Today's Agenda,

- → Sorting Basics
- → Min Cost to remove elements
- → Noble Integer
  - ↓ Distinct
  - ↘ Duplicate
- → Comparator.

Sorting :- Arrangement of data in particular order on the basis of some parameter.

2, 3, 9, 12, 17, 19

Sorted in asc order (magnitude)

19	6	5	2	-1	-19

Sorted in desc. order (magnitude)

1	13	9	6	12
1	2	3	4	6

count of factors

Sorted in asc. order of count of factors.

Why Sorting ? Searching becomes easier.

How ~~to~~ Sort advanced.

inbuilt sort functions

sort ( ); T.C → O(nlogn)

(n^2) → nlogn. → Always try to think about sorting.

(n)

Q array of N integers :- you have to delete all the elements from the array.

you have to pay some cost for every deletion.

> Cost = Sum of all elements left in the array.

find the min cost.

A : [~~2~~ 1 ~~4~~] → [4, 2, 1]

delete 1 → 2 + 1 + 4 ⇒ 7
delete 2 → 2 + 4 ⇒ 6
delete 4 → 4 ⇒ 4
17

delete 4 → 4 + 2 + 1 ⇒ 7
delete 2 → 2 + 1 ⇒ 3
delete 1 → 1 ⇒ 1
11

A : [~~3~~ ~~6~~ ~~2~~ ~~4~~] → [6 4 3 2]

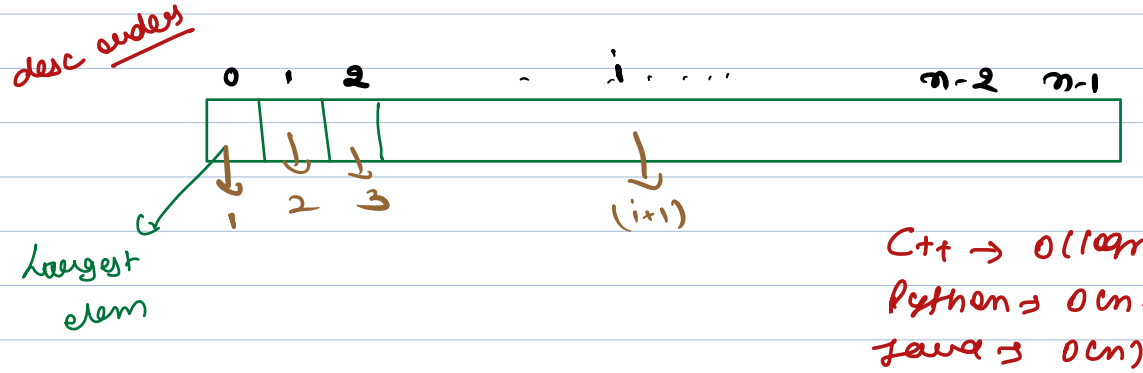
delete 6 = 3 + 6 + 2 + 4 ⇒ 15
delete 4 = 4 + 2 + 3 ⇒ 9
delete 3 = 3 + 2 ⇒ 5
delete 2 = 2 ⇒ 2
31

[a b c d]

Remove	a	a + b + c + d
Remove	b	b + c + d
Remove	c	c + d
Remove	d	d
		a + 2b + 3c + 4d

Start removing from largest end.

Sort (arr) → $n \log_2 n$



C++ → O(log n)
Python → O(n)
Java → O(n)
} Space

1) Sort the data in desc order

```
ans = 0;
for (i=0; i<n; i++) {
    ans += (i+1) * arr[i];
```

T.C → O(n log n)
S.C → O(n).

Java + Python → Tim Sort
C++ → Quicksort + Insertion Sort.

Ques) Find the count of Noble Integers.
Array of size n.

→ Elements are distinct.

A[i] is noble if

Count of smaller elements than A[i] = A[i]

0	1	2	3	4	5
1	-5	3	5	-10	4
↓	↓	↓	↓	↓	↓
2	1	3	5	0	4

Ans = 3.

