## **https://blog.techbridge.cc/2019/08/30/leetcode-pattern-two-pointer/**

## 

## **前言**

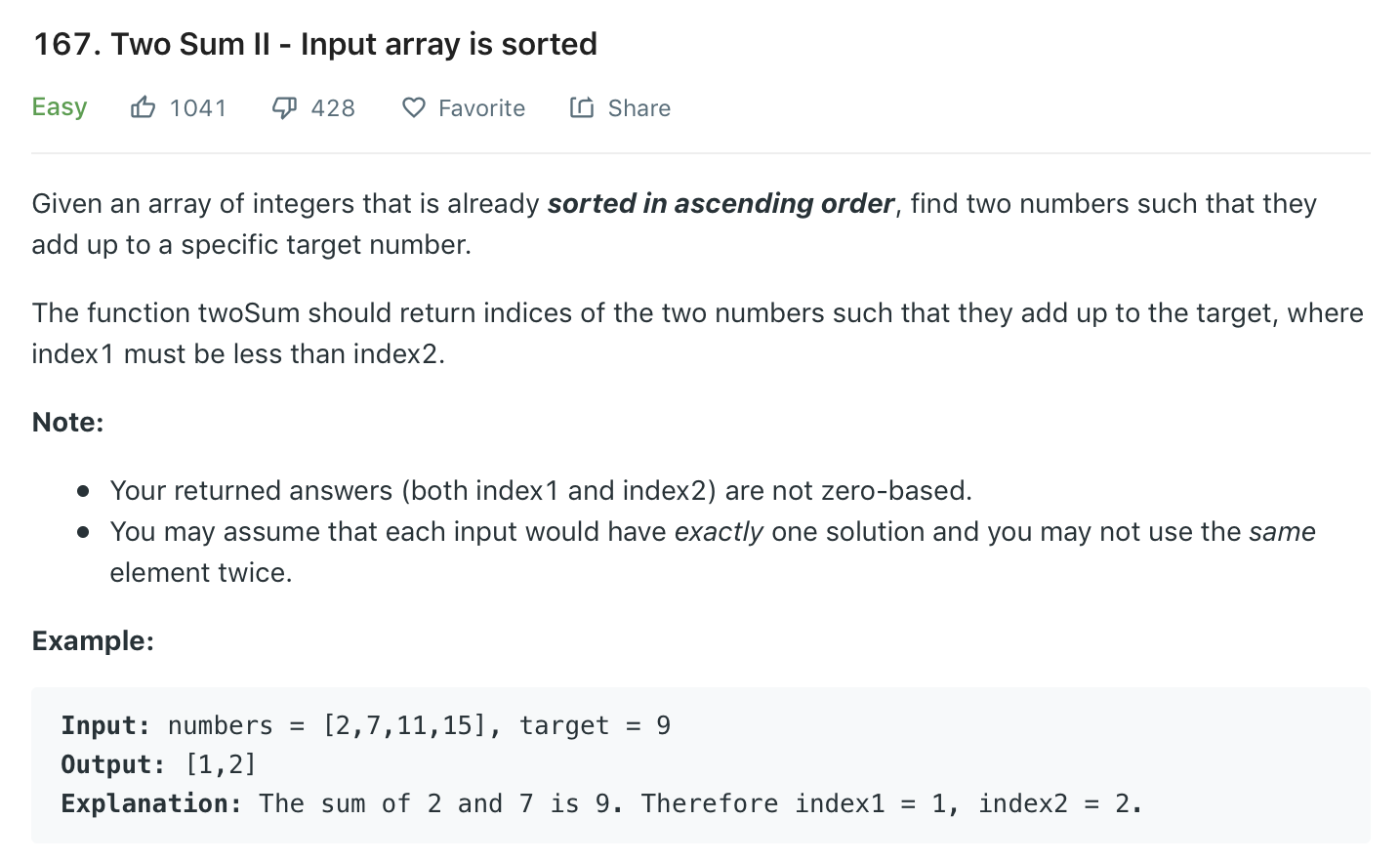
身在大 CS 時代，有越來越多人投入刷題的行列，在眼花撩亂的題海中，要想有效率地刷題，掌握並通達題目解法背後，可以不斷被拿來使用的觀念，才能做到以簡禦繁。

