□□□□□□ प्रोग्रामिंग में ऑनलाइन प्रश्न हल करना

यहां हम ऑनलाइन जज सिस्टम का उपयोग करके प्रश्न हल करेंगे। अगर आपकी अंग्रेजी अच्छी है, तो आप Codeforces और LeetCode का उपयोग कर सकते हैं। चीनी भाषा में आप □□□ और □□ का उपयोग कर सकते हैं। यहां हम LeetCode का उपयोग करेंगे। मैंने यहां 10 प्रश्न हल किए हैं। साथ ही, अंतिम प्रश्न को कई तरीकों से हल किया गया है, जिससे प्रोग्राम की दक्षता को 10% सबमिशन से बेहतर करके 99% तक पहुंचाया गया है।

□□□□□□ 1: □□

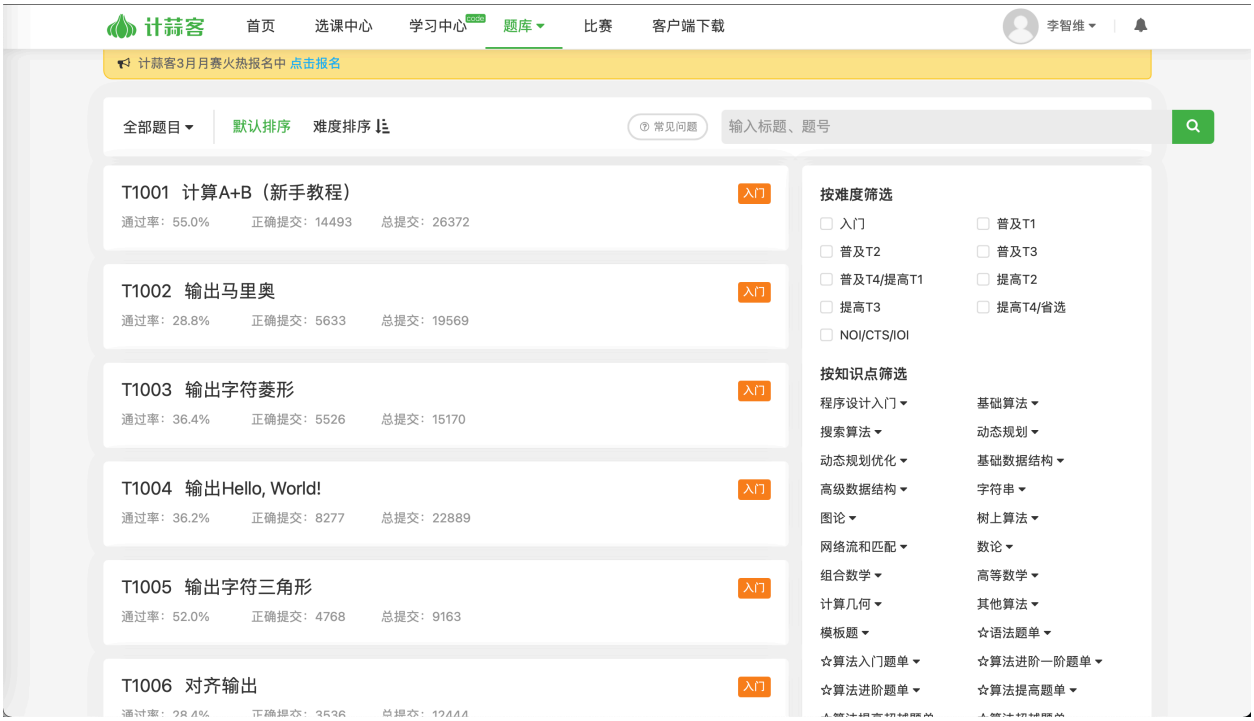
1480. 1डी ऐरे का रनिंग योग

समस्या का विवरण

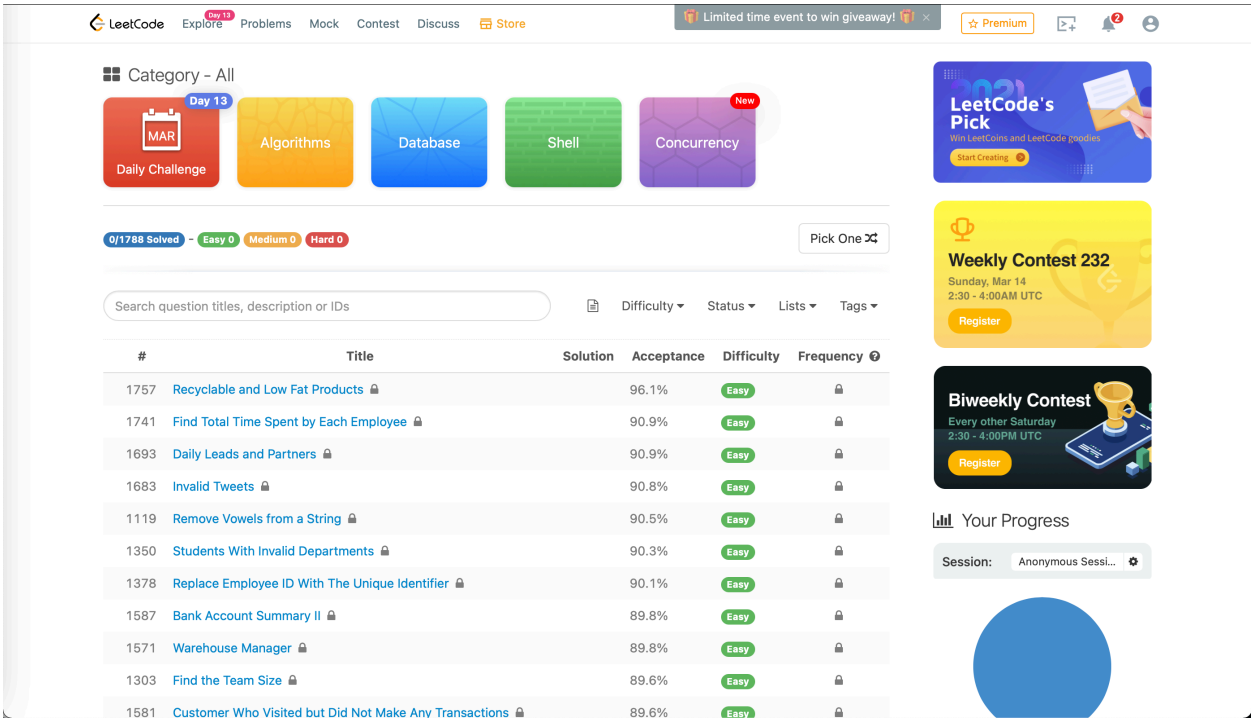
एक सरणी nums दी गई है। हमें एक नई सरणी runningSum लौटानी है जहां runningSum[i] = sum(nums[0]…nums[i]) हो।

