□ برمجة □□□□□□: حل المسائل عبر الإنترنت

انه نستخدم نظام التقييم عبر الإنترنت لحل المسائل. إذا كنت تجيد اللغة الإنجليزية، يمكنك استخدام Codeforces و LeetCode. أما إذا كنت تفضل اللغة الصينية، فيمكنك استخدام □□□□□ و □□□□. انه نستخدم LeetCode. لقد قمت بحل 10 مسائل. وفي المسائل الأخيرة، استخدمت عدة طرق لتحسين كفاءة البرنامج من التغلب على 10□ من الإجابات إلى التغلب على 99□.

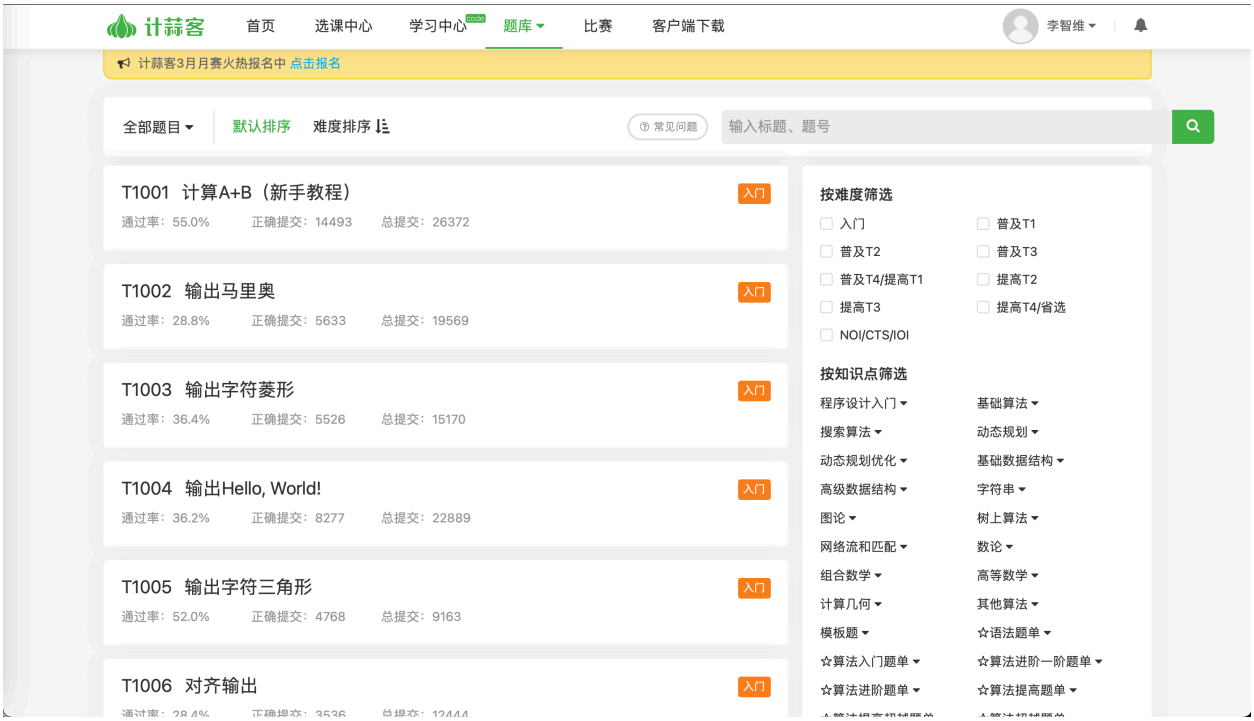
□□□□□□ 1: □□

1480. مجموع تشغيل مصفوفة أحادية البعد

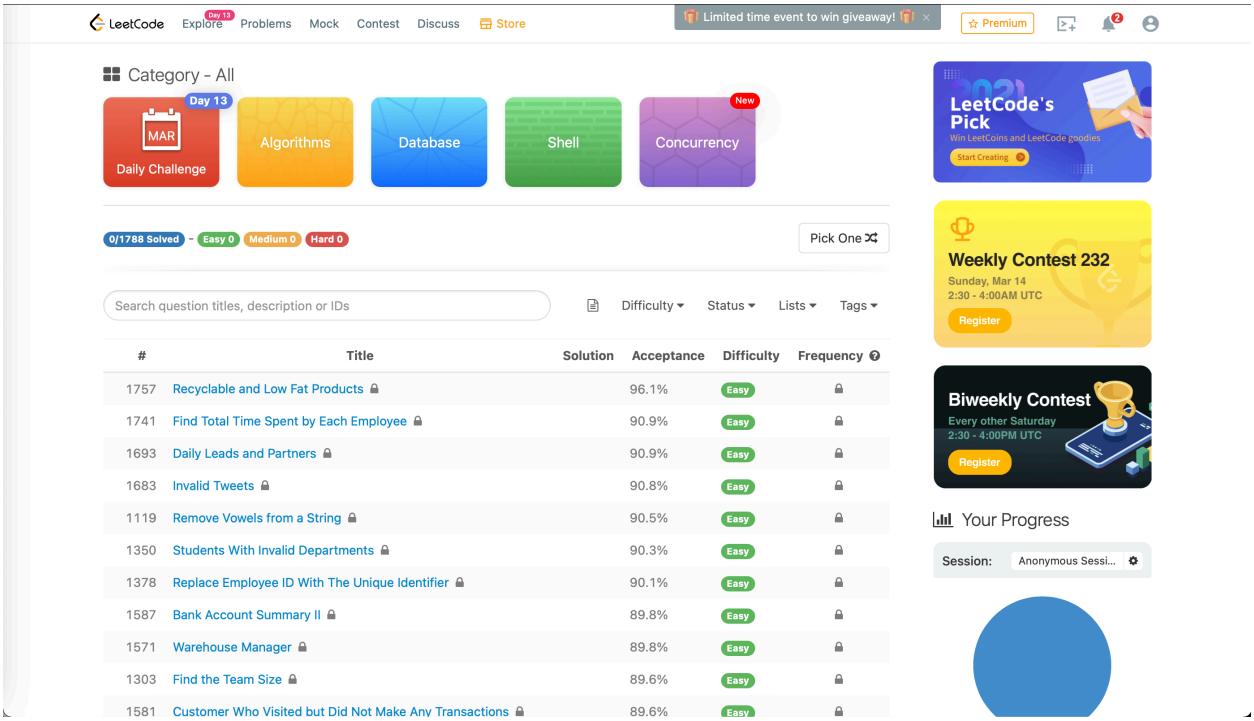
لديك مصفوفة nums. نعرف المجموع التراكمي للمصفوفة على أنه runningSum[i] = sum(nums[0]…nums[i]).

