INTRODUCTION TO SORTING

1. Intro - Sorting
2. Problem 1: Min Cost to Remove All Elements
3. Problem 2: Minimum Difference
4. Problem 3: Noble Integer K
5. Comparator - what it is and how to use, where to use

1. Slack
2. deepak.kasera_1@scaler.com
3. 7015608331

⇒ SORTING :- Arranging the 'data' in Increasing / Decreasing order based on some parameter.

1) arr[] : {4, 5, 8, 19, 21} → YES. ↳ Ascending

2) arr[] : {21, 19, 8, 5, 4} → YES ↳ Decreasing

3) arr[] : {1, 5, 3, 9, 6, 10, 12}

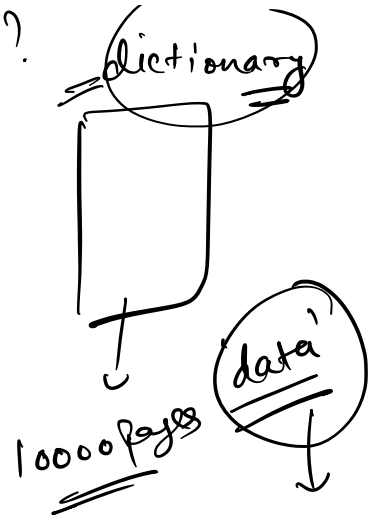
0	1	2	3	4	5	6
1	5	3	9	6	10	12
↓	↓	↓	↓	↓	↓	↓
1	2	2	3	4	4	6

↳ numbers of factors of elements.

(Custom Sorted)

→ Why do we need sorting ?

1) Ease of searching.
2) Reduce the access time
3) Data Analytics



(10)

→ Sort 'N' numbers :-

↳ T.C :- O(N log N)

C++
Java
C
Python
→ Inbuilt Sorting function → O(N log N)

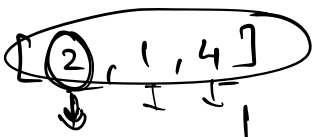
C++ :-

sort(A, A+n) ✓

Problem 1: → Amazon / Adobe / GS → Goldman Sachs.

Min Cost to Remove All Elements
- Given N array elements, at every step, remove an array element.
- Cost to delete/remove element = Sum of all the elements in the array
- Find the minimum cost to remove all the elements.

Example 1: [2, 1, 4]
Example 2: [3, 5, 1, -3]



$C_1 = 2+1+4$

$[1, 4] \rightarrow [4]$

$C_2 = 1+4$

$C_3 = 4$

$C = 7+5+4$

$= 16$

$[2, 1, 4]$

$C_1 = 4+1+2 = 7$

$C_2 = 2+1 = 3$

$C_3 = 2$

$C = 12$

$\Rightarrow [4, 5, 1]$

1) $4+6+1$

$6+1$

1

2) $6+4+1$

$4+1$

1

3) $6+4+1$

$4+1$

4

4) $4+6+1$

$1+6$

6

⇒ Observation | Hint :-

A :- a_1 | a_2 | a_3 | a_4

Delete a_4 ⇒ $a_1 + a_2 + a_3 + a_4$

a_3 ⇒ $a_1 + a_2 + a_3$

a_2 ⇒ $a_1 + a_2$

a_1 ⇒ a_1

Cost ⇒ $4*a_1 + 3*a_2 + 2*a_3 + a_4$

Minimize

$4*a_1$: 4th max / min

$3*a_2$: 3rd max

$2*a_3$: 2nd max.

a_4 : max (1st)

[4, 6, 1]

↓ sort in decreasing.

[6, 4, 1]

Cost = $1 \times 6 + 2 \times 4 + 3 \times 1 = 6 + 8 + 3 = 17$

0 1 2

[6, 4, 1]

$6 + 4 + 1$

$4 + 1$

1

$1 \times 6 + 2 \times 4 + 3 \times 1$

(0+1)

1. Sort the Array in decreasing order.
2. Remove the element in order.
3. Cost += ((i+1) * A[i])

Sort in decreasing order. → Library fun^

O(NlogN) ← Sort in decreasing order.

