Agenda:

1. Introduction → What? Why? ⇒ colab
2. Min cost to remove
3. Min difference
4. Noble Element
5. Comparators. ⇒ colab

What is Sorting?

⇒ Ordering of elements

⇒ arranging in some specific pattern

⇒ Arrange the elements in orderly manner. based on a parameter

⇒ [ 1, 3, 7, 9, 15 ] Yes

is it sorted? ✓

⇒ [ 14, 11, 8, 5, 3, 1 ]

Yes.⇒

[ 0, 0, -1, 1, 2, 2 ]

is it sorted?

abs (val) => magnitude

[ 0, 0, 1, 1, 2, 2 ]

Square (num)

0, 0, 1, 1, 4, 4

[ 1, 7, 9, 6 12 ]

No. of factor

1 2 [1,7] {1,3,9} (1,2,3,6) {1,2,3,4,6,12}

## Applications of Sorting :

→ Searching in swift manner → Binary search

→ nearest data point

→ Analysis - Indexing

→ Top Trending analysis

→ To keep in a systematic -

## Sorting in Python !

```
l = [4, 1, 2, 3, 7]
```

→ ✓ `l.sort()` ⇒ It will change the original list

→ ✓ `l = sorted(l)` ↓ Doesn't effect the original list

returns a sorted copy of that list.

Desc `l.sort()` / `l[::-1]` `reverse = True` Key

TC  Sort =?

(*) O(N Log N) → merge sort, Heap sort

SC: O(1) → Quick sort O(N²)

Q Given an array. You have to remove every element from the array. Cost of removing an element from the array is sum of all elements present in the array. Find min cost to remove all elements.

Example: [4 1 2] ⇒ [1 2] ⇒ [2]

Removing 4 ⇒ 4+2+1 = 7

Removing 1 ⇒ 1+2 = 3

Removing 2 ⇒ 2 = 2

12

minimise this cost!

4 1 2 ⇒ 2 1 4 × 1 2 7 → 7 → 3 → 1 = (11)

11 > 12 ✓

4+1+2 = 7

1+4 = 5

4 4

16

Observation

(1) Remove Elements in Descending order:

$a_1$ $a_2$ $a_3$ $a_4$

1st removal: × – – – = $a_1+a_2+a_3+a_4$

2nd removal: × – – = $a_2+a_3+a_4$

3rd removal: × – = $a_3+a_4$

4th removal: $a_4$

Total cost:

$(a_1 + 2*a_2 + 3*a_3 + 4*a_4)$

$a_1 > a_2 > a_3 > a_4$ ⇒ optimal order for min sum

① Prefix Sum
SC: O(N)

```
def min_cost(A):
    A.sort(reverse=True)
    ans = 0
    for i in range(len(A)):
        ans += (i+1) * A[i]
    return ans
```

TC => O(N log N)

SC => O(1)

Q2 **Minimum Difference**

You are given an array of N elements
Find out the minimum value of

$$\min\left(\left|A[i] - A[j]\right|\right)$$ where $i, j$ are distinct

$A = [1, -5, 3, 5, -10, 4]$

ans = ?

**BF**

ans = INF.

```
for i → (0 → n)
    for j → (i+1, n)
        ans = min(ans, abs(A[i] - A[j]))
```

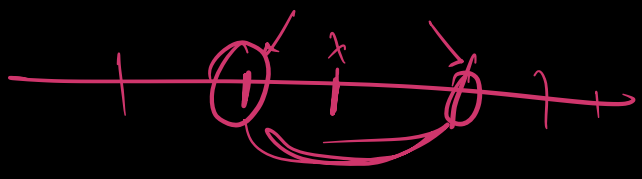
$O(N^2)$

Closest value to A[i]

⇒ Find closest value for every number

↳ SORTING

```
A.sort()
ans = INF
for
```



i = 0 → A[0] A[1]

i = 1 → A[0] A[1], A[1] A[2]

i = 2 → A[1] A[2], A[2] A[3]

i = 3 → A[2] A[3], A[3] A[4]

Every pair is being checked twice

A[i] A[i+1]

```
n = len(l)
l.sort()          # O(NlogN)
ans = INF         # math.inf
for i in range(n-1):      # O(N)
    ans = min(ans, A[i+1] - A[i])
return ans
```

A[i+1] > A[i]

TC : O(nlogn)

SC : O(1)

O(N) + O(NlogN)

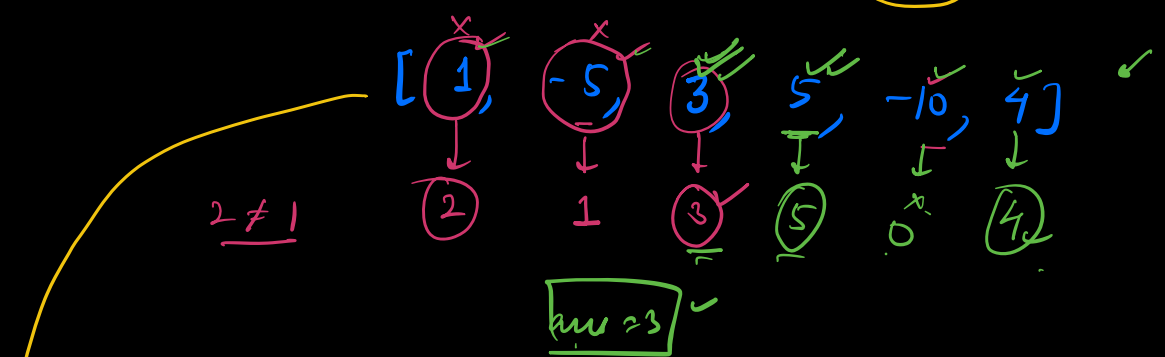
[ -7, -6, -5, -1 ]