قم بإرجاع المجموع التراكمي لـ nums.

```
class Solution:
    def runningSum(self, nums: [int]) -> [int]:
      running = []
      s = 0
      for num in nums:
        s += num
```


计蒜客
首页
选课中心
学习中心
题库
比赛
客户端下载
李智维
计蒜客3月月赛火热报名中 点击报名
全部题目
默认排序
难度排序
常见问题
输入标题、题号
T1001 计算A+B（新手教程）
通过率：55.0% 正确提交：14493 总提交：26372
入门
T1002 输出马里奥
通过率：28.8% 正确提交：5633 总提交：19569
入门
T1003 输出字符菱形
通过率：36.4% 正确提交：5526 总提交：15170
入门
T1004 输出Hello, World!
通过率：36.2% 正确提交：8277 总提交：22889
入门
T1005 输出字符三角形
通过率：52.0% 正确提交：4768 总提交：9163
入门
T1006 对齐输出
入门
按难度筛选
入门
普及T1
普及T2
普及T3
普及T4/提高T1
提高T2
提高T3
提高T4/省选
NOI/CTS/IOI
按知识点筛选
程序设计入门
基础算法
搜索算法
动态规划
动态规划优化
基础数据结构
高级数据结构
字符串
图论
树上算法
网络流和匹配
数论
组合数学
高等数学
计算几何
其他算法
模板题
☆语法题单
☆算法入门题单
☆算法进阶一阶题单
☆算法进阶题单
☆算法提高题单

□□□□□□ 2: □□□


LeetCode
Explore
Problems
Mock
Contest
Discuss
Store
Limited time event to win giveaway!
Premium
Category - All
Day 13
MAR
Daily Challenge
Algorithms
Database
Shell
New
Concurrency
0/1788 Solved
Easy 0
Medium 0
Hard 0
Pick One
Search question titles, description or IDs
Difficulty
Status
Lists
Tags
Title Solution Acceptance Difficulty Frequency
1757 Recyclable and Low Fat Products 96.1% Easy
1741 Find Total Time Spent by Each Employee 90.9% Easy
1693 Daily Leads and Partners 90.9% Easy
1683 Invalid Tweets 90.8% Easy
1119 Remove Vowels from a String 90.5% Easy
1350 Students With Invalid Departments 90.3% Easy
1378 Replace Employee ID With The Unique Identifier 90.1% Easy
1587 Bank Account Summary II 89.8% Easy
1571 Warehouse Manager 89.8% Easy
1303 Find the Team Size 89.6% Easy
1581 Customer Who Visited but Did Not Make Any Transactions 89.6% Easy
LeetCode's Pick
Win LeetCoins and LeetCode goodies
Start Creating
Weekly Contest 232
Sunday, Mar 14
2:30 - 4:00AM UTC
Register
Biweekly Contest
Every other Saturday
2:30 - 4:00PM UTC
Register
Your Progress
Session:
Anonymous Sessi...

□□□□□□ 3: □□□□□□□□

```
            running.append(s)

        return running
```

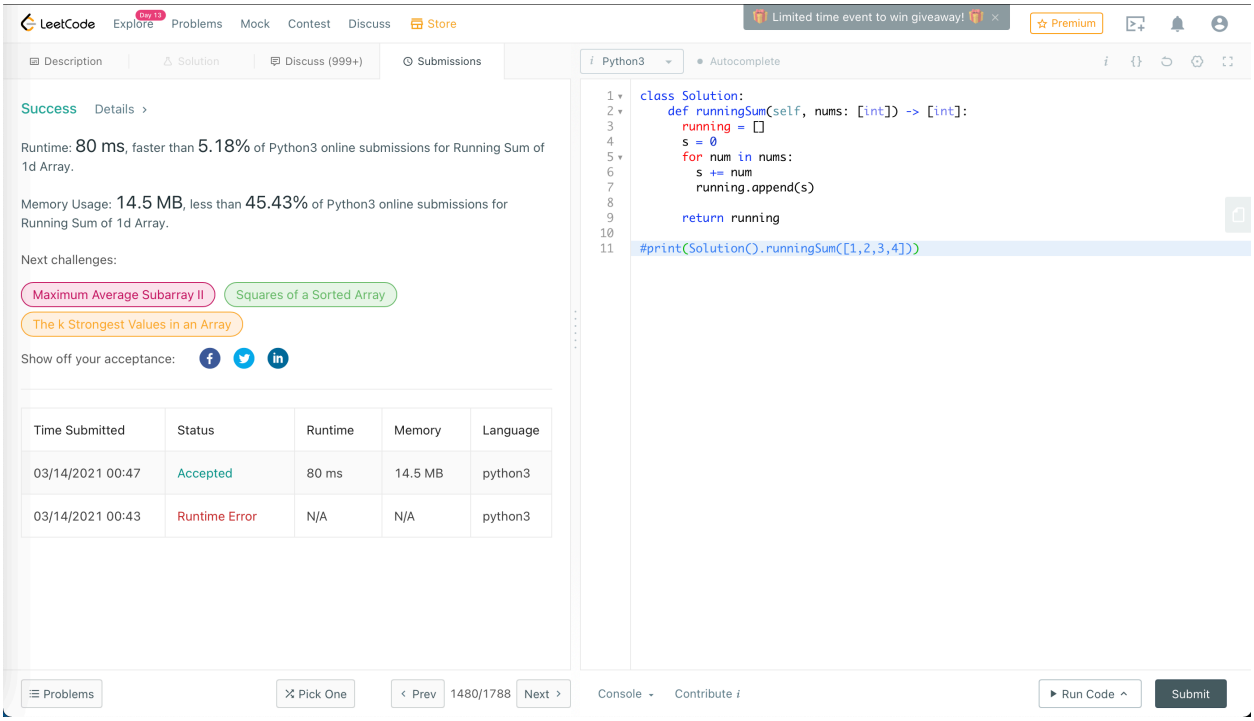
تمت ترجمة الكود والوظيفة. البنية نفس الخوارزمية مع العربية اللغة إلى أعلاه الكود يقوم بحساب المجاميع. هذه على تحتوي جديدة قائمة وإرجاع الأرقام من للقائمة التراكمي المجموع

```
#print(Solution().runningSum([1,2,3,4]))
```

تم تعليق إزالة أردت إذا البرنامج. تشغيل عند تنفيذه يتم لن أنه يعني مما #، باستخدام أعلاه الكود تعليق تم كالتالي: الكود ليصبح # إزالة يمكنك للتنفيذ، قابلًا الكود لجعل

```
print(Solution().runningSum([1,2,3,4]))
```

هذا الكود يستدعي دالة runningSum من الكلاس Solution ويمرر لها القائمة [1,2,3,4]. الدالة تُوسيطة runningSum من المفترض أن تقوم بحساب المجموع التراكمي للعناصر في القائمة.



[illegible] 4: [illegible]

تم اجتياز السؤال الأول. بنجاح

1108. عنوان من النقاط إزالة

الوصف

بـ "." نقطة لكل استبدال يتم حيث عنوان من معدلة نسخة بإرجاع قم ،4 صالح عنوان إلى بالنظر "[.]".