今天就要跟大家介紹一種演算法的 pattern - Two Pointer。

## **Two Pointer 的第一個範例 - Leetcode #167 Two Sum II**

### **題目**

我們先看一下題目的敘述：



輸入是一個 array，裡面是已經排好序的 int，剩下就是要找到加總起來等於 target 的一組 index。

### **暴力法**

最最最直覺的暴力法，就是每一組搭配都試試看，所以 pseudo code 會像下面這樣：

For every i in numbers start from index 0:

For every j in numbers start from index i+1:

If(numbers[i]+numbers[j] equals to target)

Return i&j

Endif

Endfor

Endfor

如果 array 的長度是 n，那時間複雜度就是

O(

n

2

)

O(n2)。也就是說，如果你的字串長度是 1,000，最糟得跑這個迴圈 1,000,000 次(這邊是粗算，當然不用那麼多，因為 j 是從 i+1 開始，只是抓個大概)！

### **好一點的解法 - Hash Table**

有寫過 Leetcode 天字第一題 Two Sum 的朋友一定很直覺想到，要避免

O(

n

2

)

O(n2) 還不簡單，用個 Hash Table 就好了嘛。（如果你不知道在說什麼，可以看看 [之前寫過的這篇文章](https://blog.techbridge.cc/2017/01/21/simple-hash-table-intro/)）

沒錯！如果用 Hash Table，確實可以讓時間複雜度降到

O(n)

O(n)，如此一來，如果你的 array 長度是 1,000，你只需要跑 1,000 個迴圈，假設你自己是電腦裡面的 CPU 小精靈（?），要負責跑迴圈，你從需要跑 1,000,000 圈到變成只要跑 1,000 圈，是不是會謝天謝地謝上帝ＸＤ

不過如果你是記憶體小精靈（?），你可能會覺得很煩，剛剛明明只有 CPU 小精靈要跑 1,000,000 圈，你正想說趁他在跑，你要去跑 UberEats，結果現在你也多了 1,000 個東西（最多）要管理。你可能想說，能不能再想想！不要因你的懶惰叫我多做事！

### **Two Pointer 解法**

聽到了記憶體小精靈的呼喚，Two Pointer 演算法出來救援了。在這題中有個很巧妙的性質，就是 array 裡面的東西是排好序的，如果只能用跟 Two Sum 一樣的解法，幹嘛要多給排序好的這個大特惠？

我們來看一下範例：

numbers = [2,7,11,15]

target = 9

先想想，如果我用左右手兩根食指，左手指向 2，右手指向 15，這時兩根手指的數字加起來是 17，比 9 還大，這表示什麼？是左手指的數字有問題嗎？

肯定不是啊，因為 array 已經排過序了，左手指向的 2 是最小的，我們希望兩根手指加起來的數字等於 9，也就是說右手指的數字太大了！

所以接下來我們會：

* 把右手往左移，指向 11; 這時 2+11 == 13 還是比 9 大
* 把右手再往左移，指向 7; 這時 2+7 == 9，找到了！

想不到動兩根手指就可以這麼快找到！這就是 Two Pointer 的魅力，這個演算法的時間複雜度是

O(n)

O(n)，空間複雜度是

O(1)

O(1)（不管字串有多長，我都只需要兩根手指啦！）。

class Solution {

public:

vector<int> twoSum(vector<int>& nums, int target) {

if(nums.size() < 2) return {};

int l = 0, r = nums.size()-1;

while(l <= r) {

if(nums[l] + nums[r] == target) {

*// index start from 1, so we need to add 1*

return {l+1, r+1};

}

else if(nums[l] + nums[r] > target) r--;

else if(nums[l] + nums[r] < target) l++;

}

return {};