उदाहरण 1:

```
: nums = [1,2,3,4]
 : [1,3,6,10]
 :                    : [1, 1+2, 1+2+3, 1+2+3+4]
```


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
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: □□□□□□□□□

उदाहरण 2:

```
: nums = [1,1,1,1,1]
 : [1,2,3,4,5]
  :                    : [1, 1+1, 1+1+1, 1+1+1+1, 1+1+1+1+1]
```

उदाहरण 3:

```
: nums = [3,1,2,10,1]
 : [3,4,6,16,17]
```

प्रतिबंध: - `1 <= nums.length <= 1000` - `-10^6 <= nums[i] <= 10^6`

समाधान

हम इस समस्या को एक साधारण लूप का उपयोग करके हल कर सकते हैं। हम एक नई सरणी `runningSum` बनाएंगे और प्रत्येक इंडेक्स पर पिछले सभी तत्वों का योग जोड़ेंगे।

```python
class Solution:
    def runningSum(self, nums: List[int]) -> List[int]:
        runningSum = []
        current_sum = 0
        for num in nums:
            current_sum += num
            runningSum.append(current_sum)
        return runningSum
```

व्याख्या

1. हम एक खाली सूची `runningSum` बनाते हैं और एक वेरिएबल `current_sum` को 0 पर सेट करते हैं।
2. हम `nums` सरणी के प्रत्येक तत्व को लूप करते हैं।
3. प्रत्येक तत्व को `current_sum` में जोड़ते हैं और इसे `runningSum` सूची में जोड़ते हैं।
4. अंत में, `runningSum` सूची को लौटाते हैं।

यह समाधान □(□) समय जटिलता में चलता है, जहां □ सरणी की लंबाई है।

एक सरणी `nums` दी गई है। हम एक सरणी का रनिंग योग (□□□□□□□□ □□□) इस प्रकार परिभाषित करते हैं: `runningSum[i] = sum(nums[0]…nums[i])`।

