1. Given Array

2. find and return strictly increasing subarray.

$A = [1, 2, 3, 4, 5]$

o/p = 15

$1 \le |A| \le 10^5$

$1 \le A[i] \le 10^9$

$A = [9\ 9\ 4\ 2]$

ans = 9

$A = [1\ 2\ 3\ 9]$ o/p = 15

$A = [1\ 2\ 3\ 1\ 9]$

6 10 o/p = 10

ans =

Current sum = A[0]

(1 -> n)

if A[i] > A[i-1]

↓ Current sum += A[i]

else:

↓ ans = max(ans, currentSum)

currentSum = 0

A[1] > A[0]

currentSum = A[0]

if currentSum == 0

currentSum = A[i]

BF: 1. going through all subarrays
2. find the sum if elements are in increasing order

$$O(N^2) \times (N) = O(N^3)$$

optimized: $O(N)$

```
ans = 0
currentSum = 0
for i in range (N):
    if currentSum == 0:
        currentSum = A[i]
    else:
        if A[i] > A[i-1]:
            currentSum += A[i]
        else:
            ans = max(ans, currentSum)
            currentSum = A[i]
return max(ans, currentSum)
```

Dry run:

A = [1 2 3 1 9]
i

ans = ~~0~~ 6

currentSum = ~~0~~ ~~1~~ ~~3~~ ~~6~~ ~~1~~ 10

(2). Highest Product

1. Given int[] A
2. return highest product possible,

Product ⇒ multiplication 3 elements in A.

A = [0 -1 3 100 70 50]

⇒ 100 × 70 × 50

⇒ 3,50,000

a = ∅ 100
b = ✗ 70
c = ∅ 50

[100 1 9 7 8 50 2 90]

a = ∅ 100
b = ∅ ✗ 7 8 5 6 90
c = ∅ 9

BF: ⇒ 2 negative elements also can make highest positive value

⇒ by creating A[i] × A[j] × A[k]
⇒ which will take $O(n^3)$ Time

optimized:

⇒ 2 negative * 1 max positive

⇒ 3 positive

code:

```
A.sort()

option1 = A[0] * A[1] * A[n-1]

option2 = A[-1] * A[-2] * A[-3]

if option1 > option2:
    return option1
else: return option2
```

TC: O(nlogn)

SC: O(1)

0	1	2	3	4	5
0	-1	3	100	-50	-70

[-70, -50, -1, 0, 3, 100]

04 Librarian and Rotated Array

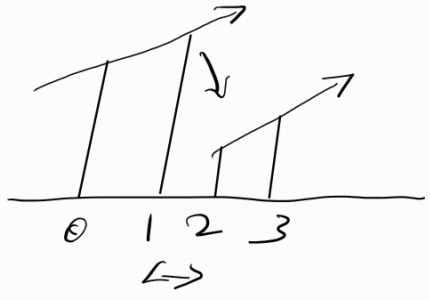
→ Python
→ pending
→ Pendre
} ICPC

if shelf was originally sorted in non-decreasing
↓ and then rotated to current order.
return 1
else
↓ return 0

$$A[i] = B[(i+x) \% A.length]$$

$A = [3, 4, 1, 2]$ (0 1 2 3)

