store1	int
store2	int
store3	int

product_id is the primary key for this table.

Each row in this table indicates the product's price in 3 different stores: store1, store2, store3. If the product is not available in a store, the price will be null in that store's column.

Write an SQL query to rearrange the Products table so that each row has (product_id, store, price) . If a product is not available in a store, do **not** include a row with that product_id and store combination in the result table.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Products table:

product_id	store1	store2	store3
0	95	100	105
1	70	null	80

Output:

product_id	store	price
0	store1	95
0	store2	100
0	store3	105
1	store1	70
1	store3	80

Explanation:

Product 0 is available in all three stores with prices 95, 100, and 105 respectively. Product 1 is available in store1 with price 70 and store3 with price 80. The product

*MySQL Solution For the Above Problem

```
select * from
(
  select
    product_id,
    "store1" as store,
    store1 as price
  from products
  group by product_id, store1
  union

  select
    product_id,
    "store2" as store,
    store2 as price
  from products
  group by product_id, store2

  union

  select
    product_id,
    "store3" as store,
    store3 as price
  from products
  group by product_id, store3

) as t1
where t1.price is not null
order by t1.product_id
```

*MSSQL Solution to the above Problem Statement...

```

SELECT
*
FROM
(
SELECT PRODUCT_ID, 'store1' as store, STORE1 as price
FROM PRODUCTS
GROUP BY PRODUCT_ID, STORE1

UNION

SELECT PRODUCT_ID, 'store2' as store, STORE2 as price
FROM PRODUCTS
GROUP BY PRODUCT_ID, STORE2

UNION

SELECT PRODUCT_ID, 'store3' as store, STORE3 as price
FROM PRODUCTS
GROUP BY PRODUCT_ID, STORE3

) AS T
WHERE T.price IS NOT NULL
ORDER BY T.PRODUCT_ID

```

1822. Sign of the Product of an Array ↗

There is a function `signFunc(x)` that returns:

- 1 if x is positive.
- -1 if x is negative.
- 0 if x is equal to 0.

You are given an integer array `nums`. Let `product` be the product of all values in the array `nums`.

Return `signFunc(product)`.

Example 1:

```

Input: nums = [-1,-2,-3,-4,3,2,1]
Output: 1
Explanation: The product of all values in the array is 144, and signFunc(144) = 1

```

Example 2:

Input: nums = [1,5,0,2,-3]

Output: 0

Explanation: The product of all values in the array is 0, and signFunc(0) = 0

Example 3:

Input: nums = [-1,1,-1,1,-1]

Output: -1

Explanation: The product of all values in the array is -1, and signFunc(-1) = -1

Constraints:

- $1 \leq \text{nums.length} \leq 1000$
- $-100 \leq \text{nums}[i] \leq 100$

```
class Solution:  
    def arraySign(self, nums: List[int]) -> int:  
        prod = 1  
        for val in nums:  
            prod = prod * val  
        if prod > 0:  
            return 1  
        elif prod < 0:  
            return -1  
        else:  
            return 0
```

1809. Ad-Free Sessions ↗



Table: Playback

Column Name	Type
session_id	int
customer_id	int
start_time	int
end_time	int

session_id is the primary key for this table.

customer_id is the ID of the customer watching this session.

The session runs during the **inclusive** interval between start_time and end_time.

It is guaranteed that start_time <= end_time and that two sessions for the same cust

Table: Ads

Column Name	Type
ad_id	int
customer_id	int
timestamp	int

ad_id is the primary key for this table.

customer_id is the ID of the customer viewing this ad.

timestamp is the moment of time at which the ad was shown.

Write an SQL query to report all the sessions that did not get shown any ads.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Playback table:

session_id	customer_id	start_time	end_time
1	1	1	5
2	1	15	23
3	2	10	12
4	2	17	28
5	2	2	8

Ads table:

ad_id	customer_id	timestamp
1	1	5
2	2	17
3	2	20

Output:

session_id
2
3
5

Explanation:

The ad with ID 1 was shown to user 1 at time 5 while they were in session 1.
 The ad with ID 2 was shown to user 2 at time 17 while they were in session 4.
 The ad with ID 3 was shown to user 2 at time 20 while they were in session 4.
 We can see that sessions 1 and 4 had at least one ad. Sessions 2, 3, and 5 did not have any ads.

```
SELECT SESSION_ID
FROM PLAYBACK WHERE SESSION_ID NOT IN
(
  SELECT
    PB.SESSION_ID
  FROM PLAYBACK AS PB LEFT JOIN ADS AS A
  ON PB.CUSTOMER_ID = A.CUSTOMER_ID
  WHERE A.TIMESTAMP BETWEEN PB.START_TIME AND PB.END_TIME #AND
  GROUP BY PB.CUSTOMER_ID
)
```

1821. Find Customers With Positive Revenue this Year ↗

Table: Customers

Column Name	Type
customer_id	int
year	int
revenue	int

(customer_id, year) is the primary key for this table.

This table contains the customer ID and the revenue of customers in different years.
Note that this revenue can be negative.

Write an SQL query to report the customers with **positive revenue** in the year 2021.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Customers table:

customer_id	year	revenue
1	2018	50
1	2021	30
1	2020	70
2	2021	-50
3	2018	10
3	2016	50
4	2021	20

Output:

customer_id
1
4

Explanation:

Customer 1 has revenue equal to 30 in the year 2021.

Customer 2 has revenue equal to -50 in the year 2021.

Customer 3 has no revenue in the year 2021.

Customer 4 has revenue equal to 20 in the year 2021.

Thus only customers 1 and 4 have positive revenue in the year 2021.

```
SELECT CUSTOMER_ID
FROM CUSTOMERS
WHERE REVENUE > 0
AND YEAR = 2021
```

1831. Maximum Transaction Each Day ↴



Table: Transactions

Column Name	Type
transaction_id	int
day	datetime
amount	int

transaction_id is the primary key for this table.

Each row contains information about one transaction.

Write an SQL query to report the IDs of the transactions with the **maximum** amount on their respective day. If in one day there are multiple such transactions, return all of them.

Return the result table **ordered by transaction_id in ascending order**.

The query result format is in the following example.

Example 1:

Input:

Transactions table:

transaction_id	day	amount
8	2021-4-3 15:57:28	57
9	2021-4-28 08:47:25	21
1	2021-4-29 13:28:30	58
5	2021-4-28 16:39:59	40
6	2021-4-29 23:39:28	58

Output:

transaction_id
1
5
6
8

Explanation:

"2021-4-3" --> We have one transaction with ID 8, so we add 8 to the result table.
"2021-4-28" --> We have two transactions with IDs 5 and 9. The transaction with ID 5
"2021-4-29" --> We have two transactions with IDs 1 and 6. Both transactions have the same maximum amount.
We order the result table by transaction_id after collecting these IDs.



Follow up: Could you solve it without using the MAX() function?

```

SELECT TRANSACTION_ID
FROM TRANSACTIONS
WHERE (DATE(DAY), AMOUNT) IN
(
    SELECT DATE(DAY), MAX(AMOUNT) AS AMT
    FROM TRANSACTIONS
    GROUP BY DATE(DAY)
)
ORDER BY TRANSACTION_ID

```

1841. League Statistics ↗

Table: Teams

Column Name	Type
team_id	int
team_name	varchar

team_id is the primary key for this table.

Each row contains information about one team in the league.

Table: Matches

Column Name	Type
home_team_id	int
away_team_id	int
home_team_goals	int
away_team_goals	int

(home_team_id, away_team_id) is the primary key for this table.

Each row contains information about one match.

home_team_goals is the number of goals scored by the home team.

away_team_goals is the number of goals scored by the away team.

The winner of the match is the team with the higher number of goals.

Write an SQL query to report the statistics of the league. The statistics should be built using the played matches where the **winning** team gets **three points** and the **losing** team gets **no points**. If a match ends with a **draw**, both teams get **one point**.

Each row of the result table should contain:

- `team_name` - The name of the team in the `Teams` table.
- `matches_played` - The number of matches played as either a home or away team.
- `points` - The total points the team has so far.
- `goal_for` - The total number of goals scored by the team across all matches.
- `goal_against` - The total number of goals scored by opponent teams against this team across all matches.
- `goal_diff` - The result of `goal_for` - `goal_against`.

Return the result table ordered by `points` **in descending order**. If two or more teams have the same `points`, order them by `goal_diff` **in descending order**. If there is still a tie, order them by `team_name` **in lexicographical order**.

The query result format is in the following example.

Example 1:

Input:

Teams table:

team_id	team_name
1	Ajax
4	Dortmund
6	Arsenal

Matches table:

home_team_id	away_team_id	home_team_goals	away_team_goals
1	4	0	1
1	6	3	3
4	1	5	2
6	1	0	0

Output:

team_name	matches_played	points	goal_for	goal_against	goal_diff
Dortmund	2	6	6	2	4
Arsenal	2	2	3	3	0
Ajax	4	2	5	9	-4

Explanation:

Ajax (team_id=1) played 4 matches: 2 losses and 2 draws. Total points = 0 + 0 + 1 + 1 = 2.

Dortmund (team_id=4) played 2 matches: 2 wins. Total points = 3 + 3 = 6.

Arsenal (team_id=6) played 2 matches: 2 draws. Total points = 1 + 1 = 2.