1 مثال:

```
:     address = "1.1.1.1"
:     "1[.]1[.]1[.]1"
```

2 مثال:

```
:     address = "255.100.50.0"
:     "0[.]50[.]100[.]255"
```

القيود:

- address عنوان هو 4 صالح.

الحل

باستخدام ذلك تحقيق يمكن ."[.]" بـ عنوان في نقطة كل استبدال بساطة يمكننا المشكلة، هذه لحل البرمجة. لغات معظم في المتوفرة الاستبدال دالة

كود

```
def defangIPaddr(address: str) -> str:
    return address.replace('.', '[.]')
```

الكود شرح

1. الدالة نستخدم replace نقطة كل لاستبدال بـ "." "[.]".
2. دي عن ة جي تي نت ال. المعدلة

الوقت تعقيد

- الوقت: ، حيث طول هو عنوان. واحدة. مرة العنوان في حرف كل على المرور إلى نحتاج
- المساحة: ، حيث شئنا سلسلة جديدة تحتوي على العنوان. المعدل

خاتمة

يمكن المطلوبة. النتيجة لتحقيق اللغة في الأساسية الدوال استخدام كيفية وتوضح بسيطة المشكلة هذه المماثلة. الدوال باستخدام أخرى برمجة لغات في الفكرة نفس تطبيق

> address، قم بإرجاع نسخة مُعدّلة من هذا العنوان. العنوان صالح IPv4 عنوان إلى بالنظر
> "[.]" بـ "." نقطة كل يستبدل المُعدّل العنوان.

مثال:

```
def defangIPaddr(address: str) -> str:
    return address.replace('.', '[.]')
```

الدالة: استخدام

```
address = "192.168.1.1"
print(defangIPaddr(address))  # :     "1[.]1[.]168[.]192"
```

في هذا المثال، يتم استبدال كل نقطة في العنوان 192.168.1.1 بـ [.]، مما يعطي النتيجة 192[.]168[.]1[.]1.

```
class Solution:
    def defangIPaddr(self, address: str) -> str:
        return address.replace('.', '[.]')
```

العربية: إلى الكود ترجمة

```
class Solution:
    def defangIPaddr(self, address: str) -> str:
        return address.replace('.', '[.]')
```

الكود: شرح

- class Solution: يتم تعريف فئة باسم Solution.
- def defangIPaddr(self, address: str) -> str: يتم تعريف دالة داخل الفئة باسم defangIPaddr التي تأخذ سلسلة من نوع str كمدخل من نوع str وتُرجع نصية من نوع str.
- return address.replace('.', '[.]'): يتم استبدال كل نقطة . في العنوان بـ [.] باستخدام الدالة replace.

مثال:

إذا كان العنوان المدخل هو "192.168.1.1"، فإن المخرجات ستكون "192[.]168[.]1[.]1".

□□□□□□□□□□□□□□□□□.□□□□□□□□□□□□□□□□1.1.1.1□□□

1431.

`n` . `candies` `candies[i]` `i`.

`result` `n` `result[i]` `true` `i`

.

** **:1

□□□□□□: □□□□□□□□ □ □2,3,5,1,3□, □□□□□□□□□□□□□□□ □ 3 □□□□□□□: □□□□□□,□□□□□,□□□□□,□□□□□□,□□□□□□ □□□□ □□□□□□□□□□: □ :□□□□□□□□□ أكبر وهو □ حلوى 5 لديه سيكون ،3 الإضافية الحلوى جميع على حصل وإذا حلوى 2 لديه 1 الطفل
سيكون ،3 الإضافية الحلوى جميع على حصل وإذا حلوى 3 لديه 2 الطفل □ الأطفال. جميع بين الحلوى من عدد
الحلوى جميع على حصل وإذا حلوى 5 لديه 3 الطفل □ الأطفال. جميع بين الحلوى من عدد أكبر وهو □ حلوى 6 لديه
وإذا حلوى 1 لديه 4 الطفل □ الأطفال. جميع بين الحلوى من عدد أكبر وهو □ حلوى 8 لديه سيكون ،3 الإضافية
الأطفال. جميع بين الحلوى من عدد أكبر ليس وهو □ حلوى 4 لديه سيكون ،3 الإضافية الحلوى جميع على حصل
الحلوى من عدد أكبر وهو □ حلوى 6 لديه سيكون ،3 الإضافية الحلوى جميع على حصل وإذا حلوى 3 لديه 5 الطفل □
الأطفال. جميع بين

** **:2

□□□□□□: □□□□□□□□ □ □4,2,1,1,2□, □□□□□□□□□□□□□□□ □ 1 □□□□□□□: □□□□□□,□□□□□□,□□□□□□,□□□□□□,□□□□□□□ □□□□□□□□□□□□: الحلوى. من عدد أكبر لديه فقط واحد طفل هناك

** **:3

□□□□□□: □□□□□□□□ □ □12,1,12□, □□□□□□□□□□□□□□□ □ 10 □□□□□□□: □□□□□□,□□□□□□,□□□□□□

**: **

- `n == candies.length`
- `2 <= n <= 100`
- `1 <= candies[i] <= 100`
- `1 <= extraCandies <= 50`

###

```python
def kidsWithCandies(candies, extraCandies):
    max_candies = max(candies)
    result = []
    for candy in candies:
        if candy + extraCandies >= max_candies:
            result.append(True)
        else:
            result.append(False)
    return result
```

الحل: شرح

1. **إيجاد الحد الأقصى للحلوى:** للحلوى لعدد الأقصى الحد نجد أولاً، الممصفوفة في طفل أي يمتلكها التي الحلوى candies.
2. **تقييم كل طفل:** لكل طفل، لكل إذا ما نتحقق عدد كان الحلوى إليه مضافًا الديه الحلوى الإضافية extraCandies من أكبر سيكون الحد الأقصى سيساوي أو للحلوى.
3. **تحديد النتيجة:** إذا كان الشرط صحيحًا، إلى true نضيف وإلا النتيجة، false.
4. **إرجاع النتيجة:** result القائمة نعيد أخيرًا، التي تحتوي على النتائج لكل طفل.

هذا بسيط العمل، وفعال حيث O(n) وقت في يعمل حيث n هو عدد الأطفال.

> صفوفة لدينا candies وعدد صحيح extraCandies، حيث candies[i] عدد الحلوى التي يمتلكها الطفل □□□□□.
>
> تحقق إذا ما كان هناك طريقة لتوزيع extraCandies بين الأطفال بحيث يمكنه الحصول على **أكبر** عدد من الحلوى. لاحظ أنه يمكن أن يكون لدى عدة أطفال **أكبر** عدد من الحلوى بينهم.

```python
class Solution:
    def kidsWithCandies(self, candies: [int], extraCandies: int) -> [bool]:
        max = 0
        for candy in candies:
          if candy > max:
            max = candy
        greatests = []
        for candy in candies:
```

```
            if candy + extraCandies >= max:
              greatests.append(True)
            else:
              greatests.append(False)
        return greatests
```

[illegible].[illegible]2,3,5,1,3[], 3[]

1672.