}

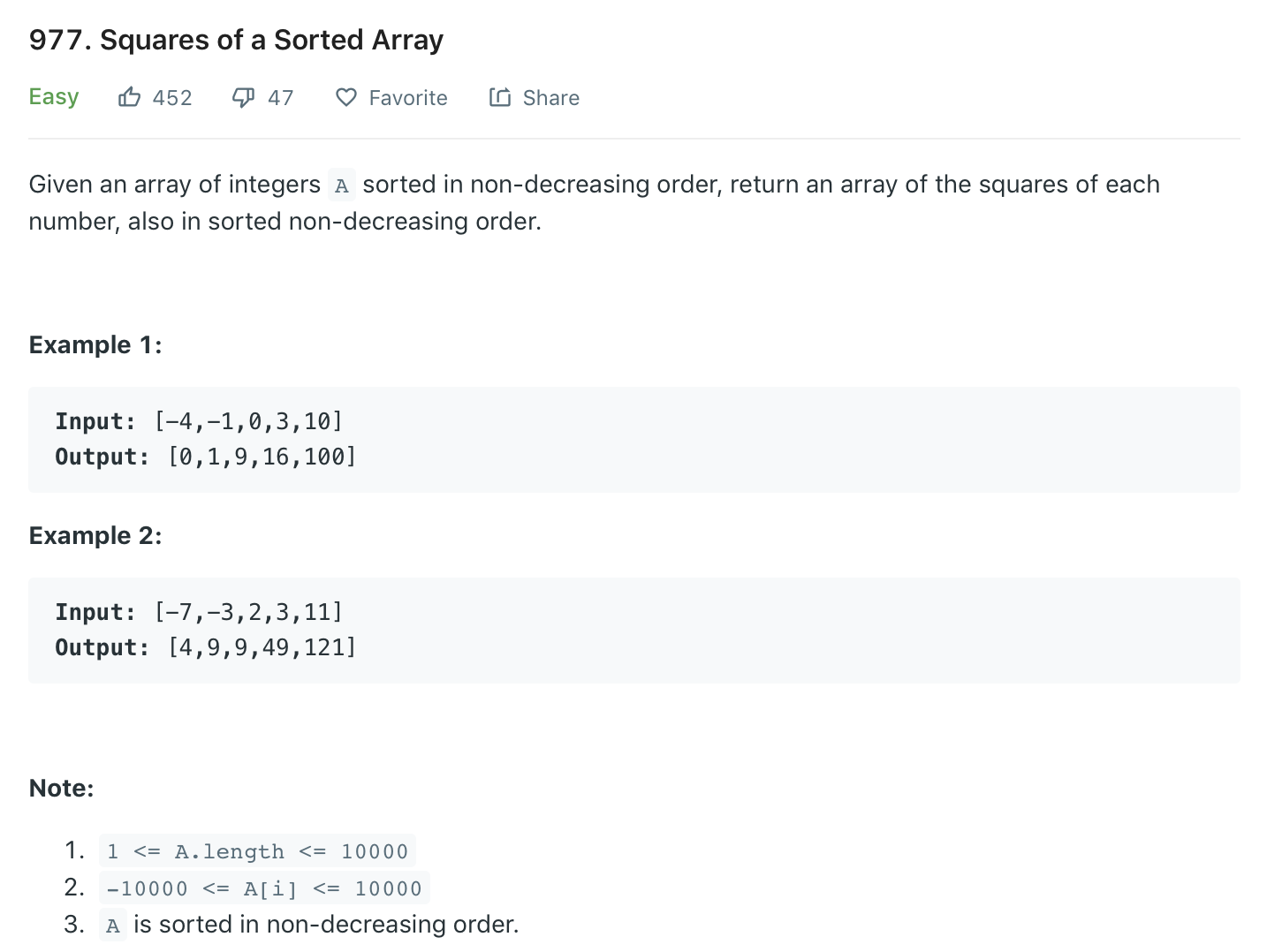
};

看到這邊是不是初步體驗到 Two Pointer 的厲害了呢，不過這個 pattern 如果只有一題適用，那就不用特別講了，讓我們繼續看下去。

## **Two Pointer 的第二個範例 - Leetcode #977 Squares of a Sorted Array**

### **題目**

這一題的敘述在這：



### **暴力法**

這題的暴力法應該很直觀，我們先把每一個 element 的平方數都算出來，存在原本的 array，然後 sort 這個 array 就好。這樣子的時間複雜度是

O(nlogn)