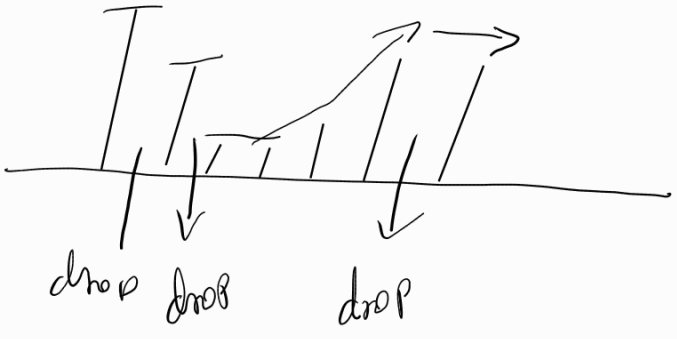
o/p = 1

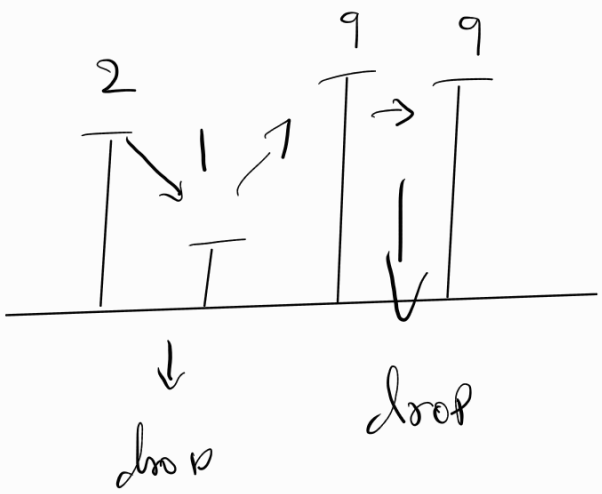


N = 4

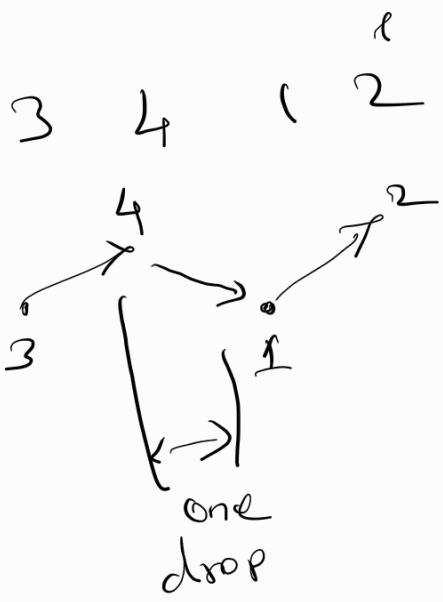
$= [9, 7, 1, 1, 5, 9, 9]$

$4 \overline{)2} \quad 1$
4
2





Note: we can not rearrange array into sorted array in increasing order if there is more than on drop.



count = 1

4 > 3

1 > 4 count++

2 > 1

617, 89, 946

89 > 617 count = 1

946 > 89

```
n = len(A)
drop_count = 0
x = 1
for i in range(n)
    if A[i] > A[(i+x) % n]:
        drop_count += 1
return 1 if drop_count <= 1 else 0
```

n = 3

i	x = 1	(i+x)	(i+x) % n
0		1	1 % 3 = 1
1		2	2 % 3 = 2
2		3	3 % 3 = 0
3		4	4 % 3 = 1
4		5	5 % 3 = 2
5		6	6 % 3 = 0

⋮

05. Rice :

1. Buy A kg of Rice. int A
2. There are N packets of rice. int[] PKT
3. Find min amount required to buy A kgs of rice

int A
int[] B

1. A = 12
B = [1, 1, 1, 1]
1 2 3 4
1 2 4 8

ans = 2
N = 4

1 2 4 8 16

0 0 0 1
0 0 0 1
0 0 0 1
0 0 0 1

→ Given string A,

→ given 2D array B of size Q x 2
↳ B[i][0] and B[i][1]

⇒ For every query, find the count of consonent and vowels in the substring A[B[i][0] ... B[i][1]]

⇒ Return array of substrings of length Q.

a e i o u

0 1 2 3 4 5 6 7 8

A = d e v e l o p e r

Q = [0, 3] d e v e — vowels: 2, consonants: 2

[2, 5] v e l o — vowels: 2, consonants: 2

[1, 3] e v e — vowels: 2, consonants: 1

0 1 2 3 4 5 6

A = e x a m p l e

Q = [1, 5] x a m p l — vowels: 1, consonants: 4

[1, 4] x a m p — vowels: 1, consonants: 3

[5, 6] l e — vowels: 1, consonants: 1

"vowels" if count of vowels ≥ to consonants
else, "consonant"

BF:

→ for every Query, going through the subarray (length = len(subar)) increase count if letter vowel

count = 0

→ update ans.

```
t = 'aeiou', ans = []
for q in Q:                          → Q
    i, j = q[0], q[1]
    vCount, cCount = 0, 0
    while (i <= j):  i += 1          → N
        if A[i] in t:
            vCount += 1
        else:
            cCount += 1
    if vCount >= cCount:
        ans.append("vowel")
    else:
        ans.append("Constant")
return ans
```

TC: O(Q × N)

SC: O(1)

optimized: Pre calculating vowels & consonants

	0	1	2	3	4	5	6
A =	e	x	a	m	p	l	e
vCount =	1	1	2	2	2	2	3
cCount =	0	1	1	2	3	4	4

Q = [1, 5] — vCount: 2-1 = 1; cCount: 4 — Consonant

[1, 4] — vCount: 2-1 = 1; cCount: 3 — Consonant

[5, 6] — vCount: 3-2 = 1; cCount: 4-3 = 1 — Vowel

Code:

```
n = len(A)   vowels = "aeiou"

# build vowels count and consonants count
vCount = [0] * n,
cCount = [0] * n
if A[i] in vowels:  vCount[0] = 1
else  cCount[0] = 1

for i in range(1, n)                    → O(n)
    if A[i] in vowels:
        vCount[i] = vCount[i-1] + 1
        cCount[i] = cCount[i-1]
    else:
        cCount[i] = cCount[i-1] + 1
        vCount[i] = vCount[i-1]

ans = [ ]
```

```
for q in Q:                                   → O(Q)
    i, j = q[0], q[1]
    if i == 0:
        if vCount[j] >= cCount[j]:
            ans.append("vowel")
        else:
            ans.append("consonant")
    else:
        countA = vCount[j] - vCount[i-1]
        countB = cCount[j] - cCount[i-1]
        if countA >= countB:
            ans.append("vowels")
        else:
            ans.append("consonants")
return ans
```

TC: O(N)

SC: O(N)

mega sale

Problem Description

A Mega Sale is going on in your favourite retail store for a day. You are excited and want to buy **A** chocolates from the store. There are **N** types of chocolates available inside the store, and their prices are given by an array **B**. You can buy only one chocolate of one type from the store.

As part of the store sale, there is a "**Buy 2 Get 1 Free**" offer, i.e. for every **two chocolate** you buy; you get **one chocolate** of any type for free.

Your task is to find the **minimum amount** of money you need to purchase **exactly A** chocolates from the store.

	1	2	3	4	5	
B =	5	6	1	7	8	A = 2

⇒ we have 5 Types of Chocalates.

⇒ i want to by 2 chocolates of same type. with minum cost.

The last min numbers are 1, 5

Total = 6

ex: 2

B = [1, 2, 3, 4, 5, 6] A = 4

buy index 0, 1, 3 and get 4 as free.

Total = 6

BF

1. Sort the array in accending (increasing) order
2.

```
A.sort()
```

while A > 0

if A >= 3 :

↓ buy 2 min chocolates, add 1

A - 3

elif A >= 2

↓ buy 2 min chocolate, A - 2

else:

↓ buy 1 min chocolate

A - 1

i = 0

while (A > 0):

if A >= 3 :

↓ ans += A[i]

ans += A[i+1]

A -= 3

i += 2

elif A >= 2 :

↓ ans += A[i]

ans += A[i+1]

A -= 2

i += 2

else:

↓ ans += A[i]

i += 1

A -= 1

ans = min amount to buy A chocolates

TC = (N log N)

SC : O(1)

optimized:

1. Sort the array.
2. if A = 4

K = A/3 # K = num of chocates comes free.

K = 1

num of chocolates to buy = A - K

Count => 4 - 1 = 3

ans = 0

for i in range(count)

↓ ans += B[i]

return ans

TC: O(A - K)

SC: O(1)

Increasing order words

Problem Description

Given an array of strings **A** of size **N**. Your task is to rearrange the words in **A** such that all words are rearranged in an increasing order of their lengths. If two words have the same length, arrange them in their original order.
Return the new array of strings sorted as mentioned above.