Ques Can Noble Integer be -ve ?

Yes NO

0	1	2	3
-3	0	2	5
↓	↓	↓	↓
0	1	2	3

Ans = 1

Basic idea :-

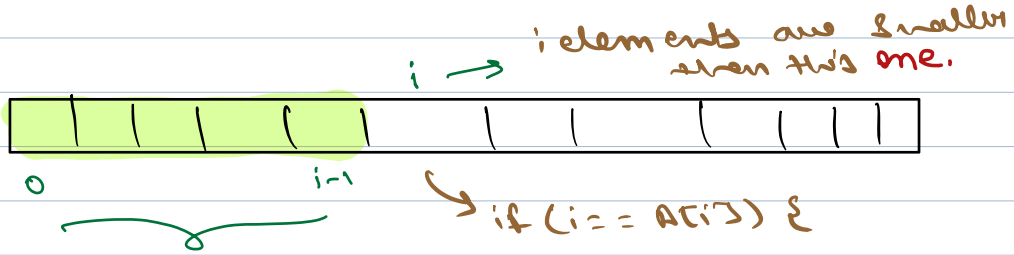
for every element count smaller elements.

```
ans = 0
for (i = 0 ⟶ n) {
    cnt = 0;
    for (j = 0; j < n; j++) {
        if (arr[j] < arr[i])
            cnt++;
    }
    if (cnt == arr[i]) ans++
}
```

T.C → O(n²)

S.C → O(1).

Idea 2 :- Sort the data (asc);




```
if (i == A[i]) {
    ans++
}
```

(i-1) - (0) + 1 => i

arr[] = 0 1 2 3 / 0 1 2 3

no. of elements smaller than A[i] == A[i]

i

// sort the data.

```
for (i=0; i<n; i++) {
    if (ans[i] == i)
        ans++;
```

count of smaller elements.

T.C → O(nlogn).

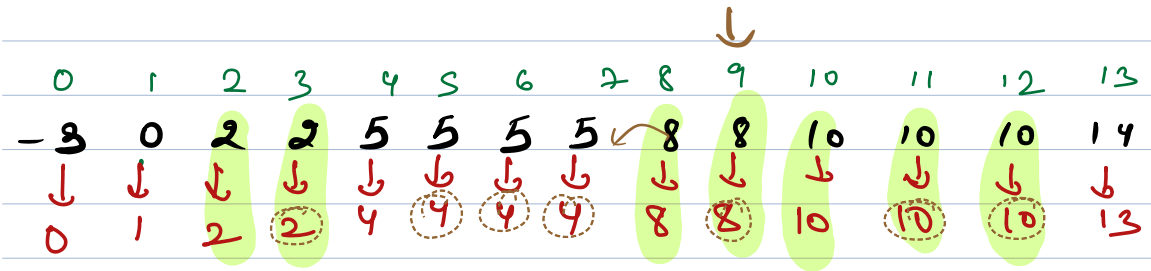
Break 10:01 - 10:11 pm.

what if there are duplicates?

	0	1	2	3	4
	-10	1	1	3	100
	↓	↓	↓	↓	↓
	0	1	1	3	4

Ans = 3.

i →	0	1	2	3	4	5	6	7	8
ans[i]	-10	1	1	2	4	4	4	8	10
	↓	↓	↓	↓	↓	↓	↓	↓	↓
no. of elements smaller.	0	1	1	3	4	4	4	7	8