O(nlogn)，但這時我們要問 - Can we do better？

答案是 Yes, I do we can！

### **Two Pointer 解法**

Two Pointer 的解法跟上一題的精神有異曲同工之妙，首先，我們要注意到這題的輸入是一個 non-decreasing array，也就是等於已經排好序啦！所以，我們一樣可以用左右手兩根食指，左手指到最前面，右手指到最後面，然後開始比較。

唯一一個不一樣的地方是，原本的 array 中有負數，但既然已經排好序，我們只要比較兩邊的平方數就好，看個範例吧：

numbers = [-4,-1,0,3,10]

初始化，拿出你的手指，左手指向 -4，右手指向 10，這時兩根手指的平方數分別是 16 跟 100，因為 100 比較大，所以我們把它放到答案 array 的尾端。

然後接下來大家大概都會了：

* 把右手往左移，指向 3; 這時 16 > 9，把 16 放到答案 array 的尾端 - 1 的位置
* 把左手往右移，指向 -1; 這時 9 > 1，把 9 放到答案 array 的尾端 - 2 的位置
* 把右手往左移，指向 0; 這時 1 > 0，把 1 放到答案 array 的尾端 - 3 的位置
* 把左手往右移，指向 0；這時兩邊指的都一樣，直接把 0 放到答案 array 的尾端 - 4 的位置

是不是也很簡單呢？而且時間複雜度就降到

O(n)

O(n) 了。

這時再看個 code，覺得棒：

class Solution {

public:

vector<int> sortedSquares(vector<int>& arr) {

int n = arr.size();

vector<int> squares(n);

int highestIdx = n-1;

int l=0, r=n-1;

while(l <= r) {

int leftSquare = pow(arr[l], 2);

int rightSquare = pow(arr[r], 2);

if(leftSquare >= rightSquare) {

squares[highestIdx--] = leftSquare;

l++;

}

else {

squares[highestIdx--] = rightSquare;

r--;