Dortmund is the first team in the table. Ajax and Arsenal have the same points, but



- Initial Draft Version Of the Code

```
WITH LEAGUE_STATS AS
(
    SELECT
        T.TEAM_NAME AS TEAM_NAME,
        COUNT(*) AS MATCHES_PLAYED,
        # SUM(IF())AS POINTS,
        SUM(HOME_TEAM_GOALS) AS GOAL_FOR,
        SUM(AWAY_TEAM_GOALS) AS GOAL AGAINST,
        SUM(HOME_TEAM_GOALS) - SUM(AWAY_TEAM_GOALS) AS GOAL_DIFF
    FROM TEAMS AS T LEFT JOIN MATCHES AS M
    ON T.TEAM_ID = M.HOME_TEAM_ID
    GROUP BY T.TEAM_NAME

    UNION ALL

    SELECT
        T.TEAM_NAME AS TEAM_NAME,
        COUNT(*) AS MATCHES_PLAYED,
        SUM(AWAY_TEAM_GOALS) AS GOAL_FOR,
        SUM(HOME_TEAM_GOALS) AS GOAL AGAINST,
        SUM(AWAY_TEAM_GOALS) - SUM(HOME_TEAM_GOALS) AS GOAL_DIFF
    FROM TEAMS AS T LEFT JOIN MATCHES AS M
    ON T.TEAM_ID = M.AWAY_TEAM_ID
    GROUP BY T.TEAM_NAME
)

SELECT
    TEAM_NAME,
    SUM(MATCHES_PLAYED) AS MATCHES_PLAYED,
    SUM(GOAL_FOR) AS GOAL_FOR,
    SUM(GOAL AGAINST) AS GOAL AGAINST,
    SUM(GOAL_DIFF) AS GOAL_DIFF
FROM LEAGUE_STATS
GROUP BY TEAM_NAME
```

- Final Code Which Works Perfectly fine

```

WITH LEAGUE_STATS AS
(
    SELECT
        T.TEAM_NAME AS TEAM_NAME,
        COUNT(*) AS MATCHES_PLAYED,
        SUM(
            (
                CASE
                    WHEN HOME_TEAM_GOALS > AWAY_TEAM_GOALS THEN 3
                    WHEN HOME_TEAM_GOALS = AWAY_TEAM_GOALS THEN 1
                    ELSE 0
                END
            ) AS POINTS,
            SUM(HOME_TEAM_GOALS) AS GOAL_FOR,
            SUM(AWAY_TEAM_GOALS) AS GOAL AGAINST,
            SUM(HOME_TEAM_GOALS) - SUM(AWAY_TEAM_GOALS) AS GOAL_DIFF
        )
        FROM TEAMS AS T LEFT JOIN MATCHES AS M
        ON T.TEAM_ID = M.HOME_TEAM_ID
        WHERE M.HOME_TEAM_ID IS NOT NULL
        GROUP BY T.TEAM_NAME
    UNION ALL
    SELECT
        T.TEAM_NAME AS TEAM_NAME,
        COUNT(*) AS MATCHES_PLAYED,
        SUM(
            (
                CASE
                    WHEN AWAY_TEAM_GOALS > HOME_TEAM_GOALS THEN 3
                    WHEN AWAY_TEAM_GOALS = HOME_TEAM_GOALS THEN 1
                    ELSE 0
                END
            ) AS POINTS,
            SUM(AWAY_TEAM_GOALS) AS GOAL_FOR,
            SUM(HOME_TEAM_GOALS) AS GOAL AGAINST,
            SUM(AWAY_TEAM_GOALS) - SUM(HOME_TEAM_GOALS) AS GOAL_DIFF
        )
        FROM TEAMS AS T LEFT JOIN MATCHES AS M
        ON T.TEAM_ID = M.AWAY_TEAM_ID
        WHERE M.AWAY_TEAM_ID IS NOT NULL
        GROUP BY T.TEAM_NAME
    )
SELECT
    TEAM_NAME,
    SUM(MATCHES_PLAYED) AS MATCHES_PLAYED,
    SUM(POINTS) AS POINTS,
    SUM(GOAL_FOR) AS GOAL_FOR,
    SUM(GOAL AGAINST) AS GOAL AGAINST,

```

```
SUM(GOAL_DIFF) AS GOAL_DIFF
FROM LEAGUE_STATS
GROUP BY TEAM_NAME
ORDER BY POINTS DESC, GOAL_DIFF DESC, TEAM_NAME ASC
```

1853. Convert Date Format ↗

Table: Days

Column Name	Type
day	date

day is the primary key for this table.

Write an SQL query to convert each date in `Days` into a string formatted as "`day_name, month_name day, year`".

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Days table:

day
2022-04-12
2021-08-09
2020-06-26

Output:

day
Tuesday, April 12, 2022
Monday, August 9, 2021
Friday, June 26, 2020

Explanation: Please note that the output is case-sensitive.

- Using CONCAT to Solve the Problem

```

SELECT
CONCAT(DAYNAME(DAY), ', ', MONTHNAME(DAY), ' ', DAY(DAY), ', ', YEAR(DAY))
AS DAY
FROM DAYS

```

1867. Orders With Maximum Quantity Above Average ↴

Table: OrdersDetails

Column Name	Type
order_id	int
product_id	int
quantity	int

(order_id, product_id) is the primary key for this table.

A single order is represented as multiple rows, one row for each product in the order. Each row of this table contains the quantity ordered of the product product_id in th

You are running an e-commerce site that is looking for **imbalanced orders**. An **imbalanced order** is one whose **maximum** quantity is **strictly greater** than the **average** quantity of **every order (including itself)**.

The **average** quantity of an order is calculated as (total quantity of all products in the order) / (number of different products in the order). The **maximum** quantity of an order is the highest quantity of any single product in the order.

Write an SQL query to find the `order_id` of all **imbalanced orders**.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

OrdersDetails table:

order_id	product_id	quantity
1	1	12
1	2	10
1	3	15
2	1	8
2	4	4
2	5	6
3	3	5
3	4	18
4	5	2
4	6	8
5	7	9
5	8	9
3	9	20
2	9	4

Output:

order_id
1
3

Explanation:

The average quantity of each order is:

- order_id=1: $(12+10+15)/3 = 12.3333333$
- order_id=2: $(8+4+6+4)/4 = 5.5$
- order_id=3: $(5+18+20)/3 = 14.333333$
- order_id=4: $(2+8)/2 = 5$
- order_id=5: $(9+9)/2 = 9$

The maximum quantity of each order is:

- order_id=1: $\max(12, 10, 15) = 15$
- order_id=2: $\max(8, 4, 6, 4) = 8$
- order_id=3: $\max(5, 18, 20) = 20$
- order_id=4: $\max(2, 8) = 8$
- order_id=5: $\max(9, 9) = 9$

Orders 1 and 3 are imbalanced because they have a maximum quantity that exceeds the

```

WITH CTE AS
(
    SELECT
        ORDER_ID,
        MAX(QUANTITY) AS QUANTITY,
        SUM(QUANTITY)/ COUNT(DISTINCT PRODUCT_ID) AS AVERAGE
    FROM ORDERSDETAILS
    GROUP BY ORDER_ID
)

SELECT
    ORDER_ID
FROM CTE WHERE QUANTITY > (SELECT MAX(AVERAGE) FROM CTE)

```

1873. Calculate Special Bonus ↗



Table: Employees

Column Name	Type
employee_id	int
name	varchar
salary	int

employee_id is the primary key for this table.

Each row of this table indicates the employee ID, employee name, and salary.

Write an SQL query to calculate the bonus of each employee. The bonus of an employee is 100% of their salary if the ID of the employee is **an odd number** and **the employee name does not start with the character 'M'**. The bonus of an employee is 0 otherwise.

Return the result table ordered by employee_id .

The query result format is in the following example.

Example 1:

Input:

Employees table:

employee_id	name	salary
2	Meir	3000
3	Michael	3800
7	Addilyn	7400
8	Juan	6100
9	Kannon	7700

Output:

employee_id	bonus
2	0
3	0
7	7400
8	0
9	7700

Explanation:

The employees with IDs 2 and 8 get 0 bonus because they have an even employee_id.

The employee with ID 3 gets 0 bonus because their name starts with 'M'.

The rest of the employees get a 100% bonus.

- T-Sql Based Solution

```
SELECT employee_id, case
    when employee_id %2 != 0 and name not like 'M%' then salary
    else 0
    end as bonus
from employees
```



- MySql Based Solution

```
SELECT employee_id,
CASE
    WHEN EMPLOYEE_ID % 2 != 0 AND NAME NOT LIKE 'M%' THEN SALARY
    ELSE
        END
    AS bonus
FROM employees
```

1875. Group Employees of the Same Salary ↗



Table: Employees

Column Name	Type
employee_id	int
name	varchar
salary	int

`employee_id` is the primary key for this table.

Each row of this table indicates the employee ID, employee name, and salary.

A company wants to divide the employees into teams such that all the members on each team have the **same salary**. The teams should follow these criteria:

- Each team should consist of **at least two** employees.
- All the employees on a team should have the **same salary**.
- All the employees of the same salary should be assigned to the same team.
- If the salary of an employee is unique, we **do not** assign this employee to any team.
- A team's ID is assigned based on the **rank of the team's salary** relative to the other teams' salaries, where the team with the **lowest** salary has `team_id = 1`. Note that the salaries for employees not on a team are **not included** in this ranking.

Write an SQL query to get the `team_id` of each employee that is in a team.

Return the result table ordered by `team_id` **in ascending order**. In case of a tie, order it by `employee_id` **in ascending order**.

The query result format is in the following example.

Example 1:

Input:

Employees table:

employee_id	name	salary
2	Meir	3000
3	Michael	3000
7	Addilyn	7400
8	Juan	6100
9	Kannon	7400

Output:

employee_id	name	salary	team_id
2	Meir	3000	1
3	Michael	3000	1
7	Addilyn	7400	2
9	Kannon	7400	2

Explanation:

Meir (employee_id=2) and Michael (employee_id=3) are in the same team because they have the same salary of 3000.
Addilyn (employee_id=7) and Kannon (employee_id=9) are in the same team because they have the same salary of 7400.
Juan (employee_id=8) is not included in any team because their salary of 6100 is unique.

The team IDs are assigned as follows (based on salary ranking, lowest first):

- team_id=1: Meir and Michael, a salary of 3000
- team_id=2: Addilyn and Kannon, a salary of 7400

Juan's salary of 6100 is not included in the ranking because they are not on a team.



```

WITH CTE AS
(
    SELECT *, DENSE_RANK() OVER(ORDER BY SALARY DESC) AS CRANK
    FROM EMPLOYEES
)

SELECT
EMPLOYEE_ID,
NAME,
SALARY,
DENSE_RANK() OVER(ORDER BY CRANK DESC) AS TEAM_ID
FROM CTE
WHERE CRANK NOT IN (SELECT
                      CRANK
                      FROM CTE
                      GROUP BY CRANK
                      HAVING COUNT(*) = 1)

ORDER BY TEAM_ID, EMPLOYEE_ID ASC

```

1907. Count Salary Categories ↗

Table: Accounts

Column Name	Type
account_id	int
income	int

account_id is the primary key for this table.

Each row contains information about the monthly income for one bank account.

Write an SQL query to report the number of bank accounts of each salary category. The salary categories are:

- "Low Salary" : All the salaries **strictly less** than \$20000 .
- "Average Salary" : All the salaries in the **inclusive** range [\$20000, \$50000] .
- "High Salary" : All the salaries **strictly greater** than \$50000 .

