Sorting

→ Understand, what is sorting
→ Few problems on sorting
→ 1 Sorting algo
→ Comparator

Sorting : Arranging data in increasing or decreasing order. based on a parameter

Eg: 2 3 7 11 13 : inc.

Eg: 15 9 7 3 2 0 : dec.

Eg:

	1	2	3	7	4	9	6
# factors:	1	2	2	2	3	3	4

In-built Library in Language

↳ Sort() function (in every language)

↳ How? Logic? { In adv. module }

TC : $O(N \log N)$

↳ No. of elements to sort

Q1: Elements removal

Given N elements, at every step remove an a[] element.

Cost to remove an element = sum of all elements in array present at that instance.

Find min. cost to remove all elements.

Eg: a[] : { 2 1 4 }

	Total Cost
remove 1 :	2+1+4 = 7
remove 4 :	2+4 = 6
remove 2 :	2 = 2
Total Cost:	15 ✗

remove 4 :	2+1+4 = 7
remove 2 :	2+1 = 3
remove 1 :	1
Total Cost :	11

Eg: a[] : { 3 6 2 4 }

	Cost
remove 6 :	15 (3+6+2+4)
remove 4 :	3+2+4 = 9
remove 3 :	3+2 = 5
remove 2 :	2
Total Cost :	31

$a = \{ 6 \quad -3 \quad 4 \}$

remove 6 : $6 - 3 + 4 = 7$

remove 4 : $-3 + 4 = 1$

remove -3 : -3

5

Obs: We are deleting / removing elements in desc. order.

WHY?

$a[4] = \{ a \quad b \quad c \quad d \}$

remove a : $a + b + c + d$

remove b : $b + c + d$

remove c : $c + d$

remove d : d

Total Cost: Reduce the cost

$(a + 2b + 3c + 4d)$ (0, 1, 2, 3)

a → 1st max

$2b$ → 2nd max

$3c$ → 3rd max

$4d$ → 4th max

```
int minCost(int a[], int N) {
    Sort(arr, desc) → Sorts the array in desc. order
        ↳ check syntax in your prof. lang.

    int c = 0
    for(i = 0; i < n; i++) {
        //How many times will a[i] occur = (i+1) times
        c = c + a[i] * (i+1)
    }
    return c
}
```

TC: $O(N \log N + N) = O(N \log N)$

SC: $O(1)$

Q2: Noble Integers { data is distinct }
↳ can't repeat

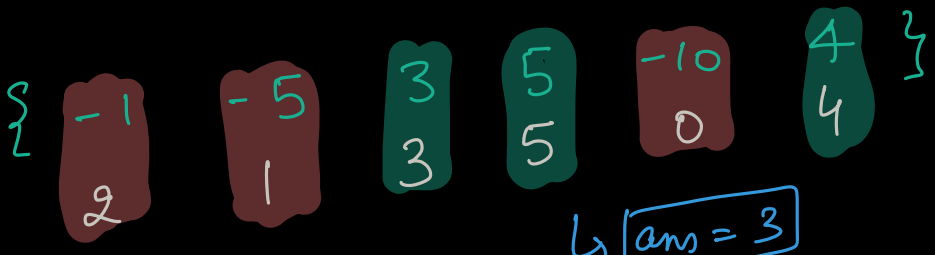
Given a[N]. Calc the no. of noble integers.

Noble integer

An element a[i] is said to be noble if:

No. of elements < a_i = a_i itself

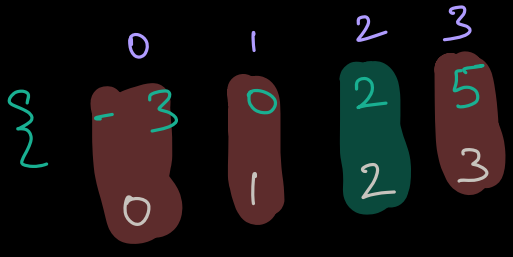
Eg:



less:

↳ ans = 3

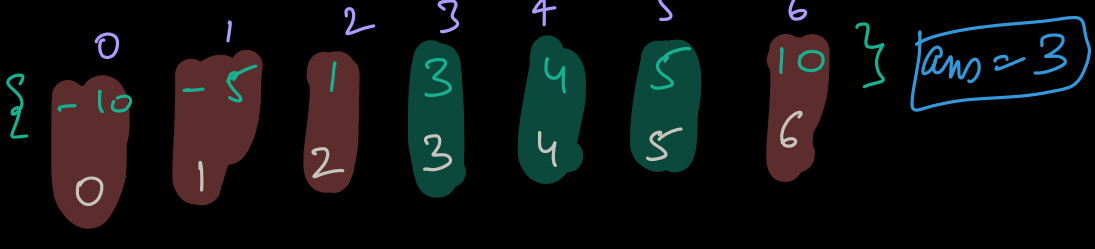
Eg:



less:

↳ ans = 1

Eg:



less:

ans = 3

Brute force: for every element a_i, iterate on $a[]$ & get count of elements $< a_i$ & compare with a_i

```
ans = 0
for(i=0; i<n; i++) {
    // a[i], need ele < a[i]
    c = 0
    for(j=0; j<n; j++) {
        if(a[j] < a[i]) {c++}
    }
    if(a[i] == c) {ans++}
}
```

Taking time ∴ to find count of smaller elements, SORT

TC: $O(N^2)$
SC: $O(1)$

// Sort:

$[0 \dots i-1]$ i^{th}

↳ all elements $< a_i$

# of elements $< a_i$ = $[0\ i-1] \Rightarrow i$ elements.

```
int noble(int a[], int N) {
    Sort(arr, inc) // Sort in inc. order
    c = 0
    for(i=0; i<n; i++) {
        if(a[i] == i) {c++}
    }
    return c
}
```