}

}

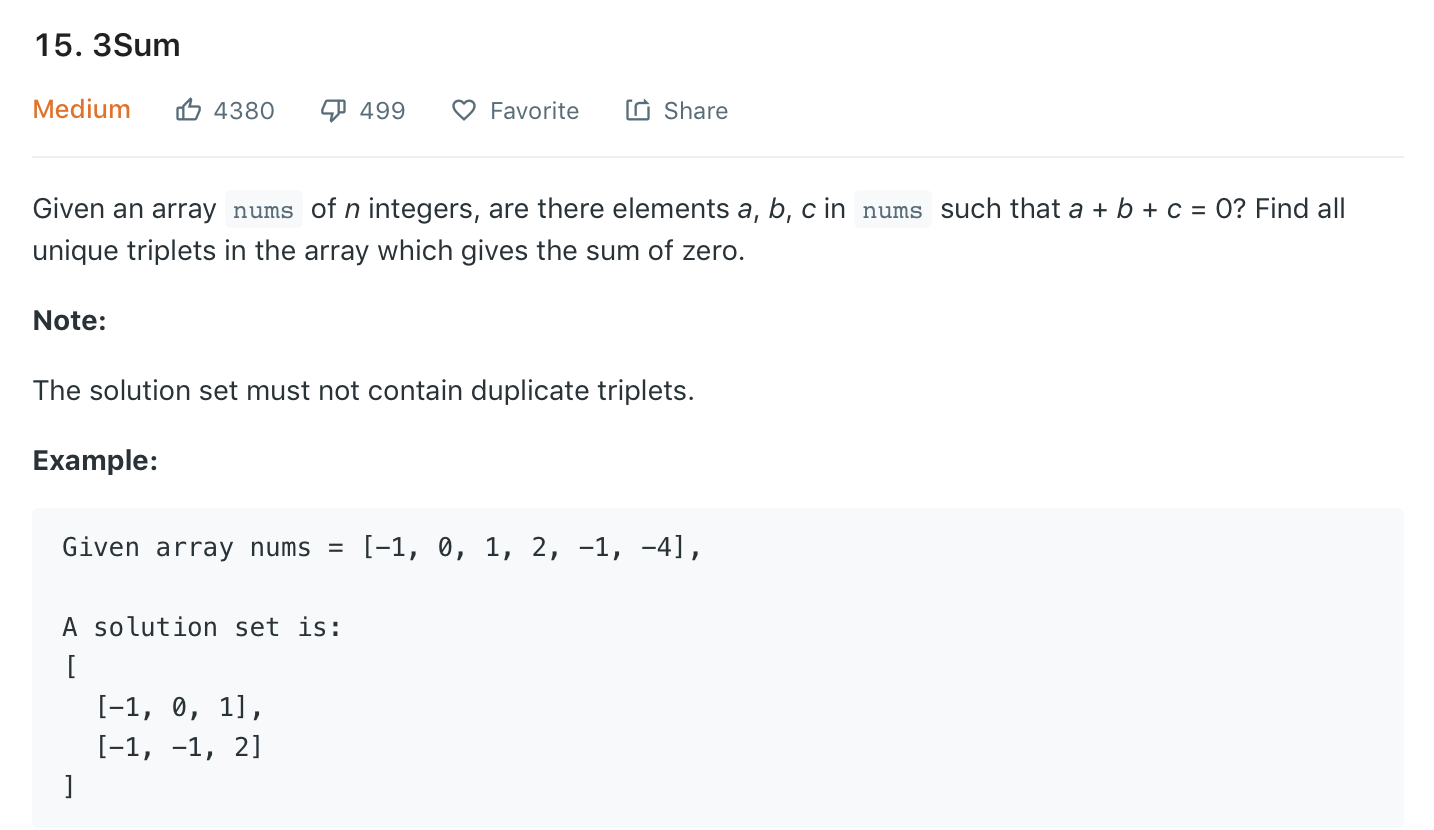
return squares;

}

};

## **Two Pointer 的第三個範例 - Leetcode #15 3Sum**

### **題目**

****

### **暴力法**

一樣，我們先上最直覺的暴力法，也就是每一組搭配都試試看，pseudo code 會像下面這樣：

ans = empty array

For every i in nums start from index 0:

For every j in nums start from index i+1:

For every k in nums start from index j+1:

If(nums[i]+nums[j]+nums equals to 0)

Add {nums[i], nums[j], nums[k]} to the ans

Endif

Endfor

Endfor

Endfor

return ans

如果 array 的長度是 n，那時間複雜度就是

O(

n

3

)

O(n3)。

### **Two Pointer 解法**

寫到這邊，廢話就不多說，我們就來看看怎麼優化。首先仔細觀察一下題目，這題的輸入是一個沒有 sort 過的 array，這時心裏可能會想說 GG 思密達，看來應該是沒 Two Pointer 的戲了。

但，我們是程式設計師，輸入沒有 sort 難道我們不能自己 sort 嗎？當然可以，可是要注意一件事，如果我們要輸出的答案是 index，那就麻煩了。不過很幸運地，我們要輸出的是數值！（這邊就發現題目設計者可能，就是，偷偷地想要讓你可以 sort）

所以我們可以先把輸入的 array sort 完，然後依序把每個 element 當作 nums[i]，剩下就是尋找 nums[j]+nums[k] == -nums[i]。恭喜恭喜，變回剛剛學過的 #167 了。

讓我們實作出下面的程式碼：

class Solution {

public:

vector<vector<int>> threeSum(vector<int>& nums) {

vector<vector<int>> triplets;

int n = nums.size();

sort(nums.begin(), nums.end());

*// Note:*

*// 1. Iterate to n-2 only*

*// 2. Need to skip same element to avoid duplicate triplets*

for(int i=0; i<n-2; i++) {

*// skip same element to avoid duplicate triplets*

if (i > 0 && nums[i] == nums[i - 1]) {

continue;

}

searchPair(nums, -nums[i], i+1, triplets);