<   `accounts`   `m x n`   `accounts[i][j]`   `i`   `j`.
<
> **   **   .   .** **

```python
class Solution:
    def maximumWealth(self, accounts: [[int]]) -> int:
        max = 0
        for account in accounts:
          s = sum(account)
          if max < s:
            max = s
        return max
```

العربية: إلى الكود ترجمة

```
class Solution:
    def maximumWealth(self, accounts: [[int]]) -> int:
        max = 0
        for account in accounts:
          s = sum(account)
          if max < s:
            max = s
        return max
```

الكود: شرح

- الكلاس Solution يحتوي على دالة maximumWealth التي تأخذ معاملًا واحدًا هو accounts، وهو قائمة من القوائم التي تحتوي على أعداد صحيحة [[int]].
- المتغير max يتم تهيئته بقيمة 0، وسيتم استخدامه لتخزين أكبر قيمة للثروة.
- يتم التكرار خلال كل حساب في accounts، ويتم حساب مجموع القيم في كل حساب باستخدام الدالة sum.
- إذا كان المجموع s أكبر من القيمة الحالية لـ max، يتم تحديث max ليكون مساويًا لـ s.
- في النهاية، يتم إرجاع القيمة max، وهي أكبر قيمة للثروة تم العثور عليها.

```
#print(Solution().maximumWealth([[1,2,3],[3,2,1]]))
```

1470. خلط المصفوفة

لدينا المصفوفة nums التي تتكون من 2n عنصر بالشكل [x1,x2,...,xn,y1,y2,...,yn].

قم بإعادة المصفوفة بالشكل [x1,y1,x2,y2,...,xn,yn].

```
class Solution:
  def shuffle(self, nums: [int], n: int) -> [int]:
    ns1 = nums[:n]
    ns2 = nums[n:]
    ns = []
    for i in range(n):
      ns.append(ns1[i])
      ns.append(ns2[i])
    return ns
```

ترجمة الكود إلى العربية:

```
class Solution:
  def shuffle(self, nums: [int], n: int) -> [int]:
    #
    ns1 = nums[:n]   #
    ns2 = nums[n:]   #
    ns = []   #
    for i in range(n):
      #
      ns.append(ns1[i])
      ns.append(ns2[i])
    return ns   #
```

مت س ا و ي ي ن ق س م ي ن إ ل ى ت ق س ي م ه ا ط ر ي ق ع ن nums ا ل ص ح ي ح ة ا ل أ ع د ا د م ن ق ا ئ م ة ب خ ل ط ي ق و م ا ل ك و د **ا ل ك و د: ش ر ح**
ا ل م س ق ل ا ع ن ا ص ر م ن إ ض ا ف ة ط ر ي ق ع ن ns ج د ي د ة ق ا ئ م ة إ ن ش ا ء ي ت م ث م ا ل ق ا ئ م ة. ط و ل ن ص ف ه و n ح ي ث ا ل ط و ل ف ي
ا ل م خ ل و ط ة. ا ل ق ا ئ م ة إ ر ج ا ع ي ت م ا ل ن ه ا ي ة، ف ي ب ا ل ت ن ا و ب. ا ل ث ا ن ي ا ل ق س م ث م ا ل أ و ل

.2,5,1,3,4,7, 3

1512.

**: **

`nums` `(i, j)` `i < j` `nums[i] == nums[j]`.

 ** **:1

ا ل إ د خ ا ل: 1,2,3,1,1,3 ا ل إ خ ر ا ج: 4 ه ن ا ك ا ل ت ف س ي ر: أ ز و ا ج 4 ج ي د ة: 0,3, 0,4, 3,4, 2,5.

 ** **:2

ا ل إ د خ ا ل: 1,1,1,1 ا ل إ خ ر ا ج: 6 ا ل ع ن ا ص ر م ن ز و ج ل ك ا ل ت ف س ي ر: ك ي ش ا ي ج ً ز و ج ا. ي د ً ج ي

 ** **:3

ا ل إ د خ ا ل: 1,2,3 ا ل إ خ ر ا ج: 0 ا ل ت ف س ي ر: ل ا ت و ج د أ ز و ا ج ج ي د ة.

**: **

- `1 <= nums.length <= 100`
- `1 <= nums[i] <= 100`

**: **

. .

```python
def numIdenticalPairs(nums):
    count = 0
```

```
    freq = {}
    for num in nums:
        if num in freq:
            count += freq[num]
            freq[num] += 1
        else:
            freq[num] = 1
    return count
```

الحل: شرح

1. تم تغيير count لتخزين عدد الأزواج الجيدة. تم تهيئة freq كقاموس فارغ لتتبع عدد مرات ظهور كل عنصر في المصفوفة. تم استخدام حلقة for لتكرار كل عنصر في المصفوفة:
2. إذا كان العنصر موجودًا في freq، فإننا نضيف عدد مرات ظهور هذا العنصر إلى count، ثم نزيد عدد مرات ظهوره في freq بمقدار 1.
3. إذا لم يكن العنصر موجودًا في freq، فإننا نضيفه إلى freq مع قيمة 1.
4. في النهاية، تم إرجاع count التي تمثل عدد الأزواج الجيدة.

الوقت: تعقيد

تعقيد الوقت لهذا الحل هو O(n) حيث n هو طول المصفوفة nums. وذلك لأننا نمر بعبر المصفوفة مرة واحدة فقط.

المساحة: تعقيد

تعقيد المساحة هو O(n)، أيضًا قد نحتاج إلى تخزين العناصر جميع في الفريدة في الخريطة في أسوأ الحالات. هذا الحل فعال بشكل ويعمل جيد مع القيود المحددة.

إليك المصفوفة. في عنصر لكل تكرارات عد على تعتمد بسيطة خوارزمية استخدام يمكننا المسألة، هذه لحل الخطوات:

1. **إنشاء قاموس**: استخدم قاموسًا لتخزين عدد مرات ظهور كل عنصر في المصفوفة.
   - تهيئة متغير count لحفظ عدد الأزواج الجيدة.
   - المرور على كل عنصر في المصفوفة.
   - إذا كان العنصر موجودًا في القاموس، فإننا نضيف عدد مرات ظهوره في القاموس إلى count، ثم نزيد عدد مرات ظهوره في القاموس بمقدار 1.
   - إذا لم يكن العنصر موجودًا في القاموس، فإننا نضيفه إلى القاموس مع قيمة 1.
   - في النهاية، تم إرجاع count التي تمثل عدد الأزواج الجيدة.

الوقت: تعقيد

تعقيد الوقت لهذا الحل هو O(n) حيث n هو طول المصفوفة nums. وذلك لأننا نمر بعبر المصفوفة مرة واحدة فقط.

