

#!/bin/python3

import math

import os

import random

import re

import sys

class DoublyLinkedListNode:

    def \_\_init\_\_(self, node\_data):

        self.data = node\_data

        self.next = None

        self.prev = None

class DoublyLinkedList:

    def \_\_init\_\_(self):

        self.head = None

        self.tail = None

    def insert\_node(self, node\_data):

        node = DoublyLinkedListNode(node\_data)

        if not self.head:

            self.head = node

        else:

            self.tail.next = node

            node.prev = self.tail

        self.tail = node

def print\_doubly\_linked\_list(node, sep, fptr):

    while node:

        fptr.write(str(node.data))

        node = node.next

        if node:

            fptr.write(sep)

#

# Complete the 'sortedInsert' function below.

#

# The function is expected to return an INTEGER\_DOUBLY\_LINKED\_LIST.

# The function accepts following parameters:

#  1. INTEGER\_DOUBLY\_LINKED\_LIST llist

#  2. INTEGER data

#

#

# For your reference:

#

# DoublyLinkedListNode:

#     int data

#     DoublyLinkedListNode next

#     DoublyLinkedListNode prev

#

#

def sortedInsert(llist, data):

    new\_node = DoublyLinkedListNode(data)

    # Case 1: Empty list

    if llist is None:

        return new\_node

    # Case 2: Insert at the beginning

    if data < llist.data:

        new\_node.next = llist

        llist.prev = new\_node

        return new\_node

    # Case 3: Traverse to find the position

    current = llist

    while current.next and current.data < data:

        current = current.next

    if current.data >= data:

        # Insert before current

        prev\_node = current.prev

        prev\_node.next = new\_node

        new\_node.prev = prev\_node

        new\_node.next = current

        current.prev = new\_node

    else:

        # Insert after current (end of list)

        current.next = new\_node

        new\_node.prev = current

    return llist

if \_\_name\_\_ == '\_\_main\_\_':

    fptr = open(os.environ['OUTPUT\_PATH'], 'w')

    t = int(input())

    for t\_itr in range(t):

        llist\_count = int(input())

        llist = DoublyLinkedList()

        for \_ in range(llist\_count):

            llist\_item = int(input())

            llist.insert\_node(llist\_item)

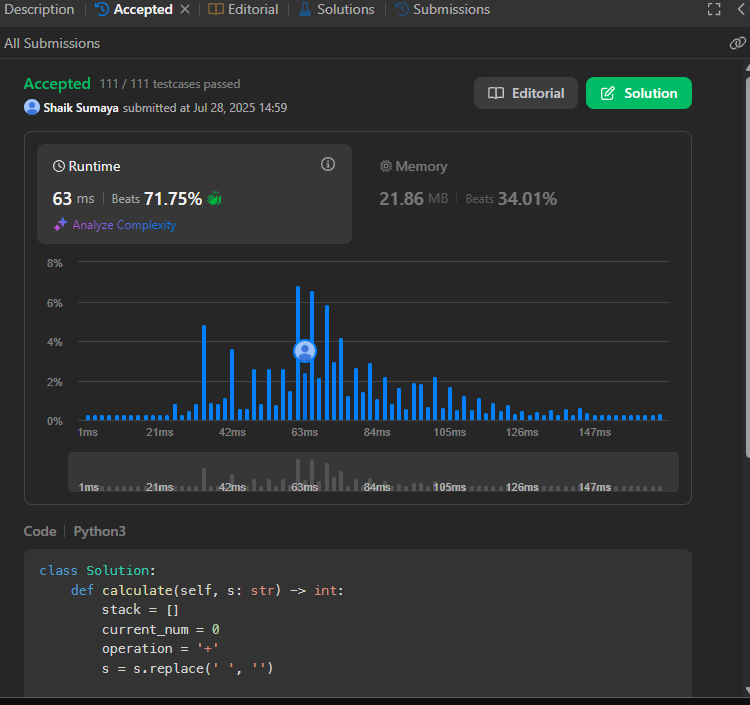
        data = int(input())

        llist1 = sortedInsert(llist.head, data)

        print\_doubly\_linked\_list(llist1, ' ', fptr)

        fptr.write('\n')

    fptr.close()



class Solution:

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

stack = []

current\_num = 0

operation = '+'

s = s.replace(' ', '')

for i in range(len(s)):

char = s[i]

if char.isdigit():

current\_num = current\_num \* 10 + int(char)

# If we reach an operator or end of string

if char in '+-\*/' or i == len(s) - 1:

if operation == '+':

stack.append(current\_num)

elif operation == '-':

stack.append(-current\_num)

elif operation == '\*':

stack.append(stack.pop() \* current\_num)

elif operation == '/':

prev = stack.pop()

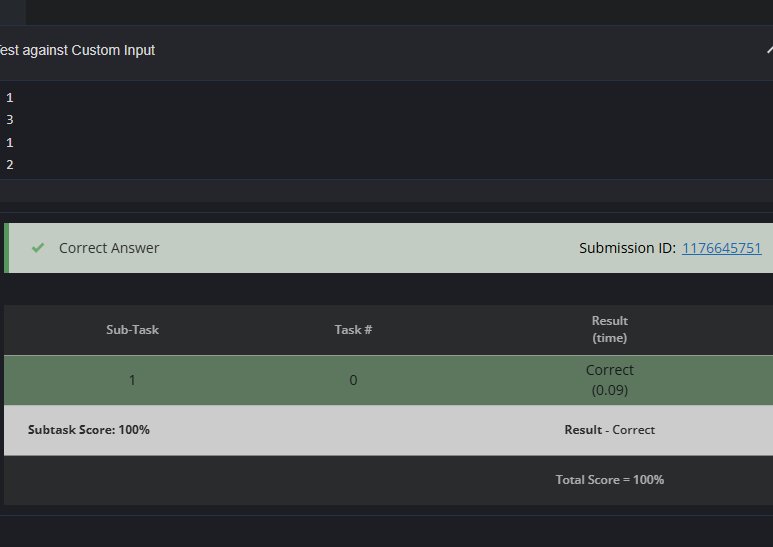
# Division that truncates towards zero

stack.append(int(prev / current\_num))

operation = char

current\_num = 0

return sum(stack)



# cook your dish here

def find\_missing\_doll(dolls):

result = 0

for doll in dolls:

result ^= doll

return result

# Read input

T = int(input())

for \_ in range(T):

N = int(input())

dolls = []

for \_ in range(N):

doll\_type = int(input())

dolls.append(doll\_type)

print(find\_missing\_doll(dolls))