O(N) ←

```
for (i = 0; i < n; i++) {
    cost += ((i+1) * a[i]);
}
return cost;
```

T.C ⇒ O(NlogN + N) ⇒ O(NlogN)

S.C ⇒ O(1)

DIY :- Try to write the code by sorting in Ascending order.

Problem 2: → Amazon / Google / FB / Microsoft - - - -

Noble Integer

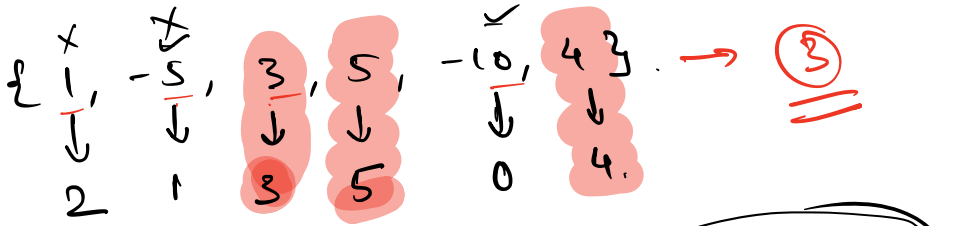
Given N array elements, arr[i] is said to be noble if: (count of elements < arr[i]) = arr[i]

Count the no of noble integers in the array.

NOTE:

- Given that all the elements are distinct.

Example: [1, -5, 3, 5, -10, 4]



A: { 3, -1, 0, 2 } → 2

(Count of elements < a[i]) = a[i]

A: { 3, 6, 0, 2 } → 1

→ Brute force :-

```
int countLesser(int a[], int n){
    count = 0;
    for(i=0; i<n; i++){
        if(a[i] < x){
            count++;
        }
    }
    return count;
```

O(N)

```
    }
    int res = 0;
    for (i = 0; i < n; i++) {                    → O(N)
        if (countLesser(arr, arr[i])
            == arr[i])
            res++;
    }
    return res;
```

T.C ⇒ $O(N^2)$

S.C ⇒ $O(1)$

⇒ [-10, -5, 1, 3, 4, 5] → Sorted Array.

0 1 2 3 4 5

0 1 2 3 4 5

```
Sort(A)
if (arr[i] == i)      res++;
```

count[i]

1. Sort the Array. → $O(N \log N)$
2. Check if (arr[i] == i).
3. if arr[i] == i then res++;

O(N)

{-4, -2, 0, -1, -6} → 0

1 2 4 3 0

T.C ⇒ O(N log N)

S.C ⇒ O(1)

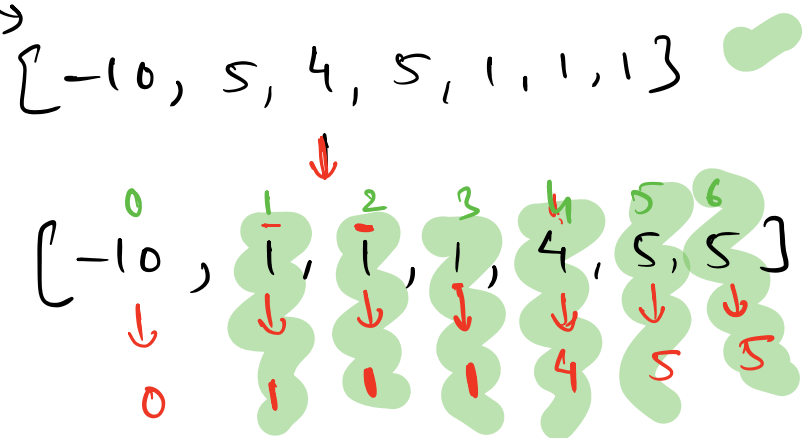
Problem 2 (Variation):