`nums` का रनिंग योग लौटाएं।

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

      return running
```

इस कोड में, `Solution` नामक एक क्लास है जिसमें `runningSum` नामक एक मेथड है। यह मेथड एक इनपुट लिस्ट `nums` लेता है और एक नई लिस्ट `running` वापस करता है। `running` लिस्ट में `nums` लिस्ट के तत्वों का क्रमशः योग (□□□□□□□□ □□□) होता है।

1. `running` एक खाली लिस्ट है जिसमें हम रनिंग योग को स्टोर करेंगे।
2. `s` एक वेरिएबल है जो योग को ट्रैक करता है, शुरुआत में यह 0 है।
3. `for` लूप `nums` लिस्ट के हर तत्व को लेता है और उसे `s` में जोड़ता है।
4. हर बार जब `s` अपडेट होता है, तो उसे `running` लिस्ट में जोड़ दिया जाता है।
5. अंत में, `running` लिस्ट को वापस कर दिया जाता है।

उदाहरण के लिए, यदि `nums = [1, 2, 3, 4]`, तो `runningSum` मेथड `[1, 3, 6, 10]` वापस करेगा।

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

पहला प्रश्न पास हो गया।

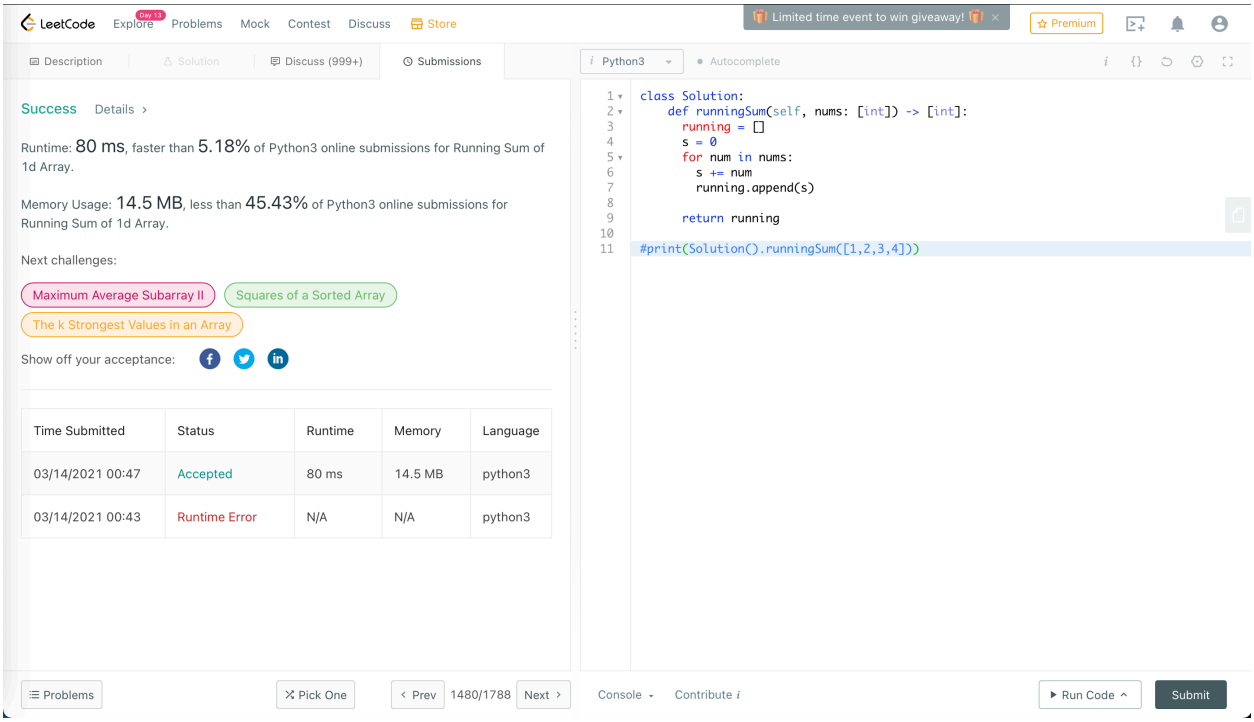
1108. □□ एड्रेस को डिफैंग करना

एक □□ एड्रेस को डिफैंग करने का मतलब है कि एड्रेस में मौजूद सभी डॉट्स (`.`) को `[.]` से बदल देना। यह सुरक्षा कारणों से किया जाता है ताकि □□ एड्रेस को टेक्स्ट के रूप में सुरक्षित रूप से प्रदर्शित किया जा सके।

उदाहरण के लिए, यदि □□ एड्रेस `192.168.1.1` है, तो इसे डिफैंग करने के बाद यह `192[.]168[.]1[.]1` हो जाएगा।

समस्या का विवरण

आपको एक वैध □□□4 एड्रेस दिया गया है, और आपको इसे डिफैंग करना है।



□□□□□□□ 4: □□

उदाहरण

उदाहरण 1:

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

उदाहरण 2:

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

समाधान

इस समस्या को हल करने के लिए, हम सरलता से □□ एड्रेस में मौजूद सभी डॉट्स को `[.]` से बदल सकते हैं। यह काम □□□□□□□ में `replace()` मेथड का उपयोग करके आसानी से किया जा सकता है।

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

समय और स्थान जटिलता

- **समय जटिलता:** □(□), जहां □ इनपुट स्ट्रिंग की लंबाई है।
- **स्थान जटिलता:** □(□), क्योंकि हम एक नई स्ट्रिंग बना रहे हैं।

परीक्षण

आइए उपरोक्त समाधान को दिए गए उदाहरणों पर परीक्षण करें:

```
print(defangIPaddr("1.1.1.1"))       # Output: "1[.]1[.]1[.]1"
print(defangIPaddr("255.100.50.0")) # Output: "255[.]100[.]50[.]0"
```

यह समाधान सरल और प्रभावी है, और यह समस्या को □(□) समय में हल करता है।

एक वैध (□□□4) □□ `address` दिया गया है, उस □□ पते का एक डिफेंज्ड (□□□□□□□□□) संस्करण लौटाएं।

एक डिफेंज्ड □□ पता हर पीरियड `"."` को `"[.]"` से बदल देता है।

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

यह कोड एक □□ एड्रेस को "डिफैंग" करता है, जिसका अर्थ है कि इसमें मौजूद सभी डॉट्स (`.`) को `[.]` से बदल दिया जाता है। उदाहरण के लिए, `"192.168.1.1"` को `"192[.]168[.]1[.]1"` में बदल दिया जाएगा।

□□□□□(□□□□□□□□□().□□□□□□□□□□□□□□('1.1.1.1'))

```
## 1431.

**   :**

   `n`                `candies`      ,    `candies[i]` `i`-                                    `extraCa

              `result`      ,    `result[i]` `true`       `i`-      `extraCandies`

**   :**

**    1:**
```

□□□□□□: □□□□□□□□ = [2,3,5,1,3], □□□□□□□□□□□□□□ = 3 □□□□□□□: [□□□□□,□□□□□,□□□□□,□□□□□□,□□□□□]

**   :**

- 3 , 5 ,
- 3 , 6 ,
- 3 , 8 ,
- 3 , 4 ,
- 3 , 6 ,

**   2:**

□□□□□□: □□□□□□□□ = [4,2,1,1,2], □□□□□□□□□□□□□□ = 1 □□□□□□□: [□□□□□,□□□□□□,□□□□□□,□□□□□□,□□□□□□]

**   :**

-

**   :**