The result table **must** contain all three categories. If there are no accounts in a category, then report 0 .

Return the result table in **any order**.

The query result format is in the following example.

Example 1:**Input:**

Accounts table:

account_id	income
3	108939
2	12747
8	87709
6	91796

Output:

category	accounts_count
Low Salary	1
Average Salary	0
High Salary	3

Explanation:

Low Salary: Account 2.

Average Salary: No accounts.

High Salary: Accounts 3, 6, and 8.

```

WITH CTE1 AS
(
    SELECT "Low Salary" AS COL1
    UNION
    SELECT "Average Salary" AS COL1
    UNION
    SELECT "High Salary" AS COL1
),
CTE2 AS (
    SELECT
        A.CATEGORY, COUNT(DISTINCT A.ACCOUNT_ID) AS ACCOUNTS_COUNT
    FROM
        (
            SELECT
                *,
                CASE
                    WHEN INCOME < 20000 THEN "Low Salary"
                    WHEN INCOME >= 20000 AND INCOME <= 50000 THEN "Average Salary"
                    ELSE "High Salary"
                END AS CATEGORY
            FROM ACCOUNTS
        ) AS A
    GROUP BY A.CATEGORY
)
SELECT * FROM CTE2
UNION
SELECT COL1 AS CATEGORY, 0 AS ACCOUNTS_COUNT
FROM CTE1 WHERE COL1 NOT IN (SELECT CATEGORY FROM CTE2)

```

1934. Confirmation Rate ↗



Table: Signups

Column Name	Type
user_id	int
time_stamp	datetime

user_id is the primary key for this table.

Each row contains information about the signup time for the user with ID user_id.

Table: Confirmations

```
+-----+-----+
| Column Name | Type      |
+-----+-----+
| user_id     | int       |
| time_stamp   | datetime  |
| action       | ENUM      |
+-----+-----+
(user_id, time_stamp) is the primary key for this table.
user_id is a foreign key with a reference to the Signups table.
action is an ENUM of the type ('confirmed', 'timeout')
Each row of this table indicates that the user with ID user_id requested a confirmation message at time_stamp with action action.
```

The **confirmation rate** of a user is the number of 'confirmed' messages divided by the total number of requested confirmation messages. The confirmation rate of a user that did not request any confirmation messages is 0. Round the confirmation rate to **two decimal** places.

Write an SQL query to find the **confirmation rate** of each user.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Signups table:

user_id	time_stamp
3	2020-03-21 10:16:13
7	2020-01-04 13:57:59
2	2020-07-29 23:09:44
6	2020-12-09 10:39:37

Confirmations table:

user_id	time_stamp	action
3	2021-01-06 03:30:46	timeout
3	2021-07-14 14:00:00	timeout
7	2021-06-12 11:57:29	confirmed
7	2021-06-13 12:58:28	confirmed
7	2021-06-14 13:59:27	confirmed
2	2021-01-22 00:00:00	confirmed
2	2021-02-28 23:59:59	timeout

Output:

user_id	confirmation_rate
6	0.00
3	0.00
7	1.00
2	0.50

Explanation:

User 6 did not request any confirmation messages. The confirmation rate is 0.

User 3 made 2 requests and both timed out. The confirmation rate is 0.

User 7 made 3 requests and all were confirmed. The confirmation rate is 1.

User 2 made 2 requests where one was confirmed and the other timed out. The confirma

- Simple Solution Using Left Join

```

SELECT
SG.USER_ID,
ROUND(SUM(CASE WHEN C.ACTION = 'confirmed' THEN 1 ELSE 0 END)/ SUM(CASE WHEN
C.ACTION = 'confirmed' THEN 1 ELSE 1 END),2) AS CONFIRMATION_RATE
FROM SIGNUPS AS SG LEFT JOIN CONFIRMATIONS AS C
ON SG.USER_ID = C.USER_ID
GROUP BY SG.USER_ID

```

1965. Employees With Missing Information ↗



Table: Employees

Column Name	Type
employee_id	int
name	varchar

employee_id is the primary key for this table.

Each row of this table indicates the name of the employee whose ID is employee_id.

Table: Salaries

Column Name	Type
employee_id	int
salary	int

employee_id is the primary key for this table.

Each row of this table indicates the salary of the employee whose ID is employee_id.



Write an SQL query to report the IDs of all the employees with **missing information**. The information of an employee is missing if:

- The employee's **name** is missing, or
- The employee's **salary** is missing.

Return the result table ordered by `employee_id` **in ascending order**.

The query result format is in the following example.

Example 1:

Input:

Employees table:

employee_id	name
2	Crew
4	Haven
5	Kristian

Salaries table:

employee_id	salary
5	76071
1	22517
4	63539

Output:

employee_id
1
2

Explanation:

Employees 1, 2, 4, and 5 are working at this company.

The name of employee 1 is missing.

The salary of employee 2 is missing.

- MYSQL Solution to find the result of the Above Statement.

```

SELECT EMPLOYEE_ID
FROM EMPLOYEES AS E
WHERE EMPLOYEE_ID NOT IN (SELECT EMPLOYEE_ID FROM SALARIES)

UNION

SELECT EMPLOYEE_ID
FROM SALARIES
WHERE EMPLOYEE_ID NOT IN (SELECT EMPLOYEE_ID FROM EMPLOYEES)

ORDER BY EMPLOYEE_ID

```

2006. Count Number of Pairs With Absolute Difference K ↗

Given an integer array `nums` and an integer `k`, return *the number of pairs* (i, j) where $i < j$ such that $|nums[i] - nums[j]| == k$.

The value of $|x|$ is defined as:

- x if $x \geq 0$.
- $-x$ if $x < 0$.

Example 1:

Input: `nums = [1,2,2,1]`, `k = 1`

Output: 4

Explanation: The pairs with an absolute difference of 1 are:

- [1,2,2,1]
- [1,2,2,1]
- [1,2,2,1]
- [1,2,2,1]

Example 2:

Input: `nums = [1,3]`, `k = 3`

Output: 0

Explanation: There are no pairs with an absolute difference of 3.

Example 3:

Input: `nums = [3,2,1,5,4]`, `k = 2`

Output: 3

Explanation: The pairs with an absolute difference of 2 are:

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

Constraints:

- $1 \leq \text{nums.length} \leq 200$
- $1 \leq \text{nums}[i] \leq 100$
- $1 \leq k \leq 99$

Given an integer array `nums` and an integer `k`, return the number of pairs (i, j) where $i < j$ such that $|nums[i] - nums[j]| == k$.

The value of $|x|$ is defined as:

```
x if x >= 0.  
-x if x < 0.
```

Example 1:

Input: `nums = [1,2,2,1]`, `k = 1`

Output: 4

Explanation: The pairs with an absolute difference of 1 are:

- [1,2,2,1]
- [1,2,2,1]
- [1,2,2,1]
- [1,2,2,1]

Example 2:

Input: `nums = [1,3]`, `k = 3`

Output: 0

Explanation: There are no pairs with an absolute difference of 3.

Example 3:

Input: `nums = [3,2,1,5,4]`, `k = 2`

Output: 3

Explanation: The pairs with an absolute difference of 2 are:

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

Solution for the Above problem Statement:

```
class Solution:  
    def countKDifference(self, nums: List[int], k: int) -> int:  
        count = 0  
        # O(N^2) Time Complexity  
        for i in range(len(nums)):  
            for j in range(i+1, len(nums)):  
                if abs(nums[i] - nums[j]) == k:  
                    count +=1  
        return count
```

1978. Employees Whose Manager Left the Company ↗



Table: Employees

```
+-----+-----+
| Column Name | Type      |
+-----+-----+
| employee_id | int       |
| name         | varchar   |
| manager_id   | int       |
| salary        | int       |
+-----+-----+
```

`employee_id` is the primary key for this table.

This table contains information about the employees, their salary, and the ID of the



Write an SQL query to report the IDs of the employees whose salary is strictly less than \$30000 and whose manager left the company. When a manager leaves the company, their information is deleted from the `Employees` table, but the reports still have their `manager_id` set to the manager that left.

Return the result table ordered by `employee_id`.

The query result format is in the following example.

Example 1:

Input:

Employees table:

employee_id	name	manager_id	salary
3	Mila	9	60301
12	Antonella	null	31000
13	Emery	null	67084
1	Kalel	11	21241
9	Mikaela	null	50937
11	Joziah	6	28485

Output:

employee_id
11

Explanation:

The employees with a salary less than \$30000 are 1 (Kalel) and 11 (Joziah).

Kalel's manager is employee 11, who is still in the company (Joziah).

Joziah's manager is employee 6, who left the company because there is no row for emp

```
SELECT E.EMPLOYEE_ID
FROM EMPLOYEES AS E LEFT JOIN EMPLOYEES AS EE
ON E.MANAGER_ID = EE.EMPLOYEE_ID
WHERE E.MANAGER_ID IS NOT NULL AND EE.EMPLOYEE_ID IS NULL
AND E.SALARY < 30000
ORDER BY E.EMPLOYEE_ID
```

2011. Final Value of Variable After Performing Operations ↴

There is a programming language with only **four** operations and **one** variable X :

- `++X` and `X++` **increments** the value of the variable X by 1 .
- `--X` and `X--` **decrements** the value of the variable X by 1 .

Initially, the value of X is 0 .

Given an array of strings `operations` containing a list of operations, return *the final value of X after performing all the operations.*

Example 1:

Input: operations = ["--X", "X++", "X++"]
Output: 1
Explanation: The operations are performed as follows:
Initially, X = 0.
--X: X is decremented by 1, X = 0 - 1 = -1.
X++: X is incremented by 1, X = -1 + 1 = 0.
X++: X is incremented by 1, X = 0 + 1 = 1.

Example 2:

Input: operations = ["++X", "++X", "X++"]
Output: 3
Explanation: The operations are performed as follows:
Initially, X = 0.
++X: X is incremented by 1, X = 0 + 1 = 1.
++X: X is incremented by 1, X = 1 + 1 = 2.
X++: X is incremented by 1, X = 2 + 1 = 3.

Example 3:

Input: operations = ["X++", "++X", "--X", "X--"]
Output: 0
Explanation: The operations are performed as follows:
Initially, X = 0.
X++: X is incremented by 1, X = 0 + 1 = 1.
++X: X is incremented by 1, X = 1 + 1 = 2.
--X: X is decremented by 1, X = 2 - 1 = 1.
X--: X is decremented by 1, X = 1 - 1 = 0.