}

return triplets;

}

private:

void searchPair(vector<int>& nums, int target, int start, vector<vector<int>>& triplets) {

int l=start, r=nums.size()-1;

while(l<r) {

if(nums[l]+nums[r] == target) {

triplets.push\_back({-target, nums[l], nums[r]});

*// Intuitively, we should only do l++ or r--*

*// so that we won't miss the case of nums[l]+nums[r-1] or nums[l+1]+nums[r]*

*// But think deeper, if we only do r--*

*// After preventing duplicates, we know that*

*// nums[l] and nums[new r] cannot fulfill sum==target*

*// So we won't miss anything even we do l++ and r--*

l++;

r--;

while (l < r && nums[l] == nums[l-1]) {

l++; *// skip same element to avoid duplicate triplets*

}

while (l < r && nums[r] == nums[r+1]) {

r--; *// skip same element to avoid duplicate triplets*

}

}

else if(nums[l]+nums[r] > target) r--;

else if(nums[l]+nums[r] < target) l++;

}

}

};

在實作到程式階段會有比較多的小細節要注意，不過整體思想其實相當簡單。

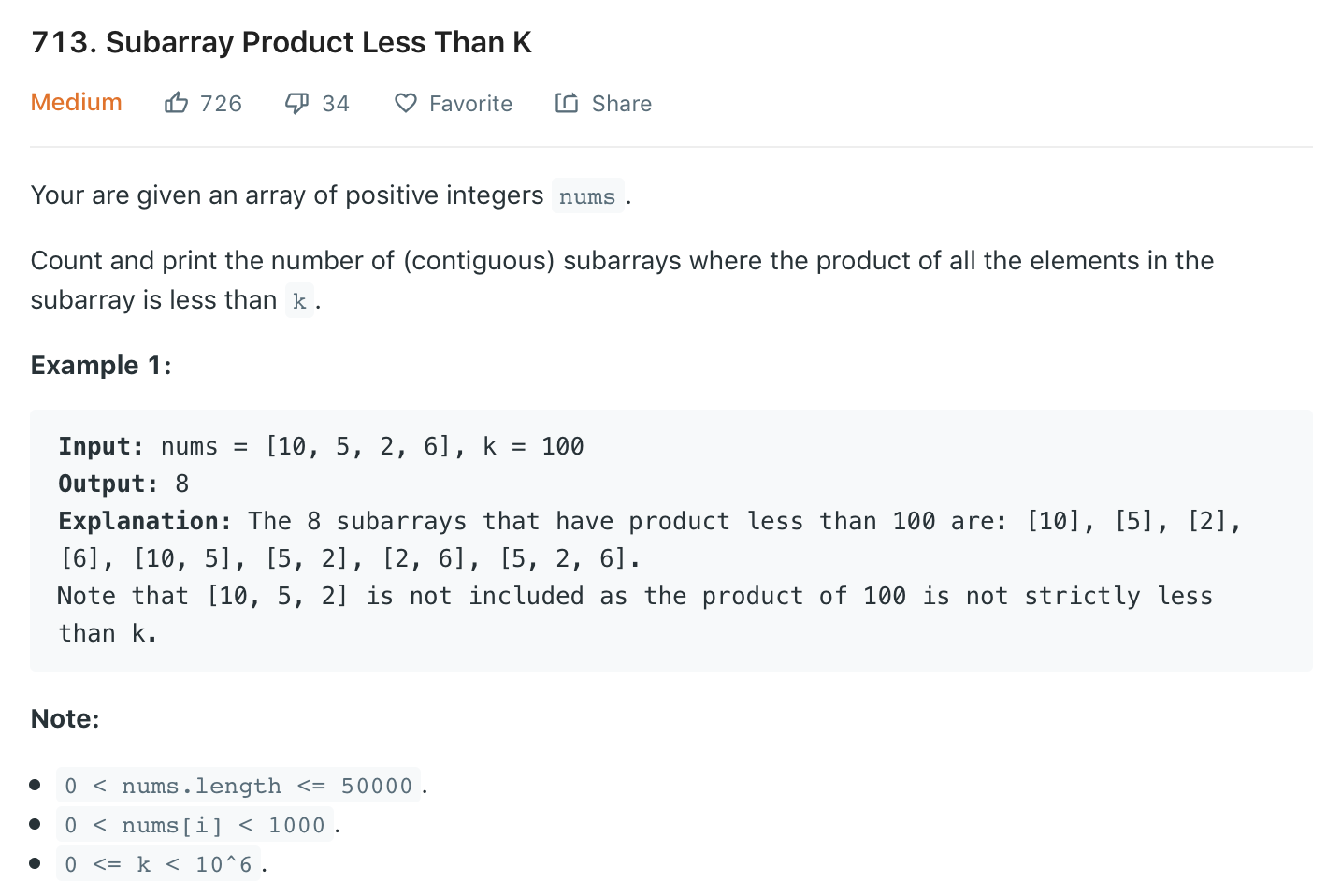
### **3Sum 的小延伸**

如果這題你懂了，也可以再去寫一寫 #16 - 3Sum Closest、# 259 - 3Sum Smaller，每次寫也可以想想如果你用暴力解，演算法的效率會差多少，你就會越來越欣賞 Two Pointer 囉！

而且，學會一個技巧就可以打掉 3 題 leetcode 題是不是有點開心ＸＤ

## **Two Pointer 的第四個範例 - Leetcode #713 Subarray Product Less Than K**

### **題目**

****

### **暴力法**

這題的暴力法也很直覺，列舉出所有的 subarray 就對了！列舉出每個 subarray 後，都把那個 subarray 的 element 乘起來，看會不會比 k 大。不過這樣做的時間複雜度是