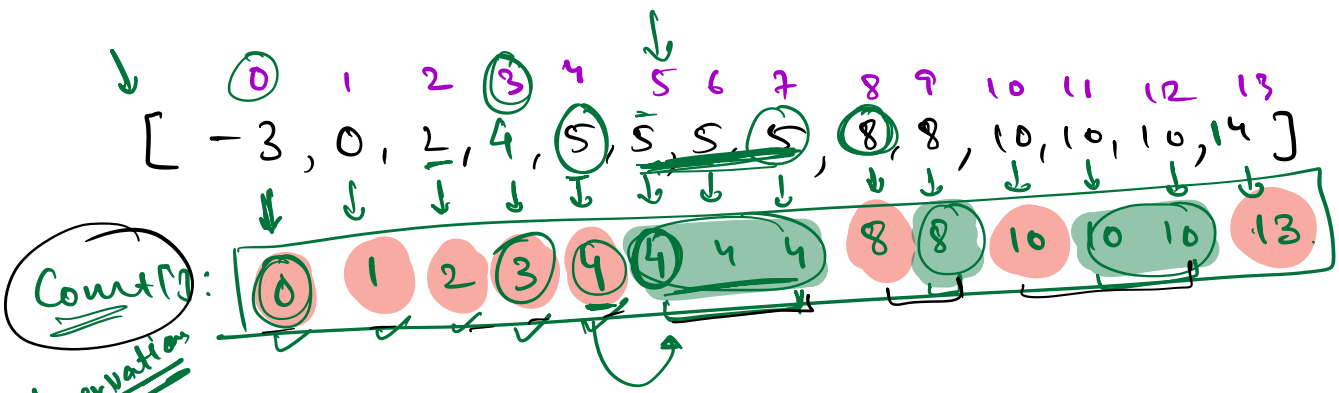
Noble Integer 2

Given N array elements, arr[i] is said to be noble if: (count of elements < arr[i]) = arr[i]
Count the no of noble integers in the array. NOTE:

- Given that elements are **NOT distinct**.

Example: [-10, 5, 4, 5, 1, 1, 1]