Constraints:

- $1 \leq \text{operations.length} \leq 100$
 - $\text{operations}[i]$ will be either " $++X$ ", " $X++$ ", " $--X$ ", or " $X--$ ".
- Here I have used, basic if and iteration conditions to get the desired result.

```

class Solution:
    def finalValueAfterOperations(self, operations: List[str]) -> int:
        x = 0
        for i in operations:
            print(i[0:-1])
            if i[0:-1] == '++':
                x += 1
                print("1st if: ",i,x)

            if i[0:-1] == '--':
                x -= 1
                print("2nd if: ",i,x)

            if i[1::] == '--':
                x -= 1
                print("3rd if: ",i,x)

            if i[1::] == '++':
                x +=1
                print("4th if: ",i,x)
        return x

```

2020. Number of Accounts That Did Not Stream ↗▼

Table: Subscriptions

Column Name	Type
account_id	int
start_date	date
end_date	date

account_id is the primary key column for this table.

Each row of this table indicates the start and end dates of an account's subscription.
Note that always start_date < end_date.

Table: Streams

```
+-----+-----+
| Column Name | Type |
+-----+-----+
| session_id | int  |
| account_id | int  |
| stream_date | date |
+-----+-----+
session_id is the primary key column for this table.
account_id is a foreign key from the Subscriptions table.
Each row of this table contains information about the account and the date associate
```

Write an SQL query to report the number of accounts that bought a subscription in 2021 but did not have any stream session.

The query result format is in the following example.

Example 1:

Input:

Subscriptions table:

account_id	start_date	end_date
9	2020-02-18	2021-10-30
3	2021-09-21	2021-11-13
11	2020-02-28	2020-08-18
13	2021-04-20	2021-09-22
4	2020-10-26	2021-05-08
5	2020-09-11	2021-01-17

Streams table:

session_id	account_id	stream_date
14	9	2020-05-16
16	3	2021-10-27
18	11	2020-04-29
17	13	2021-08-08
19	4	2020-12-31
13	5	2021-01-05

Output:

accounts_count
2

Explanation: Users 4 and 9 did not stream in 2021.

User 11 did not subscribe in 2021.

- Unused Logic for the Problem

```
(  
    SELECT  
        SUB.ACOUNT_ID,  
        CASE  
            WHEN  
                YEAR(SUB.START_DATE) IN (2020,2021) AND YEAR(SUB.END_DATE) IN (2021)  
            THEN 'Y'  
            ELSE 'N'  
        END AS IS_SUBSRIBED  
    FROM SUBSCRIPTIONS  
) AS SUB
```

- Final Solution

```

SELECT COUNT(*) AS ACCOUNTS_COUNT
FROM SUBSCRIPTIONS
WHERE ACCOUNT_ID IN (SELECT
    ACCOUNT_ID
    FROM STREAMS
    WHERE YEAR(STREAM_DATE) = 2020
)
AND YEAR(START_DATE) IN (2020,2021) AND YEAR(END_DATE) IN (2021)

```

2026. Low-Quality Problems ↗



Table: Problems

Column Name	Type
problem_id	int
likes	int
dislikes	int

problem_id is the primary key column for this table.

Each row of this table indicates the number of likes and dislikes for a LeetCode problem.



Write an SQL query to report the IDs of the **low-quality** problems. A LeetCode problem is **low-quality** if the like percentage of the problem (number of likes divided by the total number of votes) is **strictly less than 60%**.

Return the result table ordered by `problem_id` in ascending order.

The query result format is in the following example.

Example 1:

Input:

Problems table:

problem_id	likes	dislikes
6	1290	425
11	2677	8659
1	4446	2760
7	8569	6086
13	2050	4164
10	9002	7446

Output:

problem_id
7
10
11
13

Explanation: The like percentages are as follows:

- Problem 1: $(4446 / (4446 + 2760)) * 100 = 61.69858\%$
- Problem 6: $(1290 / (1290 + 425)) * 100 = 75.21866\%$
- Problem 7: $(8569 / (8569 + 6086)) * 100 = 58.47151\%$
- Problem 10: $(9002 / (9002 + 7446)) * 100 = 54.73006\%$
- Problem 11: $(2677 / (2677 + 8659)) * 100 = 23.61503\%$
- Problem 13: $(2050 / (2050 + 4164)) * 100 = 32.99002\%$

Problems 7, 10, 11, and 13 are low-quality problems because their like percentages are below 60%.

- Using WHERE + ORDER BY Solves the Problem

```
SELECT
PROBLEM_ID
FROM PROBLEMS
WHERE ROUND((LIKES/ (LIKES + DISLIKES)) * 100,2) <= 60
ORDER BY PROBLEM_ID
```

2041. Accepted Candidates From the Interviews ↗ ▾

Table: Candidates

Column Name	Type
candidate_id	int
name	varchar
years_of_exp	int
interview_id	int

candidate_id is the primary key column for this table.

Each row of this table indicates the name of a candidate, their number of years of e

Table: Rounds

Column Name	Type
interview_id	int
round_id	int
score	int

(interview_id, round_id) is the primary key column for this table.

Each row of this table indicates the score of one round of an interview.

Write an SQL query to report the IDs of the candidates who have **at least two** years of experience and the sum of the score of their interview rounds is **strictly greater than 15**.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Candidates table:

candidate_id	name	years_of_exp	interview_id
11	Atticus	1	101
9	Ruben	6	104
6	Aliza	10	109
8	Alfredo	0	107

Rounds table:

interview_id	round_id	score
109	3	4
101	2	8
109	4	1
107	1	3
104	3	6
109	1	4
104	4	7
104	1	2
109	2	1
104	2	7
107	2	3
101	1	8

Output:

candidate_id
9

Explanation:

- Candidate 11: The total score is 16, and they have one year of experience. We do not include them.
- Candidate 9: The total score is 22, and they have six years of experience. We include them.
- Candidate 6: The total score is 10, and they have ten years of experience. We do not include them.
- Candidate 8: The total score is 6, and they have zero years of experience. We do not include them.



```

SELECT
C.CANDIDATE_ID
FROM CANDIDATES AS C LEFT JOIN ROUNDS AS R
ON C.INTERVIEW_ID = R.INTERVIEW_ID
WHERE C.YEARS_OF_EXP >= 2
GROUP BY C.CANDIDATE_ID
HAVING SUM(R.SCORE) > 15

```

2051. The Category of Each Member in the Store ↗

Table: Members

Column Name	Type
member_id	int
name	varchar

member_id is the primary key column for this table.

Each row of this table indicates the name and the ID of a member.

Table: Visits

Column Name	Type
visit_id	int
member_id	int
visit_date	date

visit_id is the primary key column for this table.

member_id is a foreign key to member_id from the Members table.

Each row of this table contains information about the date of a visit to the store a

Table: Purchases

Column Name	Type
visit_id	int
charged_amount	int

visit_id is the primary key column for this table.

visit_id is a foreign key to visit_id from the Visits table.

Each row of this table contains information about the amount charged in a visit to t

A store wants to categorize its members. There are three tiers:

- **"Diamond"**: if the conversion rate is **greater than or equal to 80**.

- "**Gold**": if the conversion rate is **greater than or equal to** 50 and less than 80 .
- "**Silver**": if the conversion rate is **less than** 50 .
- "**Bronze**": if the member never visited the store.

The **conversion rate** of a member is $(100 * \text{total number of purchases for the member}) / \text{total number of visits for the member}$.

Write an SQL query to report the id, the name, and the category of each member.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Members table:

member_id	name
9	Alice
11	Bob
3	Winston
8	Hercy
1	Narihan

Visits table:

visit_id	member_id	visit_date
22	11	2021-10-28
16	11	2021-01-12
18	9	2021-12-10
19	3	2021-10-19
12	11	2021-03-01
17	8	2021-05-07
21	9	2021-05-12

Purchases table:

visit_id	charged_amount
12	2000
18	9000
17	7000

Output:

member_id	name	category
1	Narihan	Bronze
3	Winston	Silver
8	Hercy	Diamond
9	Alice	Gold
11	Bob	Silver

Explanation:

- User Narihan with id = 1 did not make any visits to the store. She gets a Bronze c
- User Winston with id = 3 visited the store one time and did not purchase anything.
- User Hercy with id = 8 visited the store one time and purchased one time. The conv
- User Alice with id = 9 visited the store two times and purchased one time. The con
- User Bob with id = 11 visited the store three times and purchased one time. The co



```

SELECT
M.MEMBER_ID, M.NAME,
CASE
    WHEN
        COUNT(P.VISIT_ID) IS NOT NULL
        AND ((100 * COUNT(P.VISIT_ID))/ COUNT(V.VISIT_ID)) >= 80
    THEN "Diamond"

    WHEN
        COUNT(P.VISIT_ID) IS NOT NULL
        AND
        (((100 * COUNT(P.VISIT_ID))/ COUNT(V.VISIT_ID)) >= 50
        AND ((100 * COUNT(P.VISIT_ID))/ COUNT(V.VISIT_ID)) < 80 )
    THEN "Gold"

    WHEN
        COUNT(P.VISIT_ID) IS NOT NULL
        AND
        (((100 * COUNT(P.VISIT_ID))/ COUNT(V.VISIT_ID)) < 50)
    THEN "Silver"
    ELSE "Bronze"
END AS CATEGORY
# Members(member_id) : Visits(member_id) => 1 : N
FROM MEMBERS AS M LEFT JOIN VISITS AS V
ON M.MEMBER_ID = V.MEMBER_ID LEFT JOIN PURCHASES AS P
# Visits (visit_id) : Purchases(visit_id) => 1 : N
ON V.VISIT_ID = P.VISIT_ID
GROUP BY M.MEMBER_ID,M.NAME

```

2066. Account Balance ↗

Table: Transactions

Column Name	Type
account_id	int
day	date
type	ENUM
amount	int

(account_id, day) is the primary key for this table.

Each row contains information about one transaction, including the transaction type, type is an ENUM of the type ('Deposit','Withdraw')

Write an SQL query to report the balance of each user after each transaction. You may assume that the balance of each account before any transaction is `0` and that the balance will never be below `0` at any moment.

Return the result table **in ascending order** by `account_id`, then by `day` in case of a tie.

The query result format is in the following example.