المساحة: تعقيد

تعقيد المساحة هو O(n)، أيضًا قد نحتاج إلى تخزين العناصر جميع في الفريدة في الخريطة في أسوأ الحالات.

هذا الحل فعال بشكل ويعمل جيد مع القيود المحددة.

2. **الجيدة الأزواج حساب**: لكل عنصر في المصفوفة، إذا كان قد ظهر من قبل، فإن عدد الأزواج الجيدة التي تنتهي بهذا العنصر يساوي عدد المرات التي ظهر فيها العنصر من قبل. نحتاج إلى تتبع هذا العدد لكل عنصر.
3. **القاموس تحديث**: بعد حساب عدد الأزواج الجيدة، قم بتحديث القاموس لزيادة عدد مرات ظهور العنصر الحالي.

هذه الخطوات: تنفيذ الكود الذي سيقوم بهذه الخطوات إليك

```
def numIdenticalPairs(nums):
    count = {}
    good_pairs = 0

    for num in nums:
        if num in count:
            good_pairs += count[num]
            count[num] += 1
        else:
            count[num] = 1

    return good_pairs
```

شرح الكود:

- `count` هو القاموس الذي سنستخدمه لتخزين عدد المرات التي ظهر فيها كل عنصر.
- `good_pairs` هو المتغير الذي سنستخدمه لتخزين عدد الأزواج الجيدة.
- نمر على كل عنصر في المصفوفة `nums`:
  - إذا كان العنصر موجودًا بالفعل في القاموس، فإننا نضيف عدد المرات التي ظهر بها العنصر إلى `good_pairs`، ثم نزيد عدد المرات في القاموس بمقدار 1.
  - إذا لم يكن العنصر موجودًا في القاموس، فإننا نضيفه إلى القاموس مع عدد المرات التي ظهر بها إلى 1.
- في النهاية، نعيد قيمة `good_pairs` التي تمثل عدد الأزواج الجيدة.

مثال:

إذا كانت المصفوفة `nums = [1, 2, 3, 1, 1, 3]`، فإن الأزواج الجيدة هي: `(0, 3)` لأن `nums[0] == nums[3] == 1`، `(0, 4)` لأن `nums[0] == nums[4] == 1`، `(3, 4)` لأن `nums[3] == nums[4] == 1`، `(2, 5)` لأن `nums[2] == nums[5] == 3`.

إذن، عدد الأزواج الجيدة هو 4.

الزمني: التعقيد

- الزمني التعقيد هذا لحل هو O(n) حيث n هو عدد العناصر في المصفوفة، لأننا نمر على كل عنصر مرة واحدة فقط.
- المكاني التعقيد هو O(n)، أيضًا، لأننا قد نحتاج إلى تخزين جميع العناصر في القاموس في أسوأ الحالات.

هذا الحل فعال لمعظم المصفوفات حتى جيد بشكل عام. المصفوفات الكبيرة.

```
class Solution:
    def numIdenticalPairs(self, nums: [int]) -> int:
        j = 1
        n = len(nums)
        p = 0
        while j < n:
          for i in range(j):
            if nums[i] == nums[j]:
              p += 1
          j+=1
        return p
```

العربية: إلى الكود ترجمة

```
class Solution:
    def numIdenticalPairs(self, nums: [int]) -> int:
        j = 1
        n = len(nums)
        p = 0
        while j < n:
          for i in range(j):
            if nums[i] == nums[j]:
              p += 1
          j+=1
        return p
```

الكود: شرح

- j = 1: تم تهيئة متغير المؤشر j بالقيمة 1، والذي يستخدم في الحلقة الثانية للعناصر المقارنة.
- n = len(nums): تم تخزين طول قائمة nums في متغير n.

- `p = 0`: يتم تهيئة المتغير p بالقيمة 0، والذي سيستخدم لحساب عدد الأزواج المتطابقة.
- `while j < n:`: يتم تنفيذ الحلقة طالما أن j أقل من طول القائمة n.
  - `for i in range(j):` يتم تنفيذ حلقة داخلية للوصول إلى العناصر التي تسبق العنصر الحالي j.
    - `if nums[i] == nums[j]:` إذا كان العنصر الحالي `nums[j]` يساوي العنصر `nums[i]`، يتم زيادة العداد p بواحد.
  - `j += 1`: يتم زيادة الفهرس j بواحد للانتقال إلى العنصر التالي في القائمة.
- `return p`: يتم إرجاع العدد الإجمالي للأزواج المتطابقة p.

هذا الكود يحسب عدد الأزواج المتطابقة في قائمة الأعداد nums، حيث يعتبر الزوج متطابقًا إذا كان العنصران في الموضعين i و j متساويين و `i < j`.

□□□□□□□□□□□□□□□□□□□.□□□□□□□□□□□□□□□□□□□□□□□1,2,3,1,1,3□□□

771.

**: **

`jewels` `stones` . `stones` .

`"a"` `"A"`.

** **:1

□□□□□: □□□□□□ □ □□□□, □□□□□□ □ □□□□□□□□□ □□□□□□: 3

** **:2

□□□□□: □□□□□□ □ □□□, □□□□□□ □ □□□□ □□□□□□: 0

**: **

- `1 <= jewels.length, stones.length <= 50`
- `jewels` `stones` .
- `jewels` .

**: **

```python
def numJewelsInStones(jewels, stones):
    jewel_set = set(jewels)
    count = 0
    for stone in stones:
        if stone in jewel_set:
            count += 1
    return count
```

الحل: شرح

1. موقن ليوحتب ةلسلس jewels ىلإ مجموعة set ليهستل ةيلمع البحث.
2. موقن ةئيهتب دادع count عبتتل ددع الأحجار الكريمة الموجودة في stones.
3. موقن ريمتب لكل رجح في stones قحقتوال امم إذا نان موجودًا في مجموعة jewels. إذا نان العدد نزيد موجودًا، ب مقدار 1.
4. في stones. الموجودة الكريمة الأحجار عدد يمثل الذي العداد قيمة نعيد النهاية، في

الوقت: تعقيد

- الوقت: [illegible] حيث، [illegible] هو طول jewels و [illegible] هو طول stones.
- المساحة: [illegible]، حيث [illegible] هو طول jewels إنشاء [illegible] لـ. المجموعة.