O(

n

3

)

O(n3)，因為要搜尋每個起終點位置的組合就已經要花

O(

n

2

)

O(n2) 時間，然後每一個 subarray 要花

O(n)

O(n) 的時間計算乘積，整個就非常拖。

### **Two Pointer 解法**

如果我們使用 Two Pointer 來形成一個 sliding window，就能夠進一步地節省時間，基本的想法是要避免：

1. 每次都得重新取 Window
2. 每次都得重新乘積

讓我們先看一個例子：

nums = [10, 5, 2, 6]

k = 100

1. product 一開始是 1，ans 一開始是 0。
2. 假設一開始左手指向 10，右手也指向 10，這時候 product \*= 10 == 10，所以我們知道可以把 ans += 1 == 1。
3. 把右手指往右移，右手指向 5，這時候 product \*= 5 == 50，因為依然小於 k，所以我們知道 [10, 5] 跟 [5] 這兩個 subarray 的乘積都小於 k。 所以可以把 ans += (r-l+1) == 1 + (1-0+1) == 3。（r 表示右手指位置，l 則表示左手指位置）

依此類推，接著把右手往右移，如果 product > k，那就得把 product /= 左手指向的值，並把左手往右移，直到 product < k，這時就又可以再把 ans += (r-l+1)。當右手指超過 array 範圍就算完了。

所以要避免暴力法冗餘部分的方法就是：

1. 避免每次都得重新取 Window：當發現某個 subwindow 的乘積比 k 小時，就知道可以直接加上 r-l+1 個 subarray
2. 避免每次都得重新乘積：移動右手指時，product \*= nums[r]；移動左手指時，product \= nums[l]

程式碼非常簡潔：

class Solution {

public:

int numSubarrayProductLessThanK(vector<int>& nums, int k) {

if(k <= 1) return 0;

int n=nums.size();

int count = 0, product = 1, l = 0;

for(int r=0; r<n; r++) {

product \*= nums[r];

while(product >= k) {

product /= nums[l];

l++;

}

count += r-l+1;

}

return count;

}

};

用 Two Pointer 構成一個 Sliding window 的話，就只需要

O(n)

O(n) 的時間囉！差超多！

## **總結**

當我們在處理 sorted array 或 sorted linked list，而且需要找到一組滿足特定條件的 element 時，就很可能可以使用 Two pointer 來加快速度。