① Sort the data

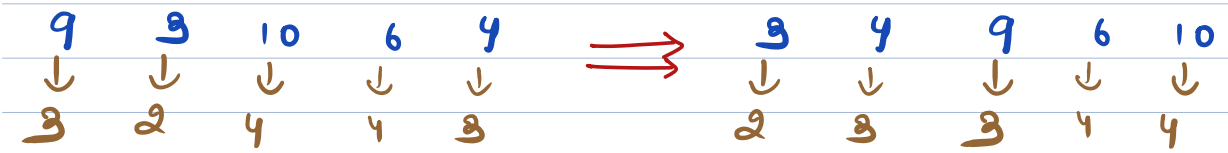
```
cnt = 0;  → Count of smaller elements.
if (arr[0] == cnt) {
    ans++;
}
for (i = 1; i < n; i++) {
    if (arr[i] != arr[i-1]) {
        cnt = i;
    }
    if (cnt == arr[i]) {
        ans++
    }
}
```

T.C → $O(n \log_2 n)$

S.C → $O(n)$.

(11) Sort the data in asc order of count of factors,

if count of factors is equal, then sort on the basis of mag.



Sort ( ) → By default, asc order of magnitude.

↓

comparing two elements.

Comparator → function.

Sort (____ , ____ , ____)

↑ start, ↑ end, ↳ comparator.

→ T.C → $(n \log n) * \sqrt{n}$

```
bool comp (int x, int y) {
```

if first argument should come first in sorted data

return True

else return False

```
int cnt x = count_factors(x); (√n)
int cnt y = count_factors(y); (√n)

if (cnt x < cnt y) {
    return True;
}
else if (cnt x > cnt y) {
    return false;
}
else {
    if (x <= y) {
        return True
    } else {
        return false;
    }
}
}
```

how to write comparator for — ?

(*) Sort in desc order of mag.

```
bool comp (int x, int y) {
    if (x >= y) {
        return True;
    }
    else {
        return False
    }
}
```

T.C → (n log n) * (complexity of comparator)

Arrays.sort ( )