Problem Constraints

$1 <= N <= 10^5$
$1 <= |A[i]| <= 10$
A[i] consists of lowercase english letters

⇒ Given str[ ] A

⇒ re-arrange words in A, such that they are arranged in increasing order by their length.

⇒ if two words in same length arrang them in their original order.

BF:

1. using custom sorting.

2. sorted(A, key = cum_to_key(customfun))

Code:

```
def castfun (x, y):
    a = len(x)
    b = len(y)
    if a < b:
        return -1
    elif a > b
        return 1
    else # while a = b
        return 1
```

TC: $O(N \log N)$

SC: $O(1)$

---

next question:

Find All Pair

Problem Description

You are given arrays **A, B, and C,** each of length **N.**

Your task is to find the count of all pairs of integers **(x, y)** such that **A[x]** is equal to **B[C[y]]** where **1<= x, y <= N.**

Problem Constraints

$1 <= |A| = |B| = |C| <= 10^5$

$1 <= A[i], B[i], C[i] <= |A|$

⇒ Given int[] A, B, C

⇒ Find count of all pairs of integers (x, y), such

$A[x] = B[C[y]], \quad 1 \le x, y \le N$

A = [1, 2]

B = [2, 1]

C = [2, 2]

(x, y)

ans = 2 , (1, 1), (1, 2)

A[1] = B[C[1]]

1 — 2 — 1

```
for i in range(len(A))
    for j in range(i, len(C))
        if A[i] == B[C[j]]
            ans += 1
```

TC: $O(N^2)$

⇒ build k array with B[C[j]],

⇒ for a in A, check if A[i] ==

↳ iterate C and update freqmap with freqency of B(C(y))

↳ iterat A and chck condition,

↳ add the freqency to ans if it is in freqmap

```
freqmap = { }
for c in C:
    value = B[C-1]  # for 0 based indexing.
    freqmap[value] += 1

for i in range(len(A))
    if A[i] in freq map:
        ans += freqMap[A[i]]
return ans
```

Moving Coins to single display place

Problem Description

You have a collection of **N** antique coins displayed in a line, where the position of the i^{th} coin is given by **A[i]**. Your goal is to arrange all the coins in a single display case at the **same position**.

In one step, you can move the i^{th} coin from **position[i]** to:

- position[i] + 2 or position[i] - 2 with cost = 0 (you can slide the coins into grooves in the display case without any additional cost).
- position[i] + 1 or position[i] - 1 with cost = 1 (moving the coins manually to adjacent positions requires careful handling, incurring a small cost).

Determine the **minimum cost** to arrange all the antique coins at the **same position** in the display case.

→ n unique coins displayed in line.

→ Position of ith coin given in A[i] value.

→ goal is to arrange all coins in a single display position.

move, A[i] to A[i+2] (or) A[i-2] = 0 cost

" A[i] to A[i]+1 (or) A[i-1] = 1 cost

→ Determine the min cost.

ex: A = 2 2 2 3 3 ans = 2

(3 even, 2 odd)

A = 2, 4 # both or even ans = 0

1. Traveling in odd element with 0 cost
2. Traveling in even elements with 0 cost
3. Traveling in odd to even (or) wiseversa cost 1

So.

1. Count odd elements frequency
2. Count even ele frequency

even = 3
odd = 2

min To take

3. Take the minimum value and return.

Code!

```
evenCount, oddCount = 0, 0
for a in A:
    if a % 2 == 0:  # even
        evenCount += 1
    else:
        oddCount += 1
return min(evenCount, oddCount)
```

Math Land and Sum

Problem Description

In the quaint village of Mathland, nestled among lush green fields and babbling brooks, lived a group of mathematicians known for their insatiable curiosity and love for numbers. These mathematicians, led by their esteemed professor, Dr. Morgan, embarked on a quest to unravel the secrets of matrices and explore their infinite possibilities.

