Today's Agenda :-

1) Intro to Sorting
2) Min cost to empty array | remove all elements
3) Noble Integers - 1
4) Noble Integers - 2
5) Intro to Comparators
6) Sort array based on their count of factors

Sorting :- arrangement of data in inc/dec based on same parameter

ar[] : { 3, 8, 9, 14, 19 } } Sorted in asc order based on magnitude

ar[] : { 19, 14, 9, 8, 3 } } Sorted in des order based on magnitude

ar[] : { 1, 13, 9, 6, 12 } → Sorted in inc order based on # of factors

1 2 3 4 5

Q1) Min cost to empty array | Remove all elements.

Given N array elements, at every step, remove an element. The cost to remove element is equal to sum of all elements present in the array at that point. Find min cost to remove all elements.

Eg arr[]: {2, 1, 4}

→ {4, 2, 1}

4*1 + 2*2 + 1*3

4+4+3 ⇒ 11

		cost
Remove 2	→ {2, 1, 4}	+7
Remove 1	→ {1, 4}	+5
Remove 4	→ {4}	+4
	{}	16

		cost
Remove 4	→ {2, 1, 4}	+7
Remove 2	→ {2, 1}	+3
Remove 1	→ {1}	+1
	ans =	11

Eg arr[]: {4, 6, 1}

→ {6, 4, 1}

6 + 2*4 + 3*1

6+8+3 ⇒ 17

		cost
Remove 6	{4, 6, 1}	+11
Remove 4	{4, 1}	+5
Remove 1	{1}	+1
		17

Eg arr[]: {3, 5, 1, -3}

		Cost
Remove 5	{3, 5, 1, -3}	6
Remove 3	{3, 1, -3}	1
Remove 1	{1, -3}	-2
Remove -3	{-3}	-3
	ans =	2

Approach :- Start from largest to smallest

$[a, b, c, d]$

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

$a > b > c > d$

$2*1 + 2*2 + 1*3$

9

$\{2, 2, 1\}$

Remove 2	$\{2, 2, 1\}$	5
Remove 2	$\{2, 1\}$	3
Remove 1	$\{1\}$	1
		9

i	#
0	1
1	2
2	3
i	(i+1)

Pseudo Code :-

```
ans = 0
Sort(arr) in desc order // inbuilt function }
for(int i=0; i<N; i++){
    ans = ans + arr[i]*(i+1)
}
return ans
```

TC: NlogN

N

TC : O(NlogN)

SC : O(1)

Q2) Noble Integer (Distinct elements) ✓

Given N array elements. Calculate no of Noble integers present.

ar[i] is said to be noble if :-

{ No of elements < ar[i] = ar[i] }

Eg

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

} ans = 3

count less than ar[i]

Eg

0	1	2	3
{ -3,	0,	2,	5 }
0	1	2	3

} ans = 1

count less than ar[i]

Eg ar[] :

0	1	2	3	4	5	6
{ -10,	-5,	1,	3,	4,	5,	10 }
↓	↓	↓	↓	↓	↓	↓
0	1	2	3	4	5	6

count < ar[i]

 # elements < ar[i] = i in a sorted array

Approach 1:- (Brute force) :- For every element, get no of ele less then ar[i]

```
int ans = 0
for(int i = 0; i < N; i++){
    int less = 0
    for(int j = 0; j < N; j++){
        if(ar[j] < ar[i]){
            less++
        }
    }
    if(less == ar[i]){
        ans++
    }
}
```

TC : $O(N^2)$

SC : $O(1)$

< ar[i]

ith ar[i]

0 i-1

a b → b - a + 1

↳ [0 i-1] → (i-1-0+1) = i

Approach 2 :

1) Sort the array in asc order

```
int ans = 0
Sort(arr) in asc order } N log N
for(int i = 0; i < N; i++){ } N
    if(ar[i] == i){
        ans++
    }
}
return ans
```

TC : $O(N \log N)$

Q3) Noble Integers - 2 (Elements can repeat)

Eg arr[] - {-10, 1, 1, 3, 100}

	0	1	2	3	4
	-10	1	1	3	100
# cnt	0	1	1	3	4
< arr[i]					

} ans = 3

Eg {-10, 1, 1, 2, 4, 4, 4, 8, 10}

	0	1	2	3	4	5	6	7	8
	-10	1	1	2	4	4	4	8	10
# cnt	0	1	1	3	4	4	4	7	8
< arr[i]									