  لديك سلسلتان من الأحرف: jewels تمثل أنواع الأحجار الكريمة، وstones تمثل الأحجار التي تمتلكها. الأحجار من نوعًا يمثل stones في حرف كل تمتلكها التي الأحجار عدد معرفة تريد أيضًا تعتبر والتي تمتلكها كريمة. أحجارًا

  الحجر من مختلفًا نوعًا "a" الحرف يعتبر لذا، [illegible] الأحرف حساسة الحالة لأحرف "A".

```python
class Solution:
    def numJewelsInStones(self, jewels: str, stones: str) -> int:
        n = 0
        for i in range(len(jewels)):
          js = jewels[i:i+1]
          n += stones.count(js)
        return n
```

العربية: إلى الكود ترجمة

```
class Solution:
    def numJewelsInStones(self, jewels: str, stones: str) -> int:
        n = 0
        for i in range(len(jewels)):
          js = jewels[i:i+1]
          n += stones.count(js)
        return n
```

الكود: شرح

- الكلاس Solution دالة على يحتوي numJewelsInStones تأخذ التي مُدخلين: jewels الجواهر و stones الحجارة.
- نستخدم n المتغير لحساب عدد الجواهر الموجودة في الحجارة.
- نستخدم حلقة for لكل عبر للتكرار في jewels.
- في كل تكرار، يتم أخذ حرف واحد من jewels باستخدام jewels[i:i+1] وتخزينه في المتغير js.
- نستخدم الدالة stones.count(js) لحساب عدد مرات ظهور الحرف js في stones وإضافته إلى n.
- في النهاية، يتم إرجاع القيمة n التي تمثل عدد الجواهر الموجودة في الحجارة.

1603.

**: **

: . .

:

- `ParkingSystem(int big, int medium, int small)`: .
- `bool addCar(int carType)`: . `true`.

** **:

- `1`:

- `2`:
- `3`:

**: **

```python
parkingSystem = ParkingSystem(1, 1, 0)
parkingSystem.addCar(1) #      true
parkingSystem.addCar(2) #      true
parkingSystem.addCar(3) #      false
parkingSystem.addCar(1) #      false
```

الت فسير:

- تم تهيئة نظام الموقف بمواقف كبيرة، واحد متوسط، واحد صغيرة. لا يوجد
- تم إضافة سيارة كبيرة. نجاح
- تم إضافة سيارة متوسطة. نجاح
- تم إضافة سيارة صغيرة. لا توجد
- لا يمكن إضافة سيارة صغيرة أخرى لأن الموقف الكبيرة الوحيد مشغول.

القيود:

- `0 <= big, medium, small <= 1000`
- `1 <= carType <= 3`
- تم استدعاء الدالة `addCar` على الأكثر 1000 مرة.

الحل:

```python
class ParkingSystem:

    def __init__(self, big: int, medium: int, small: int):
        self.spaces = [big, medium, small]

    def addCar(self, carType: int) -> bool:
        if self.spaces[carType - 1] > 0:
            self.spaces[carType - 1] -= 1
            return True
        return False
```

التفسير:

- قائمة في المواقف أنواع من نوع لكل المتاحة الأماكن عدد تخزين يتم spaces.
- المحدد. للنوع متاح مكان وجود من التحقق يتم سيارة، إضافة محاولة عند
- بقيمة النظام ويعود المتاحة الأماكن عدد تقليل يتم متاح، مكان هناك كان إذا true.
- بقيمة النظام يعود متاح، مكان هناك يكن لم إذا false.

كبيرة، الوقوف: أماكن من أنواع ثلاثة على الموقف يحتوي سيارات. لموقف مواقف نظام صمم حجم. لكل الأماكن من ثابت عدد مع وصغيرة، متوسطة،

قم بتنفيذ فئة ParkingSystem:

- فئة من كائن بتهيئة يقوم ParkingSystem(int big, int medium, int small) المُنشئ. من كجزء الوقوف أحجام من حجم لكل الأماكن عدد تحديد يتم ParkingSystem.
- للسيارة carType نوع من متاح وقوف مكان هناك كان إذا مما يتحقق bool addCar(int carType) أو متوسطة، كبيرة، أنواع: ثلاثة من carType يكون أن يمكن الموقف. إلى الدخول تريد التي **في فقط تقف أن للسيارة يمكن** التوالي. على 3 و2، 1 بالأرقام تمثيلها يتم والتي صغيرة، **نوعها نفس من وقوف مكان** carType. إذا إرجاع يتم متاح، مكان هناك يكن لم إذا false، وقوف يتم وإلا إرجاع ويتم المناسب المكان في السيارة true.

```
class ParkingSystem:
    slots = [0, 0, 0]

def __init__(self, big: int, medium: int, small: int):
  self.slots[0] = big
  self.slots[1] = medium
  self.slots[2] = small

def addCar(self, carType: int) -> bool:
  if self.slots[carType - 1] > 0:
    self.slots[carType - 1] -=1
    return True
  else:
    return False
```

يلي: كما العربية إلى أعلاه الكود ترجمة تمت

```
def addCar(self, carType: int) -> bool:
  if self.slots[carType - 1] > 0:
```

```
    self.slots[carType - 1] -=1
    return True
  else:
    return False
```

النظر بغض هي كما تبقى البرمجية الأكواد أن حيث ترجمة، إلى يحتاج ولا □□□□□□ بلغة مكتوب الكود ملاحظة:
الشرح. أو التوثيق في المستخدمة اللغة عن

```
# parkingSystem = ParkingSystem(1, 1, 0)
# print(parkingSystem.addCar(1))
# print(parkingSystem.addCar(2))
# print(parkingSystem.addCar(3))
# print(parkingSystem.addCar(1))
```

1773. عد العناصر المطابقة للقاعدة

لديك مصفوفة `items`، حيث كل عنصر `items[i] = [typei, colori, namei]` يصف النوع، اللون، والاسم للعنصر `i`. كما لديك قاعدة ممثلة بسلسلتين نصيتين، `ruleKey` و `ruleValue`.

يقال أن العنصر `i` يطابق القاعدة إذا كان **واحد** مما يلي صحيحًا:

- `ruleKey == "type"` و `ruleValue == typei`.
- `ruleKey == "color"` و `ruleValue == colori`.
- `ruleKey == "name"` و `ruleValue == namei`.

قم بإرجاع عدد العناصر التي تطابق القاعدة المعطاة.

```
class Solution:
    def countMatches(self, items: [[str]], ruleKey: str, ruleValue: str) -> int:
      i = 0
      if ruleKey == "type":
        i = 0
      elif ruleKey == "color":
        i = 1
      else:
        i = 2
      n = 0
      for item in items:
```

```
        if item[i] == ruleValue:
          n +=1
      return n
```

هي. كما الإنجليزية الأسماء على الحفاظ مع العربية إلى أعلاه الكود ترجمة تمت

[illegible].[illegible],[illegible],[illegible],[illegible]
[illegible], [illegible]

1365.

< `nums` `nums[i]` . `nums[i]` `
>
> .