One sunny morning, as the gentle breeze whispered through the village, Dr. Morgan gathered his students under the shade of an ancient oak tree. He unveiled a large whiteboard adorned with a square grid representing a 2D matrix named **A**. The matrix had dimensions **N*N**, where N denoted the number of rows and columns containing positive integers.

Dr. Morgan's challenge to his students was as intriguing as it was complex. Dr Morgan has infinite number of students standing in a queue. He asked his students to come one by one and pick a rectangle which has never been picked by another student before and he will give the **total sum** of integers of that rectangle equivalent money to that student. The students will be able to participate as long as there is unique rectangle left in the 2D matrix.

Can you find out how much money in total Dr. Morgan will have to spend if the student can pick rectangle which are parallel to the x and y axis only?

Note: Answer may not fit in 32-bit integer range.

$A = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$ o/p = 16

[1] [1 1] [1] [1 1] ⇒ sum of all subarray.
[1] [1 1] [1] 16
[1]
[1]

1 2
2 4

$\frac{1 \times 2 \times 2 \times 4}{2 \quad 8} = 16$

BF:

1. going through each sub matrixes
2. finding the total sum.

TC: $O(N^3 \times N^3)$

BF2:

1. build prefixSum for matrix
2. iterate through 1 and multiply each elements.

TC: $O(N)$ SC: $O(N)$

Contribution Technique

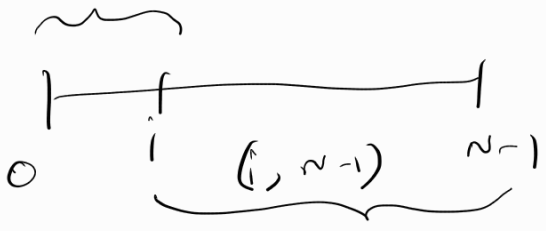
Contribution in 1D array

0	1	2	3	4	5
3	-2	4	-1	2	6

(num of subarray that ith ele contributes) x elem

ans +=

3 -2 4 -1 2 6 } 6



start	end indexes
(0, i)	(i, n-1)
i - 0 + 1	n - 1 - i + 1
(i + 1)	(n - i)

ans += ((i + 1) x (n - i)) x ele

# of subarr that ith ele can present

contribution technique for 2D arrays

[1 1]
[1 1] totalSum = 16

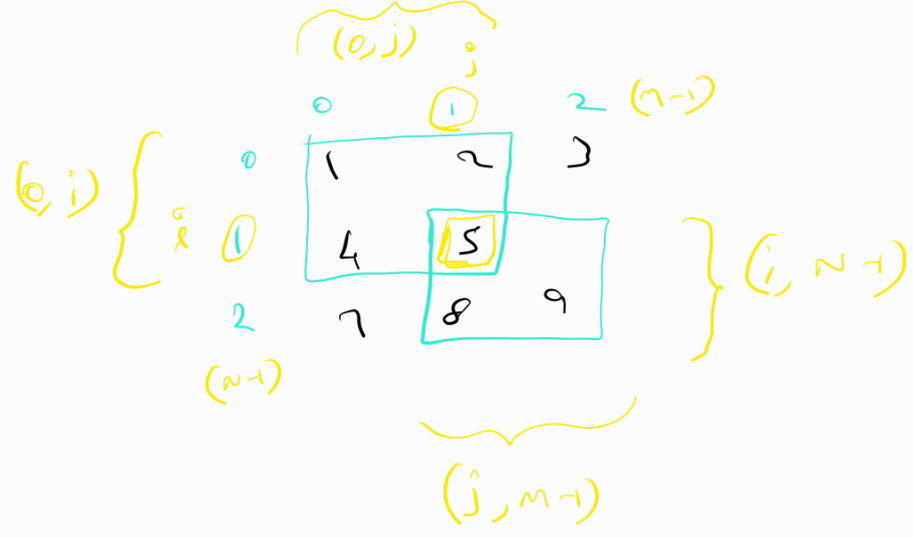
1. get num of sub-matrixes that ith element is part of.