} ans = 5 ✓

Eg {-3, 0, 2, 2, 5, 5, 5, 5, 8, 8, 10, 10, 10, 14}

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
	-3	0	2	2	5	5	5	5	8	8	10	10	10	14
# cnt	0	1	2	2	4	4	4	4	8	8	10	10	10	13
< arr[i]														

ans = 7

Observations :- (on sorted array)

1) If element is same as previous, cnt of elements < arr[i] won't change

5 5
cnt cnt

2) If element comes for first time, (diff from prev), cnt of ele < arr[i] = i

// Pseudo Code

```
Sort (arr) on asc order } NlogN
int ans = 0
int cnt = 0
if (arr[0] == 0) ans ++
for(int i = 1; i < N; i++) {
    if (arr[i] != arr[i-1]) {
        cnt = i
    }
    if (cnt == arr[i]) {
        ans++
    }
}
return ans
```

} N

TC : O(N log N)

ans = 1 2 3 4 5

cnt = 3 7

	0	1	2	3	4	5	6	7	8
{	-10,	1,	1,	2,	4,	4,	4,	8,	10 }
	↓	↓	↓	↓	↓	↓	↓	↓	↓
cnt	1	1	3	4	4	4	7	8	

Comparators (theory) } Break for 6 Min

1 Problem

Intro to Comparators :-

Comparator is a function that compares 2 values and returns a result indicating whether the values are equal, less than, or greater than the other.

The function is used in sorting algorithms to compare elements in a data structure & arrange them according to some parameter.

sort(a, a+N, comp) → (a, b)

```
int comp(a, b){
    return a-b
}
```

a < b → ⊖ ve

b < a → ⊕ ve

Comparator - A function that takes 2 arguments :-

For languages, Java, Python, JS, C#, Ruby, the logic is :-

(i) if first argument should come before second, ⊖ve value is returned

(ii) if second argument should come before first, ⊕ve value is returned.

(iii) if both are same, 0 is returned.

For C++,

```
bool comp(int a, int b){
    if (a < b){
        return true
    }
    return false
}
```

(i) if first argument should come before second, true is returned

(ii) false is returned otherwise

15 10

Descending

$a > b$

↳ return ⊖

$b > a$

↳ return ⊕

```
int comp( int a, int b) {
    if (a > b) return -1
    else return 1
}
sort ( ar, ar+N, comp)
```

Collection . sort (ar, ar+N, { });

Q4) Given N array elements. Sort the data in ascending order of count of factors. If count of factors are equal, then sort based on their magnitude.

Eg

	0	1	2	3	4
arr[] :	9	3	10	6	4
#factors	3	2	4	4	3

sort ⟶ {3, 4, 9, 6, 10} ans

Ex1

	a	b
	25	16
	↓	↓
	3	5

→ In final order, 25 will come before 16.

Ex2

10	9
↓	↓
4	3

9 will come before 10

Ex

49		25
↓		↓
3	=	3

25 will come before 49

```
int factors(int N){
    //returns no of factors of N
}
```
} O(Sqrt(N))

```
int comp(int a, int b){
    int f1 = factors(a)
    int f2 = factors(b)
    if(f1 < f2){
        return -1
    }
    else if(f2 < f1){
        return 1
    }
    else{
        return a-b;
    }
}
```
- if a < b, ⊖ve
- if a > b, ⊕ve (b < a)

```
if(f1 == f2){
    return a-b
}
else{
    return f1-f2
}
```

This is not exact syntax

→ { N log N * √N , N √N log N }

```
Collections.sort(A, comp);
```
} Advanced DSA : Sorting

A , A+N

```
sort(A, A+N, comp);
```
(C++)

C++

```cpp
int factors(int n)
{
    int count = 0;
    int sq = sqrt(n);

    // if the number is a perfect square
    if (sq * sq == n)
        count++;

    // count all other factors
    for (int i=1; i<sqrt(n); i++)
    {
        // if i is a factor then n/i will be
        // another factor. So increment by 2
        if (n % i == 0)
            count += 2;
    }
    return count;
}

bool compare(int val1, int val2)
{
    int cnt_x = countfactors(x);
    int cnt_y = countfactors(y);
    if(factors(val1) == factors(val2))
    {
        if(val1<val2)
        {
            return true;
        }
        return false;
    }
    else if(factors(val1)<factors(val2))
    {
        return true;
    }
    return false;
}
```