< ```
> : nums = [8,1,2,2,3]
> : 4],0,1,1,3]
> :
< nums[0]=8 1), 2, 2 .(3
> nums[1]=1 .
< nums[2]=2 .(1)
> nums[3]=2 .(1)
> nums[4]=3 1), 2 .(2
> ```

```python
class Solution:
    def smallerNumbersThanCurrent(self, nums: [int]) -> [int]:
        ns = []
        l = len(nums)
        for i in range(l):
          n = 0
          for j in range(l):
            if i != j:
```

```
            if nums[j] < nums[i]:
              n += 1
          ns.append(n)
        return ns
```

العربية: اللغة إلى الكود ترجمة

```
class Solution:
    def smallerNumbersThanCurrent(self, nums: [int]) -> [int]:
        ns = []  #
        l = len(nums)  #         nums
        for i in range(l):  #
          n = 0  #
          for j in range(l):  #
            if i != j:  #
              if nums[j] < nums[i]:  #
                n += 1  #
          ns.append(n)  #
        return ns  #
```

شرح الكود: لكل في القائمة، nums. القائمة في عنصر لكل عنصر من الأصغر العناصر عدد بحساب يقوم الكود زيادة يتم الحالي، العنصر من أصغر الآخر العنصر كان إذا القائمة. في الأخرى العناصر جميع مع مقارنته يتم n. العداد في النتائج. على تحتوي التي ns القائمة إلى n العدد إضافة يتم النهاية، في ns القائمة إرجاع يتم الأصلية. القائمة في عنصر كل من الأصغر العناصر عدد على تحتوي التي

[illegible].[illegible]8,1,2,2,3[illegible]

528 11.81% . .

```python
class Solution:
    def smallerNumbersThanCurrent(self, nums: [int]) -> [int]:
        l = len(nums)
```

العربية: إلى أعلاه الكود ترجمة

```
class Solution:
    def smallerNumbersThanCurrent(self, nums: [int]) -> [int]:
        l = len(nums)
```

قائمة تأخذ الدالة هذه .Solution الكلاس داخل smallerNumbersThanCurrent دالة تعريف يتم أعلاه، الكود في
الصحيحة الأعداد من []nums وتُرجع أخرى قائمة من الصحيحة. الأعداد المتغير l يتم تهيئة طول لطول القائمة
nums.

```
sort_nums = nums.copy()

ins = list(range(l))
for i in range(l):
    for j in range(i+1, l):
        if sort_nums[i] > sort_nums[j]:
            a = sort_nums[i]
            sort_nums[i] = sort_nums[j]
            sort_nums[j] = a

            a = ins[i]
            ins[i] = ins[j]
            ins[j] = a

smalls = [0]
for i in range(1, l):
    if sort_nums[i-1] == sort_nums[i]:
        smalls.append(smalls[i-1])
    else:
        smalls.append(i)
```

العربية: إلى الكود ترجمة

```
#                    0    l-1
ins = list(range(l))

#            (Bubble Sort)        sort_nums
for i in range(l):
    for j in range(i+1, l):
```

```
        if sort_nums[i] > sort_nums[j]:
            #                 sort_nums
            a = sort_nums[i]
            sort_nums[i] = sort_nums[j]
            sort_nums[j] = a

            #                 ins
            a = ins[i]
            ins[i] = ins[j]
            ins[j] = a

#         smalls
smalls = [0]
for i in range(1, l):
    if sort_nums[i-1] == sort_nums[i]:
        #
        smalls.append(smalls[i-1])
    else:
        #
        smalls.append(i)
```

شرح الكود:

1. **إنشاء قائمة ins**: يتم إنشاء قائمة تحتوي على الأرقام من 0 إلى □□1، حيث l هو طول القائمة sort_nums.
2. **عملية الفرز □□□□□□□ □□□□□**: يتم تنفيذ عملية فرز على القائمة sort_nums باستخدام خوارزمية □□□□□□ □□□□. أثناء عملية الفرز، يتم أيضًا تبديل العناصر في القائمة ins بنفس الترتيب للحفاظ على العلاقة بين العناصر في القائمتين.
3. **إنشاء قائمة smalls**: يتم إنشاء قائمة smalls لتخزين عدد العناصر الأصغر من كل عنصر في القائمة sort_nums بعد الفرز. إذا كان العنصر الحالي مساويًا للعنصر السابق، يتم استخدام نفس القيمة المخزنة في smalls للعنصر السابق. إذا كان العنصر الحالي مختلفًا، يتم استخدام الفهرس الحالي كقيمة جديدة في smalls.

```
# print(sort_nums)
# print(smalls)
```

```
r_is = list(range(l))
for i in ins:
```

```python
    r_is[ins[i]] = i

ns = []
for i in range(l):
    ns.append(smalls[r_is[i]])
return ns
```

للكود، ترجمة أو شرح إلى بحاجة كنت إذا الأصلي. الهيكل على الحفاظ مع □□□□□ لغة إلى الكود تحويل تم
ذلك. توضيح فالرجاء

□□□□□□□□□□□□□□□□□.□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□8,1,2,2,3□□□

`284ms` `528ms`.

.

````
```python
class Solution:
    def smallerNumbersThanCurrent(self, nums: [int]) -> [int]:
        #
        sort_nums = nums.copy()
        sort_nums.sort()

        #
        ns = []
        for num in nums:
          #
          ns.append(sort_nums.index(num))
        return ns
````

□□□□□□□□□□□□□□□□□.□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□8,1,2,2,3□□□

`64ms` 71%`` .

```python

```
class Solution:
    def smallerNumbersThanCurrent(self, nums: [int]) -> [int]:
        l = len(nums)
        ns = [0] * l
        for i in range(l):
          for j in range(i+1, l):
            if nums[i] > nums[j]:
              ns[i] +=1
            elif nums[i] < nums[j]:
              ns[j] +=1
            else:
              pass
        return ns
```

□□□□□□□□□□□□□□□□.□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□8,1,2,2,3□□□

. `400ms`.

```python
class Solution:
    def smallerNumbersThanCurrent(self, nums: [int]) -> [int]:
        ss = sorted((e,i) for i,e in enumerate(nums))

l = len(nums)
smalls = [0]
for i in range(1, l):
    (e0, j0) = ss[i-1]
    (e1, j1) = ss[i]
    if e0 == e1:
        smalls.append(smalls[i-1])
    else:
        smalls.append(i)

ns = [0]*l
for i in range(l):
    (e, j) = ss[i]
```

```
    ns[j] = smalls[i]
return ns
```

إلى: الكود ترجمة تمت

```
ns = [0]*l
for i in range(l):
    (e, j) = ss[i]
    ns[j] = smalls[i]
return ns
```

ترجمته. يتم ولا [illegible] برمجة بلغة مكتوب لأنه هو كما يبقى الكود ملاحظة:

[illegible].[illegible]8,1,2,2,3[illegible]

```
>    :     52             91.45%                  Python3      "                  ."
<
>      :     14.6           15.18%                    Python3      "                  ."
```

```
    !                      91.45%``    .
```

```
      .
```

```python
class Solution:
    def smallerNumbersThanCurrent(self, nums: [int]) -> [int]:
        ss = sorted((e,i) for i,e in enumerate(nums))

l = len(nums)
smalls = [0]
ns = [0]*l
for i in range(1, l):
    (e0, j0) = ss[i-1]
    (e1, j1) = ss[i]
    if e0 == e1:
        smalls.append(smalls[i-1])
```

```
    else:
        smalls.append(i)
```