Example 1:

Input:

Transactions table:

account_id	day	type	amount
1	2021-11-07	Deposit	2000
1	2021-11-09	Withdraw	1000
1	2021-11-11	Deposit	3000
2	2021-12-07	Deposit	7000
2	2021-12-12	Withdraw	7000

Output:

account_id	day	balance
1	2021-11-07	2000
1	2021-11-09	1000
1	2021-11-11	4000
2	2021-12-07	7000
2	2021-12-12	0

Explanation:

Account 1:

- Initial balance is `0`.
- `2021-11-07` --> deposit `2000`. Balance is `0 + 2000 = 2000`.
- `2021-11-09` --> withdraw `1000`. Balance is `2000 - 1000 = 1000`.
- `2021-11-11` --> deposit `3000`. Balance is `1000 + 3000 = 4000`.

Account 2:

- Initial balance is `0`.
- `2021-12-07` --> deposit `7000`. Balance is `0 + 7000 = 7000`.
- `2021-12-12` --> withdraw `7000`. Balance is `7000 - 7000 = 0`.

- Unaccepted Solution Where I was Brainstorming the Question with Different Alternatives.

```

SELECT
ACCOUNT_ID,
DAY,
SUM(AMOUNT) OVER(PARTITION BY ACCOUNT_ID ORDER BY DAY), SUM(
CASE
    WHEN TYPE = 'Withdraw' THEN -AMOUNT
    ELSE 0
END
) AS BALANCE
FROM TRANSACTIONS
GROUP BY ACCOUNT_ID, DAY

```

- Accepted Solution Using CTE + Group BY + CASE

```

WITH CTE AS
(
    SELECT
        ACCOUNT_ID,
        DAY,
        CASE
            WHEN TYPE = 'Deposit' THEN AMOUNT
            ELSE 0
        END AS DEPOSIT,
        CASE
            WHEN TYPE = 'Withdraw' THEN -AMOUNT
            ELSE 0
        END AS WITHDRAW
    FROM TRANSACTIONS
    GROUP BY ACCOUNT_ID, DAY
)

```

) SELECT ACCOUNT_ID, DAY, SUM(DEPOSIT) OVER(PARTITION BY ACCOUNT_ID ORDER BY DAY) +
 SUM(WITHDRAW) OVER(PARTITION BY ACCOUNT_ID ORDER BY DAY) AS BALANCE
 FROM CTE GROUP BY ACCOUNT_ID, DAY ORDER BY ACCOUNT_ID ASC, DAY ASC'''

2072. The Winner University ↗

Table: NewYork

Column Name	Type
student_id	int
score	int

student_id is the primary key for this table.

Each row contains information about the score of one student from New York Universit

Table: California

Column Name	Type
student_id	int
score	int

student_id is the primary key for this table.

Each row contains information about the score of one student from California Univers

There is a competition between New York University and California University. The competition is held between the same number of students from both universities. The university that has more **excellent students** wins the competition. If the two universities have the same number of **excellent students**, the competition ends in a draw.

An **excellent student** is a student that scored 90% or more in the exam.

Write an SQL query to report:

- "**New York University**" if New York University wins the competition.
- "**California University**" if California University wins the competition.
- "**No Winner**" if the competition ends in a draw.

The query result format is in the following example.

Example 1:

Input:

NewYork table:

student_id	score
1	90
2	87

California table:

student_id	score
2	89
3	88

Output:

winner
New York University

Explanation:

New York University has 1 excellent student, and California University has 0 excellent students.

Example 2:

Input:

NewYork table:

student_id	score
1	89
2	88

California table:

student_id	score
2	90
3	87

Output:

winner
California University

Explanation:

New York University has 0 excellent students, and California University has 1 excellent student.

Example 3:**Input:**

NewYork table:

student_id	score
1	89
2	90

California table:

student_id	score
2	87
3	99

Output:

winner
No Winner

Explanation:

Both New York University and California University have 1 excellent student.



```

SELECT
CASE
    WHEN (SELECT COUNT(STUDENT_ID) FROM NEWYORK WHERE SCORE >= 90 ) >
        (SELECT COUNT(STUDENT_ID) FROM CALIFORNIA WHERE SCORE >= 90 )
    THEN "New York University"
    WHEN (SELECT COUNT(STUDENT_ID) FROM CALIFORNIA WHERE SCORE >= 90 ) >
        (SELECT COUNT(STUDENT_ID) FROM NEWYORK WHERE SCORE >= 90 )
    THEN "California University"
    ELSE "No Winner"
END AS WINNER

```

2082. The Number of Rich Customers



Table: Store

Column Name	Type
bill_id	int
customer_id	int
amount	int

bill_id is the primary key for this table.

Each row contains information about the amount of one bill and the customer associated with it.



Write an SQL query to report the number of customers who had **at least one** bill with an amount **strictly greater** than 500 .

The query result format is in the following example.

Example 1:

Input:

Store table:

bill_id	customer_id	amount
6	1	549
8	1	834
4	2	394
11	3	657
13	3	257

Output:

rich_count
2

Explanation:

Customer 1 has two bills with amounts strictly greater than 500.

Customer 2 does not have any bills with an amount strictly greater than 500.

Customer 3 has one bill with an amount strictly greater than 500.

```

SELECT
COUNT(DISTINCT CUSTOMER_ID) AS RICH_COUNT
FROM STORE
WHERE AMOUNT > 500

```

2084. Drop Type 1 Orders for Customers With Type 0 Orders ↴

Table: Orders

Column Name	Type
order_id	int
customer_id	int
order_type	int

order_id is the primary key column for this table.

Each row of this table indicates the ID of an order, the ID of the customer who ordered it.

The orders could be of type 0 or type 1.

Write an SQL query to report all the orders based on the following criteria:

- If a customer has **at least one** order of type **0**, do **not** report any order of type **1** from that customer.
- Otherwise, report all the orders of the customer.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Orders table:

order_id	customer_id	order_type
1	1	0
2	1	0
11	2	0
12	2	1
21	3	1
22	3	0
31	4	1
32	4	1

Output:

order_id	customer_id	order_type
31	4	1
32	4	1
1	1	0
2	1	0
11	2	0
22	3	0

Explanation:

Customer 1 has two orders of type 0. We return both of them.

Customer 2 has one order of type 0 and one order of type 1. We only return the order.

Customer 3 has one order of type 0 and one order of type 1. We only return the order.

Customer 4 has two orders of type 1. We return both of them.



```

SELECT
ORDER_ID,
CUSTOMER_ID,
ORDER_TYPE
FROM ORDERS
GROUP BY CUSTOMER_ID, ORDER_ID
HAVING ORDER_TYPE = 0

UNION

SELECT
ORDER_ID,
CUSTOMER_ID,
ORDER_TYPE
FROM ORDERS
WHERE CUSTOMER_ID NOT IN
(SELECT CUSTOMER_ID FROM ORDERS WHERE ORDER_TYPE IN (0) GROUP BY CUSTOMER_ID, ORDER_ID)
GROUP BY CUSTOMER_ID, ORDER_ID

```

- An Awesome Approach.

```

(SELECT customer_id, MIN(order_type)
  FROM Orders
  GROUP BY customer_id)

{"headers": ["customer_id", "MIN(order_type)"], "values":
 [
 [1, 0],
 [2, 0],
 [3, 0],
 [4, 1]]}

SELECT * FROM Orders
WHERE (customer_id, order_type)
IN (SELECT customer_id, MIN(order_type)
    FROM Orders
    GROUP BY customer_id)

```

2112. The Airport With the Most Traffic ↗

Table: Flights

Column Name	Type
departure_airport	int
arrival_airport	int
flights_count	int

(`departure_airport`, `arrival_airport`) is the primary key column for this table.

Each row of this table indicates that there were `flights_count` flights that departed

Write an SQL query to report the ID of the airport with the **most traffic**. The airport with the most traffic is the airport that has the largest total number of flights that either departed from or arrived at the airport. If there is more than one airport with the most traffic, report them all.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

Flights table:

departure_airport	arrival_airport	flights_count
1	2	4
2	1	5
2	4	5

Output:

airport_id
2

Explanation:

Airport 1 was engaged with 9 flights (4 departures, 5 arrivals).

Airport 2 was engaged with 14 flights (10 departures, 4 arrivals).

Airport 4 was engaged with 5 flights (5 arrivals).

The airport with the most traffic is airport 2.

Example 2:

Input:

Flights table:

departure_airport	arrival_airport	flights_count
1	2	4
2	1	5
3	4	5
4	3	4
5	6	7

Output:

airport_id
1
2
3
4

Explanation:

Airport 1 was engaged with 9 flights (4 departures, 5 arrivals).

Airport 2 was engaged with 9 flights (5 departures, 4 arrivals).

Airport 3 was engaged with 9 flights (5 departures, 4 arrivals).

Airport 4 was engaged with 9 flights (4 departures, 5 arrivals).

Airport 5 was engaged with 7 flights (7 departures).

Airport 6 was engaged with 7 flights (7 arrivals).

The airports with the most traffic are airports 1, 2, 3, and 4.

```

WITH TOTAL_TRAFFIC AS
(
    SELECT A.AIRPORT, SUM(TRAFFIC) AS TRAFFIC
    FROM
    (
        SELECT
            DEPARTURE_AIRPORT AS AIRPORT,
            SUM(FLIGHTS_COUNT) AS TRAFFIC
        FROM FLIGHTS
        GROUP BY DEPARTURE_AIRPORT

        UNION

        SELECT
            ARRIVAL_AIRPORT AS AIRPORT,
            SUM(FLIGHTS_COUNT) AS TRAFFIC
        FROM FLIGHTS
        GROUP BY ARRIVAL_AIRPORT
    ) AS A
    GROUP BY A.AIRPORT
)

SELECT AIRPORT AS AIRPORT_ID
FROM TOTAL_TRAFFIC
WHERE TRAFFIC IN (SELECT MAX(TRAFFIC) FROM TOTAL_TRAFFIC)

```

2142. The Number of Passengers in Each Bus I ↗ ▾

Table: Buses

Column Name	Type
bus_id	int
arrival_time	int

bus_id is the primary key column for this table.

Each row of this table contains information about the arrival time of a bus at the L
No two buses will arrive at the same time.

Table: Passengers

Column Name	Type
passenger_id	int
arrival_time	int

passenger_id is the primary key column for this table.