一組 element 可以是：

1. 一個 pair
2. 3 個 element
3. 一個 subarray

上面的幾個例子分別對應到

1. 一個 pair：#167
2. 3 個 element：#15
3. 一個 subarray：#713

希望大家看完之後，可以感受到 Two Pointer 的方便和效率。體驗到這個演算法好用、厲害，才會在該用的時候，自然而然地使用，比起用背的（例如看到...，就要用...），我覺得去體驗通達各種解法，覺得酷到不自覺笑出來、感受到讚讚讚，可能就是讓演算法功力進到下一個境界的現象。

上面提供的四題是讓大家初步體會一下 Two Pointer 的威力，如果你對這個 pattern 有興趣，可以再去看看延伸閱讀的筆記，裡面記錄了 12 題 Two Pointer 的題目，而且從簡單到越來越難，如果把這些題目一次寫完，對於 Two Pointer 的掌握度應該就大大提升了！

## **延伸閱讀**

1. [我的 Leetcode 刷題筆記 - Two Pointer pattern](https://po-jen.gitbooks.io/coding-practice-advanced-topics/content/two-pointer.html)
2. [Leetcode 刷題 pattern - Sliding Window](https://blog.techbridge.cc/2019/09/28/leetcode-pattern-sliding-window/)
3. [Leetcode 刷題 pattern - Next Greater Element](https://blog.techbridge.cc/2019/10/26/leetcode-pattern-next-greater-element/)
4. [Leetcode 刷題 pattern - Fast & Slow Pointer](https://blog.techbridge.cc/2019/11/22/leetcode-pattern-fast-and-slow-pointer/)
5. [Leetcode 刷題 pattern - Breadth-First Search](https://blog.techbridge.cc/2019/12/21/leetcode-%E5%88%B7%E9%A1%8C-pattern-breadth-first-search/)
6. [Leetcode 刷題 pattern - Merge Intervals](https://blog.techbridge.cc/2020/01/16/leetcode-%E5%88%B7%E9%A1%8C-pattern-merge-intervals/)
7. [Leetcode 刷題 pattern - Cyclic Sort](https://blog.techbridge.cc/2020/02/16/leetcode-%E5%88%B7%E9%A1%8C-pattern-cyclic-sort/)
8. [Leetcode 刷題 pattern - Two Heaps](https://blog.techbridge.cc/2020/03/15/leetcode-%E5%88%B7%E9%A1%8C-pattern-two-heaps/)
9. [Leetcode 刷題 pattern - Top K elements](https://blog.techbridge.cc/2020/04/12/leetcode-%E5%88%B7%E9%A1%8C-pattern-top-k-elements/)
10. [Leetcode 刷題 pattern - Topological Sort](https://blog.techbridge.cc/2020/05/10/leetcode-topological-sort/)
11. [Leetcode 刷題 pattern - Bitwise XOR](https://blog.techbridge.cc/2020/06/07/leetcode-pattern-bitwise-xor/)
12. [Leetcode 刷題 pattern - 一週年特典](https://blog.techbridge.cc/2020/07/05/leetcode-pattern-anniversary/)
13. [Leetcode 刷題pattern - 美國軟體工程師求職有趣經驗](https://blog.techbridge.cc/2020/12/15/leetcode-%E5%88%B7%E9%A1%8C-pattern-%E7%BE%8E%E5%9C%8B%E6%B1%82%E8%81%B7%E7%B6%93%E9%A9%97%E7%B8%BD%E7%B5%90/)

關於作者：

[@pojenlai](https://pojenlai.wordpress.com/) 演算法工程師，對機器人、電腦視覺和人工智慧有少許研究，正在學習[用心體會事物的本質](https://buzzorange.com/techorange/2017/07/10/elon-musk-first-principle/)跟[不斷進入學生心態改進](https://www.ted.com/talks/eduardo_briceno_how_to_get_better_at_the_things_you_care_about)。

[**#algorithm**](https://blog.techbridge.cc/tags/algorithm/) [**#Leetcode**](https://blog.techbridge.cc/tags/Leetcode/) [**#Software Engi**](https://blog.techbridge.cc/tags/Software%20Engineer/)