إلى: الكود ترجمة تمت

```
l = len(nums)
smalls = [0]
ns = [0]*l
for i in range(1, l):
    (e0, j0) = ss[i-1]
    (e1, j1) = ss[i]
    if e0 == e1:
        smalls.append(smalls[i-1])
    else:
        smalls.append(i)
```

تغييرها. يتم ولم بالإنجليزية، ودوال متغيرات أسماء على يحتوي لأنه هو كما بقي الكود ملاحظة:

```
ns[j1] = smalls[i]
return ns
```

[8,1,2,2,3]

.

```python
class Solution:
    def smallerNumbersThanCurrent(self, nums: [int]) -> [int]:
        ss = sorted((e,i) for i,e in enumerate(nums))

l = len(nums)
last = 0
ns = [0]*l
for i in range(1, l):
    (e0, j0) = ss[i-1]
    (e1, j1) = ss[i]
```

```
    if e0 == e1:
        pass
    else:
        last = i

ns[j1] = last
return ns
```

□□□□□□□□□□□□□□□□.□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□8,1,2,2,3□□□

`40ms` 99.81%`` .

< : 40 99.81% Python3 " ."
<
> : 14.4 15.18% Python3 " ."

.

```python
class Solution:
    def smallerNumbersThanCurrent(self, nums: [int]) -> [int]:
      l = len(nums)
      n = [0] * 101
      max_num = 0
      for num in nums:
        n[num] += 1
        if num > max_num:
          max_num = num
```

ذهذا ودكلا يعرض جزءًا من دالة في لغة □□□□□□ تُسمى smallerNumbersThanCurrent والتي تأخذ من قائمة الأعداد nums الصحيحة وتُرجع قائمة جديدة تحتوي على عدد الأعداد الأصغر من كل عنصر في القائمة الأصلية.

في هذا الجزء من الكود: □ تم إنشاء قائمة n تحتوي على 101 صفرًا. □ تم تحديد طول القائمة nums وحفظه في المتغير l. □ تم استخدام حلقة for لتكرار عبر كل عنصر في القائمة nums: □ إذا كان num الحالي أكبر من max_num الحالية، □ تم تحديث المتغير max_num ليصبح num الحالي.

هذا الجزء من الكود يُعد حلًا خوارزميًا جزءًا من مشكلة أكبر تتضمن حساب عدد الأعداد الأصغر من كل عنصر في القائمة.

```
sm = [0] * (max_num + 1)
sum = 0
for i in range(max_num + 1):
    sm[i] = sum
    sum += n[i]

ns = [0] * l
for i in range(l):
    ns[i] = sm[nums[i]]

      ns
```

□□□□□□□□□□□□□□□□□.□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□8,1,2,2,3□□□

.

```
```python
class Solution:
    def smallerNumbersThanCurrent(self, nums: [int]) -> [int]:
      l = len(nums)
      n = [0] * 101
      max_num = 0
      for num in nums:
        n[num] += 1
        if num > max_num:
          max_num = num
```

في الكود أعلاه، يتم تعريف دالة smallerNumbersThanCurrent داخل فئة Solution. هذه الدالة تأخذ قائمة من الأعداد الصحيحة nums وتعيد قائمة أخرى تحتوي على عدد الأعداد الأصغر من كل عنصر في القائمة الأصلية.

شرح الكود:

1. تهيئة المتغيرات:
   - l يتم تعيينه لطول القائمة nums.
   - n يتم تهيئته كقائمة تحتوي على 101 عنصرًا، كلها تساوي 0. هذا سيتم استخدامه لحساب عدد المرات التي يظهر فيها كل رقم في القائمة.
```

- max_num تم تهيئته إلى 0، وسيتم استخدامه لتتبع أكبر رقم في القائمة.

2. **التكرارات: حساب**

- تكرار: لكل وفي nums، تم تكرار القائمة
  - تم زيادة قيمة n[num] بمقدار 1، حيث num هو رصيد العنصر الحالي في القائمة.
  - إذا كان num أكبر من max_num، تم تحديث max_num ليكون num.

تم استكمال الكود اللاحق لحساب عدد الأعداد الأصغر من كل عنصر في القائمة.

```
short_n = []
short_num = [] * l
zn = [0] * 101
j = 0
for i in range(max_num+1):
    if n[i] > 0:
        zn[i] = j
        short_n.append(n[i])
        short_num.append(num)
        j+=1

sm = [0] * j
sum = 0
for i in range(j):
    sm[i] = sum
    sum += short_n[i]

ns = [0] * l
for i in range(l):
    ns[i] = sm[zn[nums[i]]]
return ns
```

[illegible]8,1,2,2,3[illegible]

```python
class Solution:
```

```
    def smallerNumbersThanCurrent(self, nums: [int]) -> [int]:
      max_num = max(nums)
```

العربية: إلى أعلاه الكود ترجمة

```
class Solution:

    def smallerNumbersThanCurrent(self, nums: [int]) -> [int]:
      max_num = max(nums)  #
```

في هذا الكود، يتم تعريف دالة smallerNumbersThanCurrent داخل الكلاس Solution. هذه الدالة تأخذ قائمة nums كمدخل وتقوم بإرجاع قائمة جديدة تحتوي على عدد الأعداد الأصغر من كل رقم في القائمة الأصلية. الدالة تستخدام nums القائمة في القيمة القصوى على العثور يتم الدالة، داخل الأول السطر في max() الدالة.

```
n = [0] * (max_num + 1)
for num in nums:
    n[num] += 1
```

ثم تمثل الكود هذا .1 بمقدار n القائمة في المقابل العناصر جميع تهيئة يتم حيث، max_num + 1 بحجم n قائمة إنشاء عملية أعلاه الكود تمثل .0 بالقيمة العناصر جميع تكرار يتم ثم، nums قائمة عبر تكرار يتم زيادة قيمة العنصر، وفي كل تكرار، يتم استخدام عادةً حساب لحساب تكرار القيم في الأرقام قائمة nums.

```
sorted_ls = []
for i in range(max_num + 1):
    if n[i] > 0:
        sorted_ls.append(i)
```

إلى: الكود ترجمة تمت

```
sorted_ls = []
for i in range(max_num + 1):
    if n[i] > 0:
        sorted_ls.append(i)
```

ترجمته. يتم ولا [illegible] برمجة بلغة مكتوب لأنه هو كما يبقى الكود ملاحظة:

```
        sm = [0] * (max_num + 1)
        sum = 0
        for i in range(len(sorted_ls)):
          v = sorted_ls[i]
          sm[v] = sum
          sum += n[v]

        ns = []
        for i in range(len(nums)):
          ns.append(sm[nums[i]])
        return ns
# print(Solution().smallerNumbersThanCurrent([72,48,32,16,10,59,83,38,1,4,68,7,67,16,5,35,99,15,55,11,2
```

التمرين

 أعلاه. المذكورة لتلك المشابهة الأسئلة بعض بحل الطالب يقوم