Each row of this table contains information about the arrival time of a passenger at

Buses and passengers arrive at the LeetCode station. If a bus arrives at the station at time t_{bus} and a passenger arrived at time $t_{passenger}$ where $t_{passenger} \leq t_{bus}$ and the passenger did not catch any bus, the passenger will use that bus.

Write an SQL query to report the number of users that used each bus.

Return the result table ordered by `bus_id` in **ascending order**.

The query result format is in the following example.

Example 1:

Input:

Buses table:

bus_id	arrival_time
1	2
2	4
3	7

Passengers table:

passenger_id	arrival_time
11	1
12	5
13	6
14	7

Output:

bus_id	passengers_cnt
1	1
2	0
3	3

Explanation:

- Passenger 11 arrives at time 1.
- Bus 1 arrives at time 2 and collects passenger 11.
- Bus 2 arrives at time 4 and does not collect any passengers.
- Passenger 12 arrives at time 5.
- Passenger 13 arrives at time 6.
- Passenger 14 arrives at time 7.
- Bus 3 arrives at time 7 and collects passengers 12, 13, and 14.

- Below Code Doesn't Work As there are many Bugs in it...

```

WITH CTE AS
(
    SELECT
        A.PASSENGER_ID,
        MIN(A.BUS_ID) AS BUS_ID
    FROM
    (
        SELECT
            B.BUS_ID,
            P.PASSENGER_ID
        FROM BUSES AS B JOIN PASSENGERS AS P
        ON B.ARRIVAL_TIME >= P.ARRIVAL_TIME
        ORDER BY B.BUS_ID
    ) AS A
    GROUP BY A.PASSENGER_ID
)

SELECT B.BUS_ID, CASE
WHEN C.BUS_ID IS NULL THEN 0 ELSE COUNT(C.BUS_ID) END AS PASSENGERS_CNT FROM BUSES AS B
LEFT JOIN CTE AS C ON B.BUS_ID = C.BUS_ID GROUP BY B.BUS_ID

```

* The Following Code Works Perfectly Fine and i have resolved all the issues from it.

```

WITH CTE AS ( SELECT A.PASSENGER_ID, MIN(A.ARRIVAL_TIME) AS ARRIVAL_TIME FROM ( SELECT
B.BUS_ID, B.ARRIVAL_TIME, P.PASSENGER_ID, P.ARRIVAL_TIME AS PAT FROM (SELECT * FROM BUSES
ORDER BY ARRIVAL_TIME) AS B LEFT JOIN (SELECT * FROM PASSENGERS ORDER BY ARRIVAL_TIME) AS P
ON B.ARRIVAL_TIME >= P.ARRIVAL_TIME ORDER BY B.ARRIVAL_TIME ) AS A GROUP BY A.PASSENGER_ID )
SELECT D.BUS_ID, SUM(VALS) AS PASSENGERS_CNT FROM ( SELECT B.BUS_ID, CASE WHEN
C.PASSENGER_ID IS NULL THEN 0 ELSE 1 END AS VAIS FROM BUSES AS B LEFT JOIN CTE AS C ON
B.ARRIVAL_TIME = C.ARRIVAL_TIME ) AS D GROUP BY D.BUS_ID ORDER BY BUS_ID

```

2159. Order Two Columns Independently ↗ ▾

Table: Data

Column Name	Type
first_col	int
second_col	int

There is no primary key for this table and it may contain duplicates.

Write an SQL query to independently:

- order first_col in **ascending order**.
- order second_col in **descending order**.

The query result format is in the following example.

Example 1:

Input:

Data table:

first_col	second_col
4	2
2	3
3	1
1	4

Output:

first_col	second_col
1	4
2	3
3	2
4	1

```

WITH COL1 AS
(
    SELECT FIRST_COL, ROW_NUMBER() OVER() AS SECOND_COL
    FROM DATA
    ORDER BY FIRST_COL
) ,
COL2 AS (
    SELECT
        ROW_NUMBER() OVER() AS FIRST_COL ,SECOND_COL
    FROM DATA
    ORDER BY SECOND_COL DESC
)
SELECT COL1.FIRST_COL, COL2.SECOND_COL
FROM COL1 JOIN COL2
ON COL1.SECOND_COL = COL2.FIRST_COL

```

2196. Create Binary Tree From Descriptions ↗ ▾

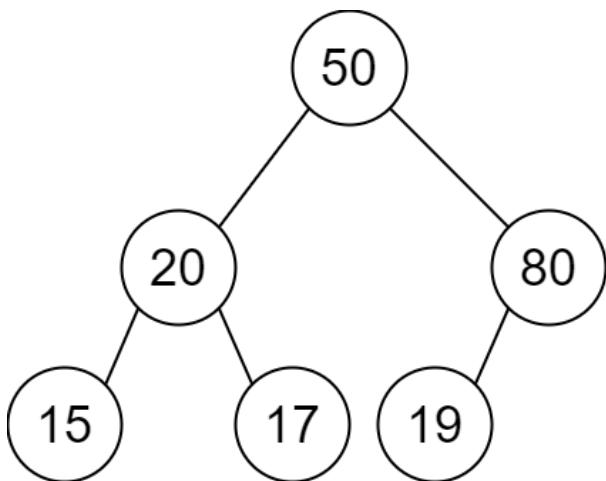
You are given a 2D integer array `descriptions` where `descriptions[i] = [parenti, childi, isLefti]` indicates that `parenti` is the **parent** of `childi` in a **binary** tree of **unique** values. Furthermore,

- If `isLefti == 1`, then `childi` is the left child of `parenti`.
- If `isLefti == 0`, then `childi` is the right child of `parenti`.

Construct the binary tree described by `descriptions` and return *its root*.

The test cases will be generated such that the binary tree is **valid**.

Example 1:

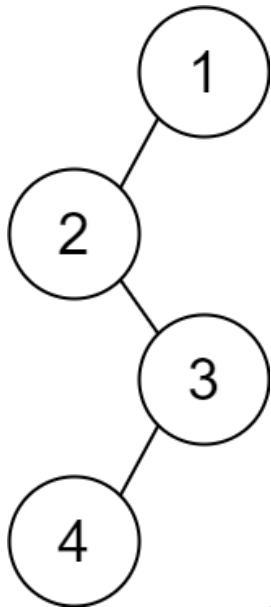


Input: descriptions = [[20,15,1],[20,17,0],[50,20,1],[50,80,0],[80,19,1]]

Output: [50,20,80,15,17,19]

Explanation: The root node is the node with value 50 since it has no parent.
The resulting binary tree is shown in the diagram.

Example 2:



Input: descriptions = [[1,2,1],[2,3,0],[3,4,1]]

Output: [1,2,null,null,3,4]

Explanation: The root node is the node with value 1 since it has no parent.
The resulting binary tree is shown in the diagram.

Constraints:

- $1 \leq \text{descriptions.length} \leq 10^4$
- $\text{descriptions}[i].length == 3$
- $1 \leq \text{parent}_i, \text{child}_i \leq 10^5$
- $0 \leq \text{isLeft}_i \leq 1$
- The binary tree described by `descriptions` is valid.

```

# 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 createBinaryTree(self, descriptions: List[List[int]]) -> Optional[TreeNode]:
        l1 = {}
        nodes = {}
        # Generating the nodes O(N) and assigning its Value.
        for node in descriptions:
            if (node[0] not in nodes):
                nodes[node[0]] = TreeNode(node[0])
            if (node[1] not in nodes):
                nodes[node[1]] = TreeNode(node[1])

        l1[node[0]] = 1

        # Connect all the nodes based on their description.
        # O(N)- Tc
        root = None
        for val in descriptions:
            if val[1] in l1:
                del l1[val[1]]
                ptr = nodes[val[0]]
                if val[2]:
                    ptr.left = nodes[val[1]]
                else:
                    ptr.right = nodes[val[1]]
        # assign the root node.
        root = nodes[list(l1.keys())[0]]
        return root

```

2175. The Change in Global Rankings ↗

Table: TeamPoints

Column Name	Type
team_id	int
name	varchar
points	int

team_id is the primary key for this table.

Each row of this table contains the ID of a national team, the name of the country it

Table: PointsChange

Column Name	Type
team_id	int
points_change	int

team_id is the primary key for this table.

Each row of this table contains the ID of a national team and the change in its points_change can be:

- 0: indicates no change in points.
- positive: indicates an increase in points.
- negative: indicates a decrease in points.

Each team_id that appears in TeamPoints will also appear in this table.

The **global ranking** of a national team is its rank after sorting all the teams by their points **in descending order**. If two teams have the same points, we break the tie by sorting them by their name **in lexicographical order**.

The points of each national team should be updated based on its corresponding points_change value.

Write an SQL query to calculate the change in the global rankings after updating each team's points.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:

TeamPoints table:

team_id	name	points
3	Algeria	1431
1	Senegal	2132
2	New Zealand	1402
4	Croatia	1817

PointsChange table:

team_id	points_change
3	399
2	0
4	13
1	-22

Output:

team_id	name	rank_diff
1	Senegal	0
4	Croatia	-1
3	Algeria	1
2	New Zealand	0

Explanation:

The global rankings were as follows:

team_id	name	points	rank
1	Senegal	2132	1
4	Croatia	1817	2
3	Algeria	1431	3
2	New Zealand	1402	4

After updating the points of each team, the rankings became the following:

team_id	name	points	rank
1	Senegal	2110	1
3	Algeria	1830	2
4	Croatia	1830	3
2	New Zealand	1402	4

Since after updating the points Algeria and Croatia have the same points, they are ranked equal. Senegal lost 22 points but their rank did not change.

Croatia gained 13 points but their rank decreased by one.

Algeria gained 399 points and their rank increased by one.
New Zealand did not gain or lose points and their rank did not change.