```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
```

समय जटिलता: □(□), जहां `n` बच्चों की संख्या है।

स्पेस जटिलता: □(□), क्योंकि हम एक नई सरणी `result` बना रहे हैं।

> दिया गया है सरणी `candies` और पूर्णांक `extraCandies`, जहां `candies[i]` ***वें*** बच्चे के पास मौजूद कैंडी की संख्या को दर्शाता है।
>
> प्रत्येक बच्चे के लिए जांचें कि क्या `extraCandies` को बच्चों के बीच इस तरह वितरित किया जा सकता है कि वह उनमें से **सबसे अधिक** कैंडी रख सके। ध्यान दें कि एक से अधिक बच्चे **सबसे अधिक** कैंडी रख सकते हैं।

```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
```

इस कोड में, `Solution` नामक एक क्लास है जिसमें `kidsWithCandies` नामक एक मेथड है। यह मेथड दो इनपुट लेता है: `candies` (एक लिस्ट जिसमें प्रत्येक बच्चे के पास कैंडी की संख्या होती है) और `extraCandies` (एक पूर्णांक जो अतिरिक्त कैंडी की संख्या को दर्शाता है)।

1. पहले, यह `candies` लिस्ट में से सबसे ज्यादा कैंडी की संख्या (`max`) को ढूंढता है।
2. फिर, यह `candies` लिस्ट के प्रत्येक बच्चे के लिए जांचता है कि क्या उनके पास `extraCandies` जोड़ने के बाद भी `max` से ज्यादा या बराबर कैंडी होगी।
3. यदि हां, तो `True` को `greatests` लिस्ट में जोड़ा जाता है, अन्यथा `False` जोड़ा जाता है।
4. अंत में, `greatests` लिस्ट को रिटर्न किया जाता है, जो यह दर्शाता है कि कौन से बच्चे `extraCandies` जोड़ने के बाद सबसे ज्यादा कैंडी वाले बच्चे बन सकते हैं।

□□□□□(□□□□□□□□□().□□□□□□□□□□□□□□□□□([2,3,5,1,3], 3))

```
## 1672.

>      `m x n`        `accounts`           `accounts[i][j]` `ith`         `jth`          *           **
>
>         **   **                                      **   **

```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
```

इस कोड को हिंदी में समझाएं:

यह एक □□□□□□ क्लास `Solution` है जिसमें एक मेथड `maximumWealth` है। यह मेथड `accounts` नामक एक 2□ लिस्ट (जो कि इंटीजर की लिस्ट्स की लिस्ट है) को इनपुट के रूप में लेता है और एक इंटीजर वैल्यू रिटर्न करता है।

1. `max` नामक एक वेरिएबल को 0 से इनिशियलाइज़ किया गया है। यह वेरिएबल अधिकतम धन (□□□□□□) को स्टोर करेगा।
2. `accounts` लिस्ट में प्रत्येक `account` (जो कि एक लिस्ट है) के लिए:
   - `s` नामक वेरिएबल में `account` लिस्ट के सभी एलिमेंट्स का योग (□□□) स्टोर किया जाता है।
   - यदि `max` की वैल्यू `s` से कम है, तो `max` को `s` के बराबर सेट कर दिया जाता है।
3. अंत में, `max` की वैल्यू रिटर्न की जाती है, जो कि सभी खातों (□□□□□□□□□) में से अधिकतम धन (□□□□□□□) को दर्शाती है।

इस प्रकार, यह फंक्शन दिए गए खातों (□□□□□□□□□) में से सबसे अधिक धन (□□□□□□□) वाले खाते की राशि को ढूंढता है और उसे रिटर्न करता है।

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

इस कोड को हिंदी में समझाया जाए तो:

यह कोड एक `Solution` क्लास के `maximumWealth` मेथड को कॉल कर रहा है और इसे एक 2□ लिस्ट `[[1,2,3],[3,2,1]]` पास कर रहा है। यह लिस्ट दो ग्राहकों की संपत्ति को दर्शाती है, जहां प्रत्येक ग्राहक की संपत्ति एक लिस्ट में है। उदाहरण के लिए, पहले ग्राहक की संपत्ति `[1,2,3]` है और दूसरे ग्राहक की संपत्ति `[3,2,1]` है।

`maximumWealth` मेथड का उद्देश्य इन ग्राहकों की कुल संपत्ति की गणना करना और सबसे अधिक संपत्ति वाले ग्राहक की कुल संपत्ति को वापस करना है।

इस कोड को रन करने पर, यह `maximumWealth` मेथड के आउटपुट को प्रिंट करेगा, जो कि सबसे अधिक संपत्ति वाले ग्राहक की कुल संपत्ति होगी।

1470. ऐरे को शफल करें

दिया गया है `2n` तत्वों वाला सरणी `nums`, जो `[x1,x2,...,xn,y1,y2,...,yn]` के रूप में है।

सरणी को इस रूप में लौटाएं `[x1,y1,x2,y2,...,xn,yn]`।

```python
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
```

इस कोड में, `Solution` क्लास में `shuffle` नामक एक मेथड है जो दो पैरामीटर लेती है: `nums` (एक इंटीजर लिस्ट) और `n` (एक इंटीजर)। यह मेथड `nums` लिस्ट को दो हिस्सों में बांटती है: `ns1` और `ns2`। फिर यह इन दोनों हिस्सों के एलिमेंट्स को एक नई लिस्ट `ns` में बारी-बारी से जोड़ती है और अंत में इस नई लिस्ट को रिटर्न करती है।

□□□□□(□□□□□□□□□().□□□□□□□□([2,5,1,3,4,7], 3))

```
## 1512.

>             `nums`
>
>       `(i,j)`   *  *           `nums[i]` == `nums[j]`   `i` < `j`
>
> * *
```

```python
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
```