TC: $O(N \log N + N)$
$\therefore = O(N \log N)$
SC: $O(1)$

Q3: Same Qn, with repeating data

↳ not distinct anymore

BF app ✓

sort idea ✗

Eg: { 0 2 2 3 3 6 }
     { 0 1 1 3 3 5 }

less: 0 1 2 3 4 5

↳ Ans = 3

Eg: { -10 1 1 1 4 4 4 7 10 }
     { 0 1 1 1 4 4 4 7 8 }

less: 0 1 2 3 4 5 6 7 8

Ans = 7

Eg: { -3 0 2 2 5 5 5 5 8 8 10 10 14 }
     { 0 1 2 2 4 4 4 4 8 8 10 10 12 }

less: 0 1 2 3 4 5 6 7 8 9 10 11 12

Ans = 6

Obs1: if an element comes for very first time, no. of elements < that = idx itself

if (a[i] != a[i-1])

Obs2: if data repeats, then count of elements < that WON'T CHANGE

if (a[i] == a[i-1])

```
int nobleRepeat(int a[], int N) {
    int less = 0 // To track the no. of elements < ai

    Sort(arr, inc)
    if(a[0] == 0) { c++ } → Edge case

    for(i=1; i<n; i++) {
        // for ai, get elements < ai
        if(a[i] != a[i-1]) { // Edge case:
            less = i          i=0  a[0] != a[-1]
        }
        else {
            // Do nothing
        }
        if(less == a[i]) { c++ }
    }
    return c
}
```

TC: $O(N \log N + N) = O(N \log N)$

SC: $O(1)$

Break :→ till 8.40am

① 1 Sorting algo

a[] = { 3 8 6 2 4 3
6 8 8 8
2 4

iter 1: 3 6 2 4 8
2 6 6
4

iter 2: 3 2 4 6 8
2 3

iter 3: 2 3 4 6 8

Obs: 1 After 1st iter → Last 1 element is in correct pos

2 After 2nd iter → Last 2 elements are in correct pos

3 After 3rd iter → Last 3 elements are in correct pos

⋮

N-1 After N-1 iter → Last N-1 elements are in correct pos
↳ all N are correct

→ In every iteration: (N-1 iter)

Iterate from first to last, compare adjacent elements

If not in correct order
↳ we swap

Bubble Sort (int a[], int N) {

```
for(i=0; i<n-1; i++) { // N-1 times
    for(j=0; j<n-1-i; j++) {
        if(a[j] > a[j+1]) { // Edge case: j = n-1, a[n-1] a[n]
            swap(a_j, a_j+1)
        }
    }
}
```

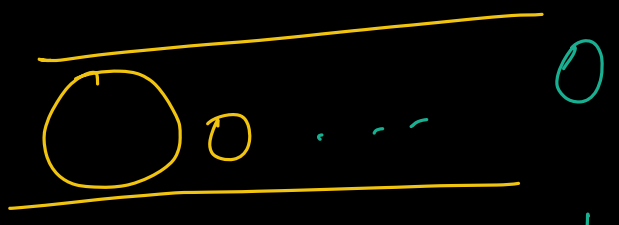
→ dec

TC: $O(N^2)$

SC: $O(1)$

Algo name: Bubble Sort

if (condn) has the power to decide order of elements.



```
void BubbleSort(int a[], int N) {
    for(i=0; i<n-1; i++) {
        for(j=0; j<n-1; j++) {
            if(comp(a_j, a_i)) {
                swap(a_j, a_i)
            }
        }
    }
}
```

```
bool comp(int a, int b) {
// Overriding Comparator
}
```

We need to know:

1. a before b
2. b before a

Comparator in Java / JS → Check Syntax

```
Comparator customComparator(Integer a, Integer b) {
    If you want a to come before b, return -1
    If you want a & b = same, return 0
    If you want b to come before a, return 1
}

Sort(arr, customComparator)
```

Comparator in Python

```
def compare(a, b):
    If a should be before b, return -1
    If a & b same, return 0
    If b before a, return 1

arr.sort(key = cmp_to_key(compare))
```

Comparator in C++

```
bool Compare(int a, int b) {
    If a should be before b, return true
    Else, return false
}
```

Qn: Given a[N]. Sort them in inc. order of their no. of factors.

If 2 elements have same factors, element with lesser value should come first.

SC: O(1)

a[]: { 9 3 4 8 16 37 6 13 15 }

factors: 3 2 3 4 5 2 4 2 4

ans = { 3, 13, 37, 4, 9, 6, 8, 15, 16 }

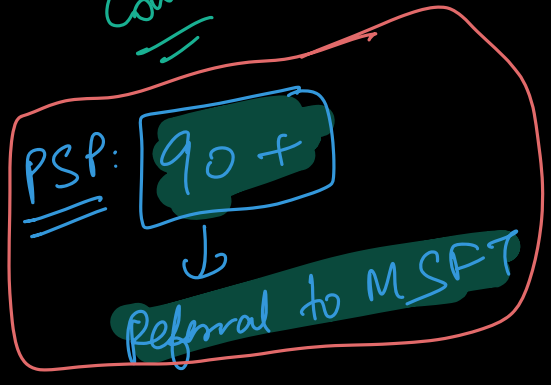
idea: given arr[]

Sort(arr, comp)

return arr

TC: → adv module

Course: Merch




```
int comp(int a, int b) {
    int fa = factors(a)   ] first class
    int fb = factors(b)
    // When should a come before b?
    if(fa < fb) {
        return -1 (a comes before b)
    }
    else if(fa == fb && a < b)
        return -1
    }
    else { // fa > fb
        b comes first
        return 1
    }
}
```