- Partially Working where more logic needs to be added.

```
WITH RANK1 AS
(
    SELECT * ,
    DENSE_RANK() OVER(ORDER BY POINTS DESC) AS POSITION
    FROM TEAMPOINTS
),
RANK2 AS
(
    SELECT *,  

    DENSE_RANK() OVER(ORDER BY POINTS DESC) AS POSITION
    FROM
    (
        SELECT
            TP.TEAM_ID,
            TP.NAME,
            (TP.POINTS - PC.POINTS_CHANGE ) AS POINTS
        FROM TEAMPOINTS AS TP JOIN POINTSCHANGE AS PC
        ON TP.TEAM_ID = PC.TEAM_ID
    ) AS A
)
```

SELECT R1.TEAM_ID, R1.NAME, R1.POSITION - R2.POSITION AS RANK_DIFF FROM RANK1 AS R1 JOIN RANK2 AS R2 ON R1.TEAM_ID = R2.TEAM_ID GROUP BY R1.TEAM_ID, R1.NAME

* Working Code

```
WITH RANK2 AS ( SELECT * , DENSE_RANK() OVER(ORDER BY POINTS DESC, NAME ASC) AS AFTER_RANK
FROM (
SELECT TP.TEAM_ID, TP.NAME, (TP.POINTS + PC.POINTS_CHANGE ) AS POINTS FROM TEAMPOINTS AS TP
JOIN POINTSCHANGE AS PC ON TP.TEAM_ID = PC.TEAM_ID ) AS A ), RANK1 AS ( SELECT * ,
DENSE_RANK() OVER(ORDER BY POINTS DESC, NAME ASC) AS BEFORE_RANK FROM TEAMPOINTS )
```

```
SELECT R1.TEAM_ID, R1.NAME, CAST(R1.BEFORE_RANK AS SIGNED) - CAST(R2.AFTER_RANK AS SIGNED)
AS RANK_DIFF FROM RANK1 AS R1 LEFT JOIN RANK2 AS R2 ON R1.TEAM_ID = R2.TEAM_ID GROUP BY
R1.TEAM_ID, R1.NAME
```

2205. The Number of Users That Are Eligible for Discount ↴

Table: Purchases

Column Name	Type
user_id	int
time_stamp	datetime
amount	int

(user_id, time_stamp) is the primary key for this table.

Each row contains information about the purchase time and the amount paid for the us

A user is eligible for a discount if they had a purchase in the inclusive interval of time [startDate, endDate] with at least minAmount amount. To convert the dates to times, both dates should be considered as the **start** of the day (i.e., endDate = 2022-03-05 should be considered as the time 2022-03-05 00:00:00).

Write an SQL query to report the number of users that are eligible for a discount.

The query result format is in the following example.

Example 1:

Input:

Purchases table:

user_id	time_stamp	amount
1	2022-04-20 09:03:00	4416
2	2022-03-19 19:24:02	678
3	2022-03-18 12:03:09	4523
3	2022-03-30 09:43:42	626

startDate = 2022-03-08, endDate = 2022-03-20, minAmount = 1000

Output:

user_cnt
1

Explanation:

Out of the three users, only User 3 is eligible for a discount.

- User 1 had one purchase with at least minAmount amount, but not within the time interval.
- User 2 had one purchase within the time interval, but with less than minAmount amount.
- User 3 is the only user who had a purchase that satisfies both conditions.

```
CREATE FUNCTION getUserIDs(startDate DATE, endDate DATE, minAmount INT) RETURNS INT
BEGIN
    RETURN (
        SELECT COUNT(DISTINCT USER_ID)
        FROM PURCHASES
        WHERE TIME_STAMP BETWEEN TIMESTAMP(startdate) AND TIMESTAMP(enddate)
        AND AMOUNT >= minamount
    );
END
```

2228. Users With Two Purchases Within Seven Days



Table: Purchases

Column Name	Type
purchase_id	int
user_id	int
purchase_date	date

purchase_id is the primary key for this table.

This table contains logs of the dates that users purchased from a certain retailer.

Write an SQL query to report the IDs of the users that made any two purchases **at most** 7 days apart.

Return the result table ordered by user_id .

The query result format is in the following example.

Example 1:

Input:

Purchases table:

purchase_id	user_id	purchase_date
4	2	2022-03-13
1	5	2022-02-11
3	7	2022-06-19
6	2	2022-03-20
5	7	2022-06-19
2	2	2022-06-08

Output:

user_id
2
7

Explanation:

User 2 had two purchases on 2022-03-13 and 2022-03-20. Since the second purchase is 7 days apart, User 2 is included in the result.

User 5 had only 1 purchase, so they are not included.

- Temporary code which doesn't work

```

SELECT
A.PURCHASE_ID,
A.USER_ID,
A.PURCHASE_DATE
FROM
(
  SELECT
  *,
  DENSE_RANK()
  OVER(PARTITION BY USER_ID ORDER BY PURCHASE_DATE ASC ) AS CRANK
  FROM PURCHASES
) AS A
WHERE A.CRANK IN (1,2)

```

```

## 2230. The Users That Are Eligible for Discount ↗ ▾

Table: Purchases

| Column Name | Type     |
|-------------|----------|
| user_id     | int      |
| time_stamp  | datetime |
| amount      | int      |

(user\_id, time\_stamp) is the primary key for this table.  
Each row contains information about the purchase time and the amount paid for the us

A user is eligible for a discount if they had a purchase in the inclusive interval of time [startDate, endDate] with at least minAmount amount. To convert the dates to times, both dates should be considered as the **start** of the day (i.e., endDate = 2022-03-05 should be considered as the time 2022-03-05 00:00:00 ).

Write an SQL query to report the IDs of the users that are eligible for a discount.

Return the result table ordered by user\_id .

The query result format is in the following example.

### Example 1:

**Input:**

Purchases table:

| user_id | time_stamp          | amount |
|---------|---------------------|--------|
| 1       | 2022-04-20 09:03:00 | 4416   |
| 2       | 2022-03-19 19:24:02 | 678    |
| 3       | 2022-03-18 12:03:09 | 4523   |
| 3       | 2022-03-30 09:43:42 | 626    |

startDate = 2022-03-08, endDate = 2022-03-20, minAmount = 1000

**Output:**

| user_id |
|---------|
| 3       |

**Explanation:**

Out of the three users, only User 3 is eligible for a discount.

- User 1 had one purchase with at least minAmount amount, but not within the time interval.
- User 2 had one purchase within the time interval, but with less than minAmount amount.
- User 3 is the only user who had a purchase that satisfies both conditions.

**Important Note:** This problem is basically the same as The Number of Users That Are Eligible for Discount (<https://leetcode.com/problems/the-number-of-users-that-are-eligible-for-discount/>).

```
CREATE PROCEDURE getUserIDs(startDate DATE, endDate DATE, minAmount INT)
BEGIN
 SELECT DISTINCT USER_ID
 FROM PURCHASES
 WHERE TIME_STAMP BETWEEN TIMESTAMP(startDate) AND TIMESTAMP(endDate)
 AND AMOUNT >= minAmount ORDER BY USER_ID;
END
```

## 2238. Number of Times a Driver Was a Passenger ↗

Table: Rides

| Column Name  | Type |
|--------------|------|
| ride_id      | int  |
| driver_id    | int  |
| passenger_id | int  |

ride\_id is the primary key for this table.

Each row of this table contains the ID of the driver and the ID of the passenger that

Note that driver\_id != passenger\_id.

◀ ▶

Write an SQL query to report the ID of each driver and the number of times they were a passenger.

Return the result table in **any order**.

The query result format is in the following example.

### Example 1:

#### Input:

Rides table:

| ride_id | driver_id | passenger_id |
|---------|-----------|--------------|
| 1       | 7         | 1            |
| 2       | 7         | 2            |
| 3       | 11        | 1            |
| 4       | 11        | 7            |
| 5       | 11        | 7            |
| 6       | 11        | 3            |

#### Output:

| driver_id | cnt |
|-----------|-----|
| 7         | 2   |
| 11        | 0   |

#### Explanation:

There are two drivers in all the given rides: 7 and 11.

The driver with ID = 7 was a passenger two times.

The driver with ID = 11 was never a passenger.

- Wrong Way to Solve the Problem.

```

WITH DRIVERS AS
(
SELECT
DISTINCT DRIVER_ID
FROM RIDES
)

SELECT DRIVER_ID,
(SELECT COUNT(*) FROM RIDES WHERE PASSENGER_ID = DRIVER_ID) AS CNT
FROM DRIVERS

```

- Using CTE to solve the Problem in a well versed way by using left joins
- This might not be an optimized way but must see.

```

WITH DRIVERS AS
(
SELECT
DISTINCT DRIVER_ID
FROM RIDES
)

SELECT
D.DRIVER_ID,
CASE
 WHEN R.RIDE_ID IS NULL THEN 0
 ELSE COUNT(*)
END AS CNT
FROM DRIVERS AS D LEFT JOIN RIDES AS R
ON D.DRIVER_ID = R.PASSENGER_ID
GROUP BY D.DRIVER_ID

```

## 2298. Tasks Count in the Weekend ↗

Table: Tasks

| Column Name | Type |
|-------------|------|
| task_id     | int  |
| assignee_id | int  |
| submit_date | date |

task\_id is the primary key for this table.

Each row in this table contains the ID of a task, the id of the assignee, and the su

Write an SQL query to report:

- the number of the tasks that were submitted during the weekend (Saturday, Sunday) as `weekend_cnt`, and
- the number of the tasks that were submitted during the working days as `working_cnt`.

Return the result table in **any order**.

The query result format is shown in the following example.

### Example 1:

#### Input:

Tasks table:

| task_id | assignee_id | submit_date |
|---------|-------------|-------------|
| 1       | 1           | 2022-06-13  |
| 2       | 6           | 2022-06-14  |
| 3       | 6           | 2022-06-15  |
| 4       | 3           | 2022-06-18  |
| 5       | 5           | 2022-06-19  |
| 6       | 7           | 2022-06-19  |

#### Output:

| weekend_cnt | working_cnt |
|-------------|-------------|
| 3           | 3           |

#### Explanation:

- Task 1 was submitted on Monday.  
 Task 2 was submitted on Tuesday.  
 Task 3 was submitted on Wednesday.  
 Task 4 was submitted on Saturday.  
 Task 5 was submitted on Sunday.  
 Task 6 was submitted on Sunday.  
 3 tasks were submitted during the weekend.  
 3 tasks were submitted during the working days.

- Solving The Problem Using COUNT + DAYNAME Function

```

SELECT
(
 SELECT
 COUNT(*)
 FROM TASKS
 WHERE DAYNAME(SUBMIT_DATE) IN ('Saturday', 'Sunday')
) AS WEEKEND_CNT,
(
 SELECT
 COUNT(*)
 FROM TASKS
 WHERE DAYNAME(SUBMIT_DATE) NOT IN ('Saturday', 'Sunday')
) AS WORKING_CNT

```

## 2314. The First Day of the Maximum Recorded Degree in Each City ↗

Table: Weather

| Column Name | Type |
|-------------|------|
| city_id     | int  |
| day         | date |
| degree      | int  |

(city\_id, day) is the primary key for this table.

Each row in this table contains the degree of the weather of a city on a certain day.  
All the degrees are recorded in the year 2022.

◀ ▶

Write an SQL query to report the day that has the maximum recorded degree in each city. If the maximum degree was recorded for the same city multiple times, return the earliest day among them.

Return the result table ordered by `city_id` in **ascending order**.

The query result format is shown in the following example.

### Example 1:

**Input:**

Weather table:

| city_id | day        | degree |
|---------|------------|--------|
| 1       | 2022-01-07 | -12    |
| 1       | 2022-03-07 | 5      |
| 1       | 2022-07-07 | 24     |
| 2       | 2022-08-07 | 37     |
| 2       | 2022-08-17 | 37     |
| 3       | 2022-02-07 | -7     |
| 3       | 2022-12-07 | -6     |

**Output:**

| city_id | day        | degree |
|---------|------------|--------|
| 1       | 2022-07-07 | 24     |
| 2       | 2022-08-07 | 37     |
| 3       | 2022-12-07 | -6     |

**Explanation:**

For city 1, the maximum degree was recorded on 2022-07-07 with 24 degrees.

For city 1, the maximum degree was recorded on 2022-08-07 and 2022-08-17 with 37 deg

For city 3, the maximum degree was recorded on 2022-12-07 with -6 degrees.