↓

merge sort

↓

— —

Selection sort

5, 10, 1, 9, 3 →

1, 3, 9

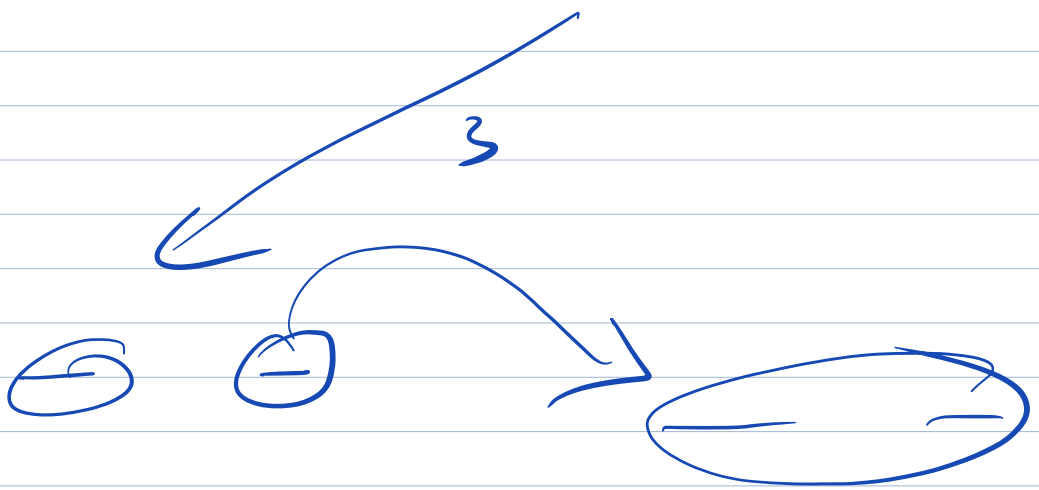
n^2

5, 10, 1, 9, 3

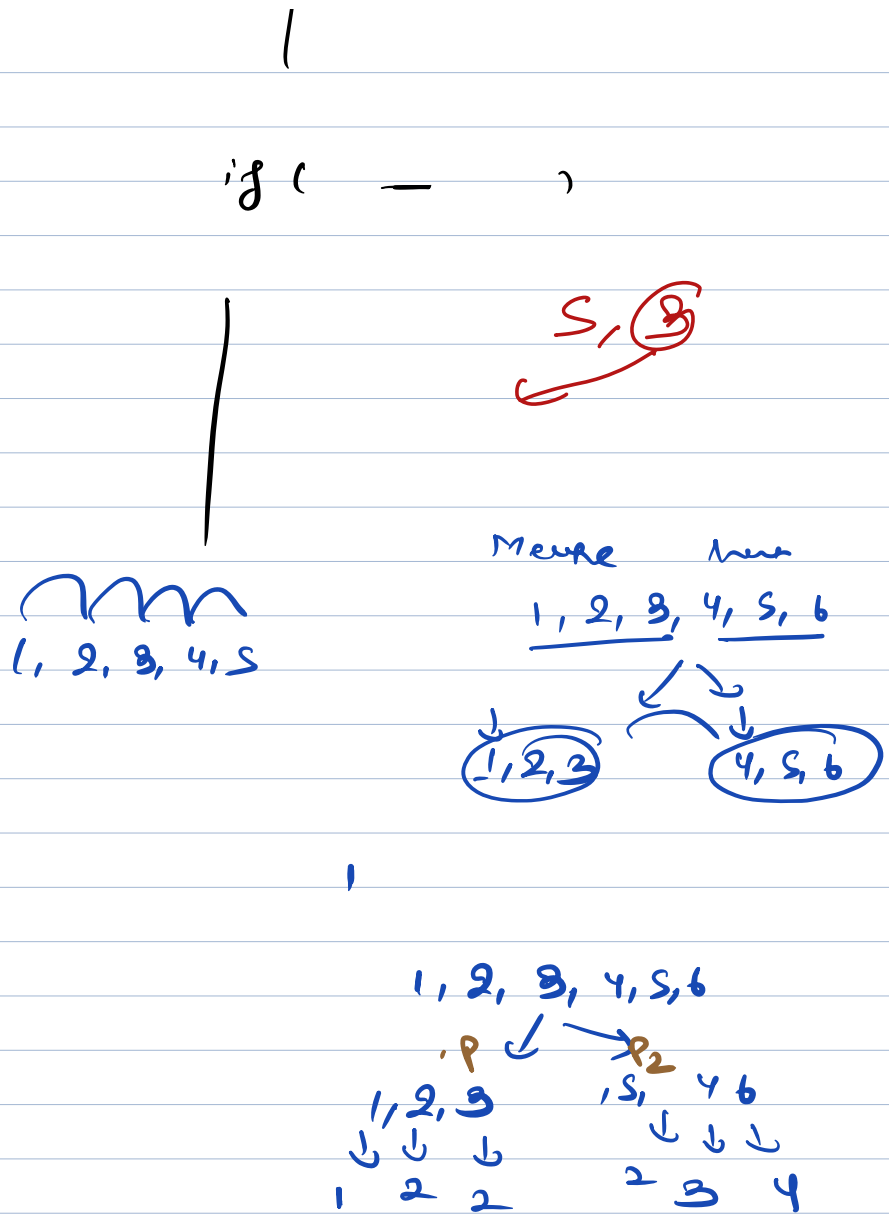
9,

if (compare current arr)

3



20, 30, 40, 50, 60, 70



1) WhatsApp group

↳ TD

↳ me

Previous Interview Experiences.

↳ revise.

$^{n}C_{r} = {}^{n-1}C_{r} + {}^{n-1}C_{r-1}$

$^{5}C_{3}$

$^{4}C_{2}$ $\quad$ $^{4}C_{3}$

$^{3}C_{2}$ $\quad$ $^{3}C_{3}$ $\rightarrow 1$

$r=0 \rightarrow 1$

n

r \ n	0	1	2	3	4	5
0	1	1	1	1	1	1
1		1	2	3	4	5
2			1	3	6	
3				1		
4					1	
5						1

$^{2}C_{1} \rightarrow {}^{1}C_{1} + {}^{1}C_{0}$