यह कोड एक समाधान प्रदान करता है जो एक सूची में समान संख्याओं के जोड़े (□□□□□) की संख्या गिनता है। यहां `nums` एक पूर्णांकों की सूची है, और `numIdenticalPairs` फ़ंक्शन इस सूची में समान संख्याओं के जोड़े की संख्या लौटाता है।

कोड का विवरण:

1. `j` को 1 से शुरू किया जाता है और `n` को सूची `nums` की लंबाई के रूप में सेट किया जाता है।
2. `p` को 0 से शुरू किया जाता है, जो समान जोड़े की संख्या को संग्रहीत करेगा।
3. एक `while` लूप का उपयोग करके, `j` को `n` से कम होने तक चलाया जाता है।
4. एक `for` लूप का उपयोग करके, `i` को 0 से `j-1` तक चलाया जाता है।
5. यदि `nums[i]` और `nums[j]` समान हैं, तो `p` को 1 से बढ़ाया जाता है।
6. अंत में, `p` को लौटाया जाता है, जो समान जोड़े की कुल संख्या को दर्शाता है।

यह कोड समान संख्याओं के सभी संभावित जोड़े की संख्या की गणना करता है।

□□□□□(□□□□□□□□□().□□□□□□□□□□□□□□□□□□□□([1,2,3,1,1,3]))

```
## 771.

>               : `jewels`      ( )             ,   `stones`                        `stones`
>
>           ,     `"a"`   `"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
```

इस कोड में, `Solution` नामक एक क्लास है जिसमें `numJewelsInStones` नामक एक मेथड है। यह मेथड दो स्ट्रिंग्स `jewels` और `stones` को इनपुट के रूप में लेता है और एक इंटीजर वैल्यू रिटर्न करता है।

इस मेथड का उद्देश्य यह जांचना है कि `stones` स्ट्रिंग में `jewels` स्ट्रिंग के कितने कैरेक्टर्स मौजूद हैं।

1. `n` नामक एक वेरिएबल को 0 से इनिशियलाइज़ किया जाता है।

2. एक लूप `jewels` स्ट्रिंग के हर कैरेक्टर के लिए चलाया जाता है।
3. `js` वेरिएबल में `jewels` स्ट्रिंग का वर्तमान कैरेक्टर स्टोर किया जाता है।
4. `stones` स्ट्रिंग में `js` कैरेक्टर की संख्या गिनी जाती है और `n` में जोड़ दी जाती है।
5. अंत में, `n` को रिटर्न किया जाता है, जो `stones` में `jewels` के कैरेक्टर्स की कुल संख्या को दर्शाता है।

□□□□□(□□□□□□□□□().□□□□□□□□□□□□□□□□□□□□(“□□”, “□□□□□□□□”))

```
## 1603.

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

```python
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
```

(यह कोड ब्लॉक है, इसे अपरिवर्तित छोड़ दिया गया है।)
```

□□□□□□□□□□□□□□□ = □□□□□□□□□□□□□□□□(1, 1, 0)

□□□□□(□□□□□□□□□□□□□□□.□□□□□□□(1))

□□□□□(□□□□□□□□□□□□□□□.□□□□□□□(2))

□□□□□(□□□□□□□□□□□□□□□.□□□□□□□(3))

□□□□□(□□□□□□□□□□□□□□□.□□□□□□□(1))

1773.

> `items` , `items[i] = [typei, colori, namei]` `ith` ,
>
> `ith` ** ** :
>
> - `ruleKey == "type"` `ruleValue == typei`
> - `ruleKey == "color"` `ruleValue == colori`
> - `ruleKey == "name"` `ruleValue == namei`
>
> * *

```python
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
```

यह कोड एक समस्या को हल करता है जहां आपको `items` नामक एक सूची दी जाती है, जिसमें प्रत्येक आइटम एक सूची होती है जिसमें `type`, `color`, और `name` शामिल होते हैं। `ruleKey` और `ruleValue` के आधार पर, आपको उन आइटम्स की संख्या गिननी होती है जो दिए गए नियम से मेल खाते हैं।

- `ruleKey` यह बताता है कि किस प्रॉपर्टी (`type`, `color`, या `name`) के आधार पर मिलान करना है।
- `ruleValue` वह मान है जिसके साथ मिलान करना है।

कोड का काम निम्नलिखित है: 1. `ruleKey` के आधार पर यह निर्धारित करता है कि `items` की किस इंडेक्स (0, 1, या 2) की जांच करनी है। 2. फिर यह `items` की सूची में से प्रत्येक आइटम की जांच करता है और देखता है कि क्या उस आइटम का चयनित इंडेक्स `ruleValue` के बराबर है। 3. यदि मिलान होता है, तो `n` की संख्या बढ़ा दी जाती है। 4. अंत में, `n` को वापस कर दिया जाता है, जो मिलान करने वाले आइटम्स की संख्या को दर्शाता है।

□□□□□□(□□□□□□□□□□().□□□□□□□□□□□□□□□([[“□□□□□□”,“□□□□□”,“□□□□□□”],[“□□□□□□□□□”,“□□□□□□”, “□□□□□□”))

```
## 1365.

>         `nums` ,     `nums[i]`                                  ,     `nums[i]`          `j's`
>
>


> ```
>   : 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
```

यह कोड एक सूची nums में दिए गए प्रत्येक संख्या के लिए उससे छोटे संख्याओं की संख्या गिनता है और उसे एक नई सूची ns में संग्रहीत करता है। अंत में, यह सूची ns को वापस लौटाता है।