```

SELECT
CITY_ID,
MIN(DAY) AS DAY,
DEGREE
FROM WEATHER
WHERE (CITY_ID, DEGREE) IN
(
 SELECT
 CITY_ID,
 MAX(DEGREE) AS DEGREE
 FROM WEATHER
 GROUP BY CITY_ID
)
GROUP BY CITY_ID, DEGREE
ORDER BY CITY_ID

```

## 2324. Product Sales Analysis IV ↗



Table: Sales

| Column Name | Type |
|-------------|------|
| sale_id     | int  |
| product_id  | int  |
| user_id     | int  |
| quantity    | int  |

sale\_id is the primary key of this table.

product\_id is a foreign key to Product table.

Each row of this table shows the ID of the product and the quantity purchased by a user.

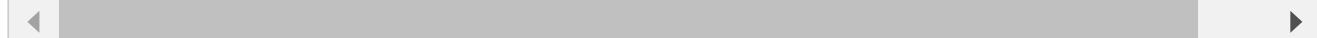


Table: Product

| Column Name | Type |
|-------------|------|
| product_id  | int  |
| price       | int  |

product\_id is the primary key of this table.

Each row of this table indicates the price of each product.

Write an SQL query that reports for each user the product id on which the user spent the most money. In case the same user spent the most money on two or more products, report all of them.

Return the resulting table in **any order**.

The query result format is in the following example.

### Example 1:

**Input:**

Sales table:

| sale_id | product_id | user_id | quantity |
|---------|------------|---------|----------|
| 1       | 1          | 101     | 10       |
| 2       | 3          | 101     | 7        |
| 3       | 1          | 102     | 9        |
| 4       | 2          | 102     | 6        |
| 5       | 3          | 102     | 10       |
| 6       | 1          | 102     | 6        |

Product table:

| product_id | price |
|------------|-------|
| 1          | 10    |
| 2          | 25    |
| 3          | 15    |

**Output:**

| user_id | product_id |
|---------|------------|
| 101     | 3          |
| 102     | 1          |
| 102     | 2          |
| 102     | 3          |

**Explanation:**

User 101:

- Spent  $10 * 10 = 100$  on product 1.
- Spent  $7 * 15 = 105$  on product 3.

User 101 spent the most money on product 3.

User 102:

- Spent  $(9 + 7) * 10 = 150$  on product 1.
- Spent  $6 * 25 = 150$  on product 2.
- Spent  $10 * 15 = 150$  on product 3.

User 102 spent the most money on products 1, 2, and 3.

- Solution Using Rank

```

SELECT
B.USER_ID,
B.PRODUCT_ID
FROM
(
 SELECT
 A.USER_ID,
 A.PRODUCT_ID,
 DENSE_RANK()
 OVER(PARTITION BY A.USER_ID
 ORDER BY A.AMT DESC) AS CRANK
 FROM
 (
 SELECT
 S.USER_ID,
 S.PRODUCT_ID,
 SUM(S.QUANTITY * P.PRICE) AS AMT
 FROM PRODUCT AS P LEFT JOIN SALES AS S
 ON P.PRODUCT_ID = S.PRODUCT_ID
 WHERE S.PRODUCT_ID IS NOT NULL
 GROUP BY S.USER_ID, S.PRODUCT_ID
) AS A
) AS B
WHERE B.CRANK = 1

```

## 2329. Product Sales Analysis V ↴

Table: Sales

| Column Name | Type |
|-------------|------|
| sale_id     | int  |
| product_id  | int  |
| user_id     | int  |
| quantity    | int  |

sale\_id is the primary key of this table.

product\_id is a foreign key to Product table.

Each row of this table shows the ID of the product and the quantity purchased by a user.

Table: Product

| Column Name | Type |
|-------------|------|
| product_id  | int  |
| price       | int  |

product\_id is the primary key of this table.

Each row of this table indicates the price of each product.

Write an SQL query that reports the spending of each user.

Return the resulting table ordered by spending in **descending order**. In case of a tie, order them by user\_id in ascending order.

The query result format is in the following example.

**Example 1:**

**Input:**

Sales table:

| sale_id | product_id | user_id | quantity |
|---------|------------|---------|----------|
| 1       | 1          | 101     | 10       |
| 2       | 2          | 101     | 1        |
| 3       | 3          | 102     | 3        |
| 4       | 3          | 102     | 2        |
| 5       | 2          | 103     | 3        |

Product table:

| product_id | price |
|------------|-------|
| 1          | 10    |
| 2          | 25    |
| 3          | 15    |

**Output:**

| user_id | spending |
|---------|----------|
| 101     | 125      |
| 102     | 75       |
| 103     | 75       |

**Explanation:**User 101 spent  $10 * 10 + 1 * 25 = 125$ .User 102 spent  $3 * 15 + 2 * 15 = 75$ .User 103 spent  $3 * 25 = 75$ .

Users 102 and 103 spent the same amount and we break the tie by their ID while user

- Working Solution Using LEFT Join

```
SELECT S.USER_ID, SUM(S.QUANTITY*P.PRICE) AS SPENDING
FROM PRODUCT AS P LEFT JOIN SALES S
ON P.PRODUCT_ID = S.PRODUCT_ID
WHERE S.PRODUCT_ID IS NOT NULL
GROUP BY S.USER_ID
ORDER BY SPENDING DESC, USER_ID ASC
```

## 2339. All the Matches of the League ↗

Table: Teams

```
+-----+-----+
| Column Name | Type |
+-----+-----+
| team_name | varchar |
+-----+-----+
team_name is the primary key of this table.
Each row of this table shows the name of a team.
```

Write an SQL query that reports all the possible matches of the league. Note that every two teams play two matches with each other, with one team being the `home_team` once and the other time being the `away_team`.

Return the result table in **any order**.

The query result format is in the following example.

### Example 1:

#### **Input:**

Teams table:

```
+-----+
| team_name |
+-----+
| Leetcode FC |
| Ahly SC |
| Real Madrid |
+-----+
```

#### **Output:**

```
+-----+-----+
| home_team | away_team |
+-----+-----+
Real Madrid	Leetcode FC
Real Madrid	Ahly SC
Leetcode FC	Real Madrid
Leetcode FC	Ahly SC
Ahly SC	Real Madrid
Ahly SC	Leetcode FC
+-----+-----+
```

**Explanation:** All the matches of the league are shown in the table.

```
SELECT
T1.TEAM_NAME AS HOME_TEAM,
T2.TEAM_NAME AS AWAY_TEAM
FROM TEAMS AS T1 JOIN TEAMS AS T2
ON T1.TEAM_NAME != T2.TEAM_NAME
```

## 2346. Compute the Rank as a Percentage ↗



Table: Students

| Column Name   | Type |
|---------------|------|
| student_id    | int  |
| department_id | int  |
| mark          | int  |

student\_id is the primary key of this table.

Each row of this table indicates a student's ID, the ID of the department in which t



Write an SQL query that reports the rank of each student in their department as a percentage, where the rank as a percentage is computed using the following formula:  $(\text{student\_rank\_in\_the\_department} - 1) * 100 / (\text{the\_number\_of\_students\_in\_the\_department} - 1)$ . The percentage should be **rounded to 2 decimal places**. `student_rank_in_the_department` is determined by **descending** `mark`, such that the student with the highest `mark` is `rank 1`. If two students get the same mark, they also get the same rank.

Return the result table in **any order**.

The query result format is in the following example.

### Example 1:

**Input:**

Students table:

| student_id | department_id | mark |
|------------|---------------|------|
| 2          | 2             | 650  |
| 8          | 2             | 650  |
| 7          | 1             | 920  |
| 1          | 1             | 610  |
| 3          | 1             | 530  |

**Output:**

| student_id | department_id | percentage |
|------------|---------------|------------|
| 7          | 1             | 0.0        |
| 1          | 1             | 50.0       |
| 3          | 1             | 100.0      |
| 2          | 2             | 0.0        |
| 8          | 2             | 0.0        |

**Explanation:**

For Department 1:

- Student 7: percentage =  $(1 - 1) * 100 / (3 - 1) = 0.0$
- Student 1: percentage =  $(2 - 1) * 100 / (3 - 1) = 50.0$
- Student 3: percentage =  $(3 - 1) * 100 / (3 - 1) = 100.0$

For Department 2:

- Student 2: percentage =  $(1 - 1) * 100 / (2 - 1) = 0.0$
- Student 8: percentage =  $(1 - 1) * 100 / (2 - 1) = 0.0$

```

SELECT
STUDENT_ID,
DEPARTMENT_ID,
ROUND(
CASE
 WHEN
 (RANK() OVER(PARTITION BY DEPARTMENT_ID ORDER BY MARK DESC) - 1) * 100 / (COUNT(*) OVER(PARTITION BY DEPARTMENT_ID) - 1) IS NOT NULL
 THEN (RANK() OVER(PARTITION BY DEPARTMENT_ID ORDER BY MARK DESC) - 1) * 100 /
 (COUNT(*) OVER(PARTITION BY DEPARTMENT_ID) - 1)
 ELSE 0.0
END, 2)
AS PERCENTAGE
FROM STUDENTS

```