x

element

=

totalSum +=



Top-Left

(0,i),(0,j)

i-0+1

Bottom-Right

(i, n-1), (j, m-1)

n-1-i+1

(i+1) x (j+1) x (n-i) x (m-j)

$N = 2$

i j	(i+1)	(j+1)	(N-i)	(N-j)	A[i]	
(0,0)	$(0+1)^1$	$(0+1)^1$	$(2-0)^2$	$(2-0)^2$	$\Rightarrow 4 \times 1 = 4$	8
(0,1)	$(0+1)^1$	$(1+1)^2$	$(2-0)^2$	$(2-1)^1$	$= 4 \times 1 = 4$	
(1,0)	$(1+1)^2$	$(0+1)^1$	$(2-1)^1$	$(2-0)^2$	$= 4 \times 1 = 4$	8
(1,1)	$(1+1)^2$	$(1+1)^2$	$(2-1)^1$	$(2-1)^1$	$= 4 \times 1 = 4$	

16 ✓

Wave Array

Problem Description

Given an array of integers **A**, sort the array into a wave-like array and return it.
In other words, arrange the elements into a sequence such that

```
a1 >= a2 <= a3 >= a4 <= a5.....
```

NOTE: If multiple answers are possible, return the lexicographically smallest one.

Problem Constraints

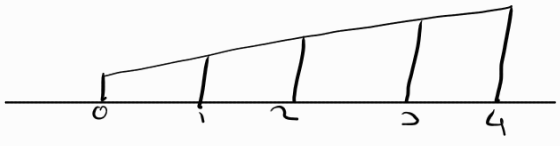
$1 <= len(A) <= 10^6$

$0 <= A[i] <= 10^6$

A = [2 1 3 4 5]

A.sort()

[1 2 3 4 5] n = 5



⇒ Increasing Order

```
i=0
while(i<N)
    if(i+1)<N:
        A[i], A[i+1] = A[i+1], A[i]
        i+=2
    else:
        i+=1
return A
```

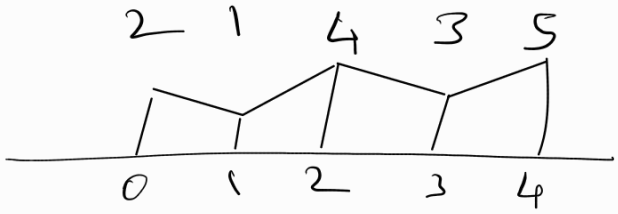
i = 0 1 2 3 4 n = 5

A = [5, 1, 3, 2, 4]

= [1 2 3 4 5]

[0] [1] [2] [3] [4]

2 1 4 3 5



⇒ wave link order

Decreasing Order Words

Problem Description

Given an array of strings **A** of size **N**. Your task is to rearrange the words in **A** such that all words are rearranged in an decreasing order of their lengths. If two words have the same length, arrange them in their original order.

Return the new array of strings sorted as mentioned above.

Problem Constraints

$1 <= N <= 10^5$

$1 <= |A[i]| <= 10$

A[i] consists of lowercase english letters

BF: using custom sorting method.

```
def customfunc(x,y)
    a,b = len(x), len(y)
    if a > b: return -1
    elif a < b: return 1
    else: return 0
from functools import cmp_to_key
return sorted(A, key = cmp_to_key(customfun))
```

TC: $(N \log N)$ SC: $O(1)$

Smallest value

Problem Description

You are given an integer **A**. You have to create an array **B** of length **A** containing integers greater than 0.

The array **B** is called valid if **B[i] != B[j]** for every **i** that divide **j** where **1 <= i < j <= A** (1-based indexing)

The score of the array is the maximum value among them. Your task is to create a valid array **B** of length **A** with the lowest possible score.

If there are multiple possible arrays, return the lexicographically smallest array **B**.

Problem Constraints

$1 <= A <= 10^5$

A = 4 ans B = [1, 2, 2, 3]

B[i] != B[j] and i // j $1 \le i \le j \le A$

A = 5 ans B = [1, 2, 2, 3, 2]

A = 6

1, 2, 2, 3, 2, 3

— Pending —