□□□□□(□□□□□□□□□().□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□([8,1,2,2,3]))

528ms   , 11.81%

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

यह कोड एक □□□□□□ क्लास Solution को परिभाषित करता है जिसमें एक मेथड smallerNumbersThanCurrent है। यह मेथड एक इनपुट लिस्ट nums लेता है, जो पूर्णांकों (□□□□□□□□□) की एक सूची है, और एक आउटपुट लिस्ट रिटर्न करता है। आउटपुट लिस्ट में प्रत्येक तत्व यह दर्शाता है कि इनपुट लिस्ट में उस तत्व से छोटे कितने तत्व हैं।

अभी कोड अधूरा है, और इसे पूरा करने के लिए और लॉजिक जोड़ने की आवश्यकता है।

```
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)
```

हिंदी व्याख्या:

1. `ins = list(range(l))`:
   - यह कोड `ins` नामक एक सूची बनाता है जिसमें `0` से `l-1` तक के नंबर होते हैं। यह सूची मूल इंडेक्स को स्टोर करने के लिए उपयोग की जाती है।
2. **बाहरी लूप (`for i in range(l)`)**:
   - यह लूप `i` को `0` से `l-1` तक चलाता है। यह सूची के प्रत्येक तत्व को एक-एक करके चुनता है।
3. **आंतरिक लूप (`for j in range(i+1, l)`)**:
   - यह लूप `j` को `i+1` से `l-1` तक चलाता है। यह `i` के बाद के सभी तत्वों को चुनता है।
4. **स्वैपिंग (`if sort_nums[i] > sort_nums[j]`)**:
   - यदि `sort_nums[i]` का मान `sort_nums[j]` से बड़ा है, तो दोनों तत्वों को स्वैप किया जाता है। इसके साथ ही, `ins` सूची में भी संबंधित इंडेक्स को स्वैप किया जाता है।
5. `smalls = [0]`:
   - यह `smalls` नामक एक सूची बनाता है जिसमें पहला तत्व `0` होता है। यह सूची छोटे तत्वों की संख्या को स्टोर करने के लिए उपयोग की जाती है।
6. **दूसरा लूप (`for i in range(1, l)`)**:

    - यह लूप `i` को `1` से `l-1` तक चलाता है। यह सूची के प्रत्येक तत्व को एक-एक करके चुनता है।

7. `if sort_nums[i-1] == sort_nums[i]:`
    - यदि वर्तमान तत्व और पिछला तत्व समान हैं, तो `smalls` सूची में पिछले तत्व का मान जोड़ा जाता है।

8. `else:`
    - यदि वर्तमान तत्व और पिछला तत्व समान नहीं हैं, तो `smalls` सूची में `i` का मान जोड़ा जाता है।

इस कोड का उद्देश्य `sort_nums` सूची को सॉर्ट करना है और साथ ही `ins` सूची में मूल इंडेक्स को भी अपडेट करना है। इसके बाद, `smalls` सूची में यह स्टोर किया जाता है कि कितने तत्व वर्तमान तत्व से छोटे हैं।

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

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

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

यह कोड एक लिस्ट `r_is` बनाता है जो `0` से `l-1` तक की संख्याओं से भरी होती है। फिर यह `ins` लिस्ट के माध्यम से लूप करता है और `r_is` लिस्ट को अपडेट करता है। अंत में, यह `smalls` लिस्ट से मान लेकर एक नई लिस्ट `ns` बनाता है और उसे वापस करता है।

□□□□□(□□□□□□□□□().□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□([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
```

इस कोड को हिंदी में समझाएं:

यह कोड एक क्लास `Solution` को परिभाषित करता है जिसमें एक मेथड `smallerNumbersThanCurrent` है। यह मेथड एक इनपुट लिस्ट `nums` लेता है, जो पूर्णांकों (□□□□□□□□□) की एक लिस्ट है, और एक आउटपुट लिस्ट रिटर्न करता है जिसमें प्रत्येक तत्व के लिए यह दर्शाता है कि मूल लिस्ट में उस तत्व से छोटे कितने तत्व हैं।

1. `sort_nums = nums.copy()`: यह लाइन `nums` लिस्ट की एक कॉपी बनाती है और इसे `sort_nums` में स्टोर करती है।
2. `sort_nums.sort()`: यह लाइन `sort_nums` लिस्ट को सॉर्ट करती है, यानी इसे आरोही क्रम (□□□□□□□□□□ □□□□□) में व्यवस्थित करती है।
3. `ns = []`: यह एक खाली लिस्ट `ns` को इनिशियलाइज़ करती है जो अंतिम परिणाम स्टोर करेगी।
4. `for num in nums:`: यह लूप `nums` लिस्ट के प्रत्येक तत्व के लिए चलता है।
5. `ns.append(sort_nums.index(num))`: यह लाइन `sort_nums` लिस्ट में `num` की इंडेक्स को ढूंढती है और इसे `ns` लिस्ट में जोड़ती है। चूंकि `sort_nums` सॉर्टेड है, इसलिए `num` की इंडेक्स यह दर्शाती है कि `nums` लिस्ट में `num` से छोटे कितने तत्व हैं।
6. `return ns`: अंत में, यह लाइन `ns` लिस्ट को रिटर्न करती है, जिसमें प्रत्येक तत्व के लिए छोटे तत्वों की संख्या होती है।

उदाहरण के लिए, यदि `nums = [8, 1, 2, 2, 3]` है, तो `sort_nums` होगा `[1, 2, 2, 3, 8]`। `ns` लिस्ट में प्रत्येक तत्व के लिए `sort_nums` में उसकी इंडेक्स होगी, जो `[4, 0, 1, 1, 3]` होगी। यह दर्शाता है कि `8` से छोटे 4 तत्व हैं, `1` से छोटे 0 तत्व हैं, `2` से छोटे 1 तत्व है, और इसी तरह।

□□□□□(□□□□□□□□□().□□□□□□□□□□□□□□□□□□□□□□□□□□□□□([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
```