Python

```python
def comapre(v1, v2):
    if(factors(v1) == factors(v2)):
        if(v1<v2):
            return -1;
        if(v2<v1):
            return 1;
        else
            return 0;
    elif (factors(v1)<factors(v2)):
        return -1;
    else
        return 1;
```

very very simple

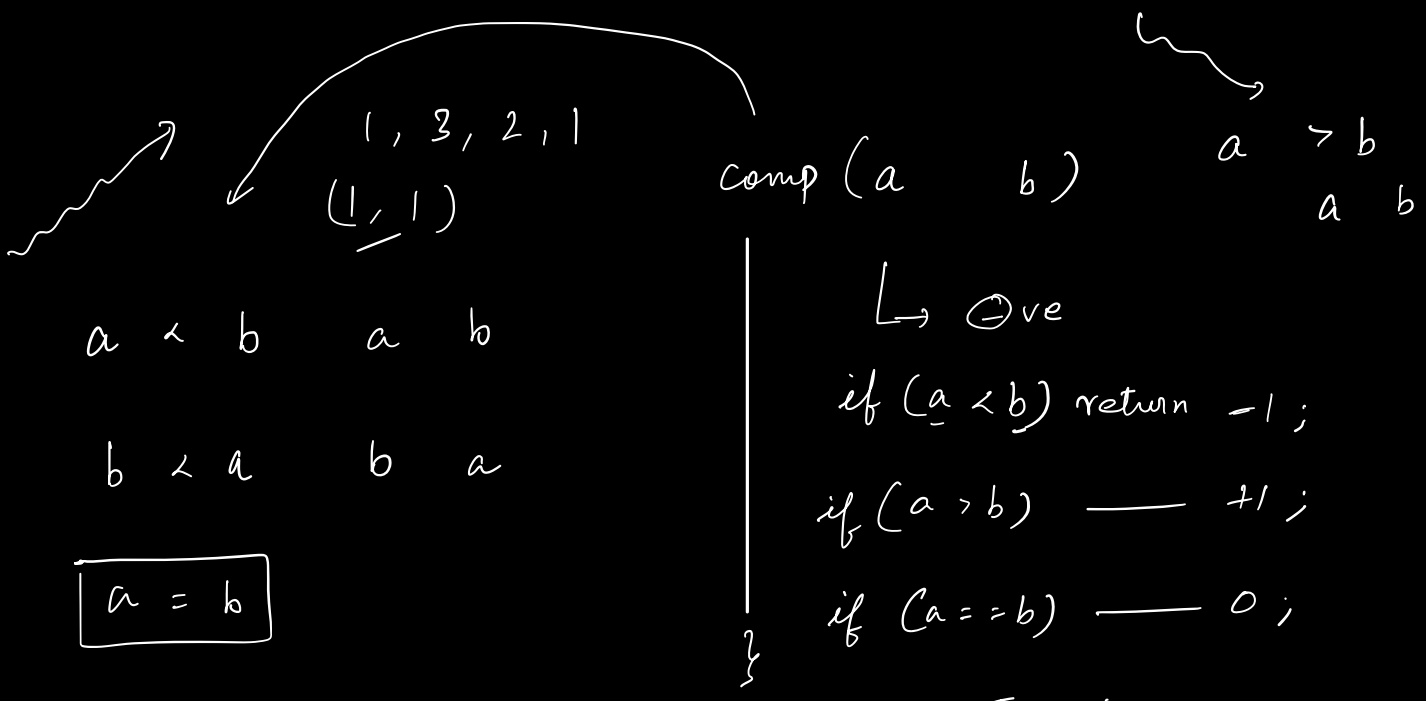
Java

```java
Collections.sort(A, new Comparator<Integer>()){
    @Override
    public int comp(Integer v1, Integer v2){
        if(factors(v1) == factors(v2)){
            if(v1<v2) return -1;
            else if(v2<v1) return 1;
            return 0;
        }
        else if(factors(v1)<factors(v2)){
            return -1;
        }
        return 1;
    }
}
```

1) Built in Sort function

2) Comparator & pass it along your Sort fun

↳ Adv DSA


1, 3, 2, 1
(1, 1)
comp (a b)
a > b
a b
a < b a b
b < a b a
a = b
}
↳ ⊖ve
if (a < b) return -1;
if (a > b) —— +1;
if (a == b) —— 0;
Java /
a b
if (factors(a) < factors(b))
return ⊖
}
a b
a > b
comp (a, b) {
if (a > b) — ⊖
}