-6 - (-7) = (+1)

## (3) Noble Elements.

Given an array of N distinct elements. Find count of noble elements

The no. of elements which are strictly less than A[i] == A[i]

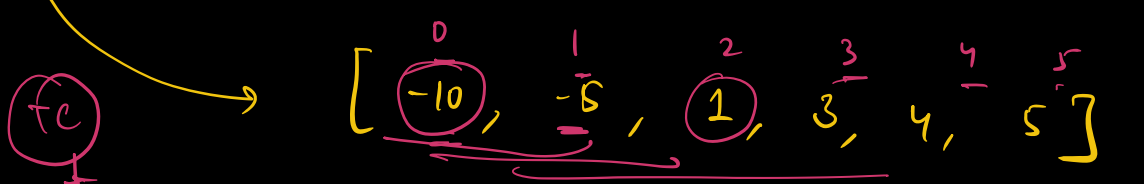



BF → TC → $O(N^2)$

go to every no.
→ count no.s less than A[i]
→ check A[i] == cnt

→ Sort in ASC order

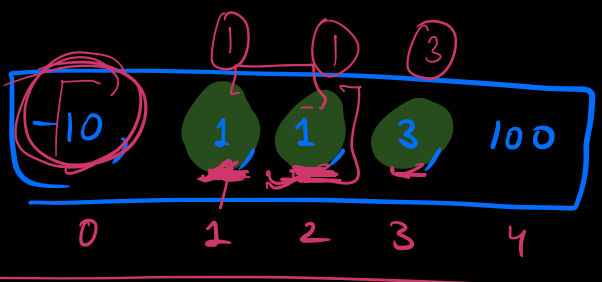
fc → $O(N \log N)$



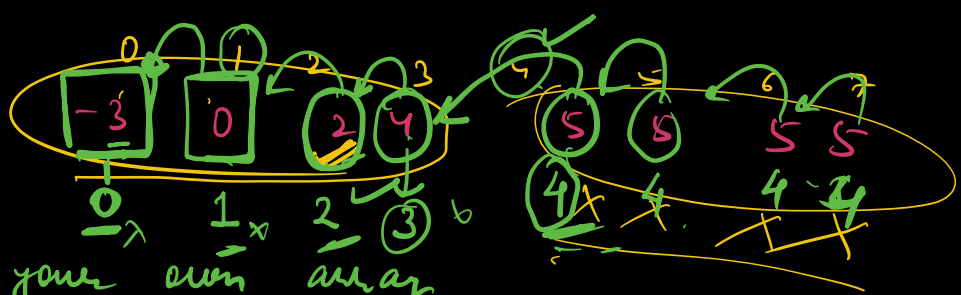

A[i] == i ⇒ cnt += 1

Follow up

Now the elements are not distinct




ans = 2 ✗




Create your own array

Storing count of elements smaller than current element.

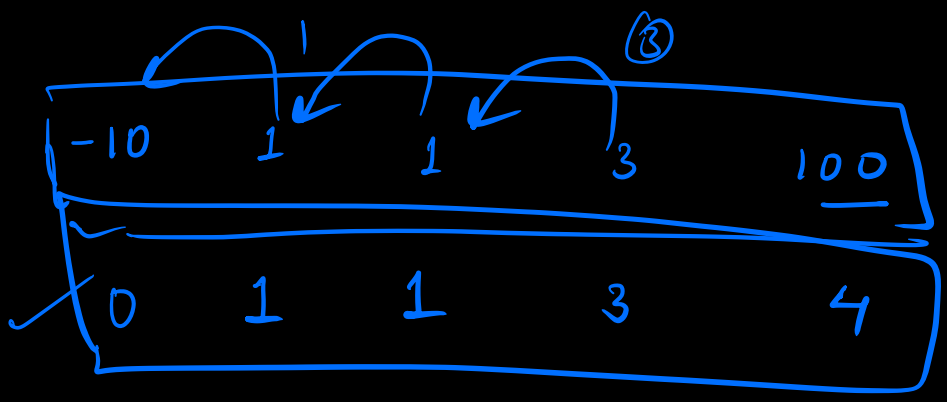
```
l = [0] * N
for i in range(1, N):
    if arr[i] == arr[i-1]:
        l[i] = l[i-1]
    else
        l[i] = i
```

repeated element

→ O(1)

for

l[i] == A[i]




- 3 0 2 4 5 5 5 5

## Comparators