इस कोड में, `Solution` क्लास में एक मेथड `smallerNumbersThanCurrent` है जो एक लिस्ट `nums` लेता है और एक नई लिस्ट `ns` रिटर्न करता है। `ns` में प्रत्येक इंडेक्स पर वह संख्या होती है जो दर्शाती है कि `nums` में उस इंडेक्स की संख्या से छोटी कितनी संख्याएँ हैं।

उदाहरण के लिए, यदि `nums = [8,1,2,2,3]` है, तो `ns` होगा `[4,0,1,1,3]`। यह इसलिए क्योंकि: - 8 से छोटी संख्याएँ हैं: 1, 2, 2, 3 (कुल 4) - 1 से छोटी कोई संख्या नहीं है (कुल 0) - 2 से छोटी संख्या है: 1 (कुल 1) - 2 से छोटी संख्या है: 1 (कुल 1) - 3 से छोटी संख्याएँ हैं: 1, 2, 2 (कुल 3)

इस कोड में दो नेस्टेड लूप्स हैं जो `nums` लिस्ट के हर जोड़े की तुलना करते हैं और `ns` लिस्ट को अपडेट करते हैं।

□□□□□(□□□□□□□□□().□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□([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))
```

यह कोड एक `Solution` क्लास को परिभाषित करता है जिसमें एक मेथड `smallerNumbersThanCurrent` है। यह मेथड एक इनपुट लिस्ट `nums` लेता है और एक आउटपुट लिस्ट रिटर्न करता है जिसमें प्रत्येक एलिमेंट के लिए यह बताया जाता है कि उससे छोटे कितने एलिमेंट्स हैं।

`ss` वेरिएबल में, `nums` लिस्ट के एलिमेंट्स को उनके इंडेक्स के साथ जोड़कर सॉर्ट किया जाता है। यह सॉर्टिंग एलिमेंट्स के मान के आधार पर होती है।

यह कोड एक सूची `nums` की लंबाई `l` निकालता है और फिर एक नई सूची `smalls` बनाता है। `smalls` सूची में प्रत्येक इंडेक्स `i` के लिए, यह जाँचता है कि `ss` सूची में पिछले इंडेक्स `i-1` और वर्तमान इंडेक्स `i` के तत्व `e0` और `e1` समान हैं या नहीं। यदि वे समान हैं, तो `smalls` में पिछले इंडेक्स `i-1` का मान जोड़ा जाता है। यदि वे समान नहीं हैं, तो `smalls` में वर्तमान इंडेक्स `i` जोड़ा जाता है।

हिंदी में समझाएं:

```
l = len(nums)  # nums
smalls = [0]   # smalls         ,       0
```

```
for i in range(1, l):  # 1   l-1
    (e0, j0) = ss[i-1]  # ss
    (e1, j1) = ss[i]    # ss

    if e0 == e1:  #
        smalls.append(smalls[i-1])  # smalls
    else:  #
        smalls.append(i)  # smalls
```

इस कोड का उद्देश्य `smalls` सूची को इस तरह से भरना है कि यदि `ss` सूची में लगातार दो तत्व समान हों, तो `smalls` में उसी इंडेक्स का मान दोहराया जाए, अन्यथा नया इंडेक्स जोड़ा जाए।

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

यह कोड एक सूची `ns` को शून्य से प्रारंभ करता है जिसकी लंबाई `l` है। फिर यह एक लूप के माध्यम से `ss` सूची के प्रत्येक तत्व को पढ़ता है, जहां प्रत्येक तत्व एक टपल `(e, j)` है। `j` का उपयोग `ns` सूची में इंडेक्स के रूप में किया जाता है, और `smalls[i]` का मान `ns[j]` में असाइन किया जाता है। अंत में, `ns` सूची को वापस लौटाया जाता है।

□□□□□(□□□□□□□□□().□□□□□□□□□□□□□□□□□□□□□□□□□□□□□([8,1,2,2,3]))

```
>    : 52     , Python3             91.45%      "                      "
>
>       : 14.6 MB, Python3             15.18%      "                      "
```

```
            !                `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)

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))
```

यह कोड एक क्लास `Solution` को परिभाषित करता है जिसमें एक मेथड `smallerNumbersThanCurrent` है। यह मेथड एक लिस्ट `nums` लेता है और एक नई लिस्ट रिटर्न करता है जिसमें प्रत्येक एलिमेंट के लिए यह दर्शाता है कि मूल लिस्ट में उससे छोटे कितने एलिमेंट्स हैं।

`ss` वेरिएबल में, `nums` लिस्ट के एलिमेंट्स को उनके इंडेक्स के साथ जोड़कर सॉर्ट किया जाता है। यह सॉर्टिंग एलिमेंट्स के मान के आधार पर की जाती है।

```
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
```

यह कोड `ns` नामक सूची में `j1` इंडेक्स पर `last` वैल्यू को असाइन करता है और फिर `ns` सूची को वापस लौटाता है।

□□□□□(□□□□□□□□□().□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□([8,1,2,2,3]))

```
       `40ms`         ,    `99.81%`

>    : 40     , Python3              99.81%     , "How Many Numbers Are Smaller Than the Current Number
>
>       : 14.4 MB, Python3              15.18%     , "How Many Numbers Are Smaller Than the Current Nu
```

```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
```

यह कोड एक समस्या को हल करने के लिए है जहां हमें एक सूची `nums` दी जाती है और हमें यह पता लगाना होता है कि प्रत्येक संख्या से छोटी संख्याओं की संख्या कितनी है।

1. `l = len(nums)` - यह सूची `nums` की लंबाई को `l` में स्टोर करता है।

2. `n = [0] * 101` - यह एक सूची `n` बनाता है जिसमें 101 शून्य होते हैं। यह सूची संख्याओं की गिनती को स्टोर करने के लिए उपयोग की जाएगी।
3. `max_num = 0` - यह वेरिएबल `max_num` को 0 पर सेट करता है, जो सूची में सबसे बड़ी संख्या को ट्रैक करेगा।
4. `for num in nums:` - यह लूप सूची `nums` में प्रत्येक संख्या के लिए चलता है।
    - `n[num] += 1` - यह संख्या `num` की गिनती को सूची `n` में बढ़ाता है।
    - `if num > max_num:` - यह जांचता है कि क्या वर्तमान संख्या `num` अब तक की सबसे बड़ी संख्या `max_num` से बड़ी है।
        - `max_num = num` - यदि हां, तो `max_num` को वर्तमान संख्या `num` पर अपडेट करता है।

इस कोड का उद्देश्य यह है कि हम प्रत्येक संख्या के लिए यह पता लगा सकें कि सूची में उससे छोटी कितनी संख्याएं हैं।

```
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]]

  return 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
```
```

यह कोड एक क्लास `Solution` को परिभाषित करता है जिसमें एक मेथड `smallerNumbersThanCurrent` है। यह मेथड एक इनपुट लिस्ट `nums` लेता है और एक आउटपुट लिस्ट रिटर्न करता है जिसमें प्रत्येक एलिमेंट के लिए यह बताया जाता है कि उससे छोटे कितने नंबर हैं।

1. `l = len(nums)`: इनपुट लिस्ट `nums` की लंबाई को `l` में स्टोर करता है।
2. `n = [0] * 101`: एक लिस्ट `n` बनाता है जिसमें 101 एलिमेंट हैं, सभी शुरू में 0 से इनिशियलाइज़ होते हैं। यह लिस्ट प्रत्येक नंबर की फ्रीक्वेंसी को स्टोर करेगी।
3. `max_num = 0`: `max_num` वेरिएबल को 0 से इनिशियलाइज़ करता है, जो इनपुट लिस्ट में सबसे बड़े नंबर को ट्रैक करेगा।
4. `for num in nums`: इनपुट लिस्ट `nums` के प्रत्येक एलिमेंट के लिए लूप चलाता है।
    - `n[num] += 1`: `n` लिस्ट में `num` इंडेक्स पर वैल्यू को 1 से इन्क्रीमेंट करता है, यह `num` की फ्रीक्वेंसी को ट्रैक करता है।
    - `if num > max_num: max_num = num`: यदि `num` `max_num` से बड़ा है, तो `max_num` को `num` के साथ अपडेट करता है।

यह कोड अभी पूरा नहीं हुआ है, लेकिन यह इनपुट लिस्ट में प्रत्येक नंबर की फ्रीक्वेंसी और सबसे बड़े नंबर को ट्रैक कर रहा है।

```
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
```

यह कोड एक सूची `short_n` और `short_num` को प्रारंभ करता है, और एक सूची `zn` को 101 शून्य मानों के साथ प्रारंभ करता है। फिर यह `max_num+1` तक एक लूप चलाता है। यदि `n[i]` का मान 0 से अधिक है, तो यह `zn[i]` को `j` के मान से सेट करता है, `short_n` में `n[i]` को जोड़ता है, और `short_num` में `num` को जोड़ता है। अंत में, `j` को 1 से बढ़ाया जाता है।

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

ns = [0] * l
```

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

यह कोड निम्नलिखित कार्य करता है:

1. `sm` नामक एक सूची बनाई जाती है जिसमें `j` शून्य होते हैं।
2. `sum` नामक एक वेरिएबल को 0 पर सेट किया जाता है।
3. एक लूप `j` बार चलता है और `sm` सूची में `sum` का मान डालता है, फिर `sum` में `short_n[i]` को जोड़ता है।
4. `ns` नामक एक सूची बनाई जाती है जिसमें `l` शून्य होते हैं।
5. एक लूप `l` बार चलता है और `ns` सूची में `sm[zn[nums[i]]]` का मान डालता है।
6. अंत में, `ns` सूची को वापस लौटाया जाता है।

□□□□□□(□□□□□□□□().□□□□□□□□□□□□□□□□□□□□□□□□□([8,1,2,2,3]))

~~~
```python
class Solution:

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

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

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

यह कोड एक खाली सूची `sorted_ls` बनाता है और फिर `0` से `max_num` तक के सभी नंबरों के लिए लूप चलाता है। यदि `n[i]` (यानी `n` सूची में `i` इंडेक्स पर मौजूद मान) `0` से बड़ा है, तो `i` को `sorted_ls` सूची में जोड़ दिया जाता है। इस तरह, `sorted_ls` सूची में उन इंडेक्स के मान शामिल होते हैं जिनके लिए `n` सूची में मान `0` से अधिक होता है।

```python
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
```

अभ्यास

- छात्र ऊपर दिए गए उदाहरण की तरह कुछ प्रश्नों को हल कर सकते हैं।