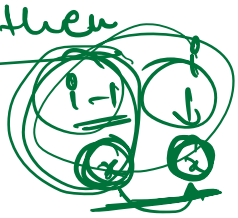


Observations

1. Previous solution can be used for some indexes.

2. If an element is repeating then

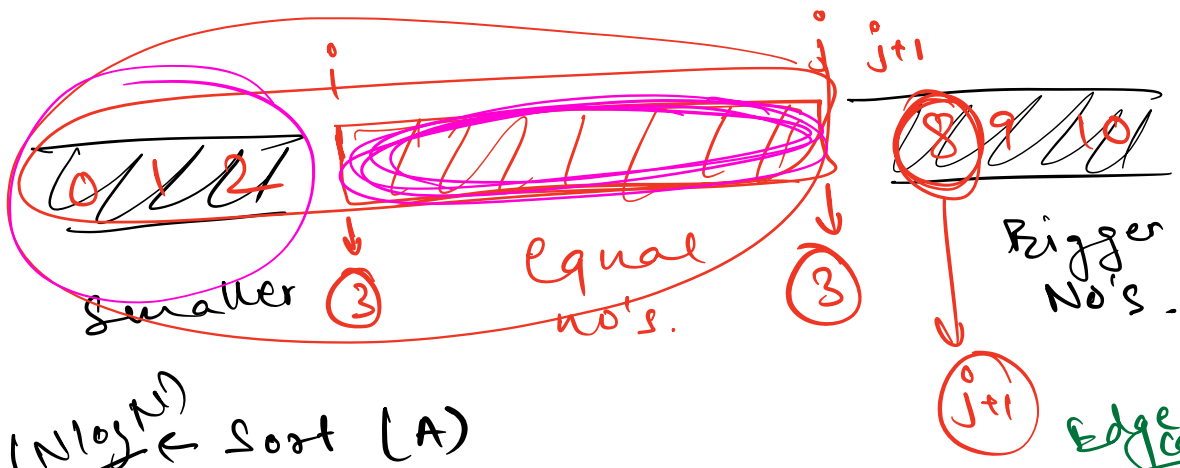
$$a[i] == a[i-1]$$



from

3. `if ( arr[i] != arr[i-1] ) Count[i] = i;` — Previous Soln

4. `if ( arr[i] == arr[i-1] ) Count[i] = Count[i-1];`

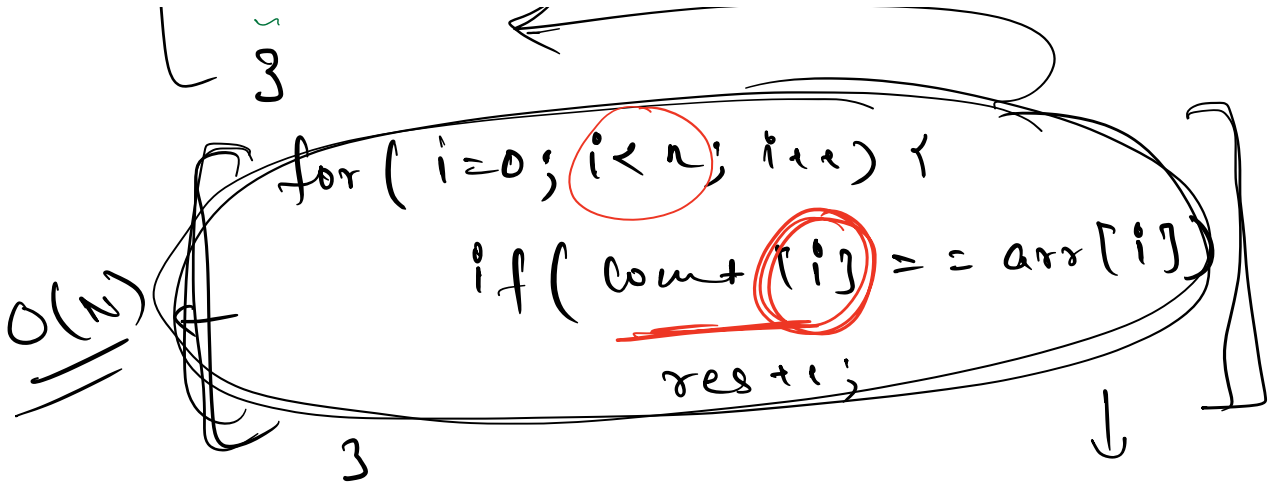


Edge case.

Index Out of Bound.

```
O(NlogN) ← Sort (A)
Count [N] = { }
Count [0] = 0;
for ( i = 1; i < N; i++ ) {
    if ( arr[i] == arr[i-1] ) {
        Count[i] = Count[i-1];
    }
    else {
        Count [i] = i;
    }
}
```

O(N) ←



T.C ⇒ O(N log N)

S.C ⇒ O(N)

↓

O(1)

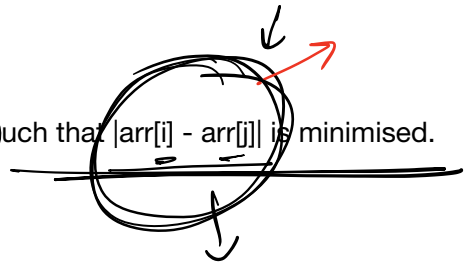
Problem 3:

Minimise Difference

Given N array elements, find the pair of indices (i, j) such that |arr[i] - arr[j]| is minimised.
Return this minimum value as your answer.

NOTE:

- i and j should be different :P
- |x| is the absolute value of x



Example: [9, 14, 21, 7, -3, 4, 26, 10]

$|a[i] - a[j]| \rightarrow$ minimum.

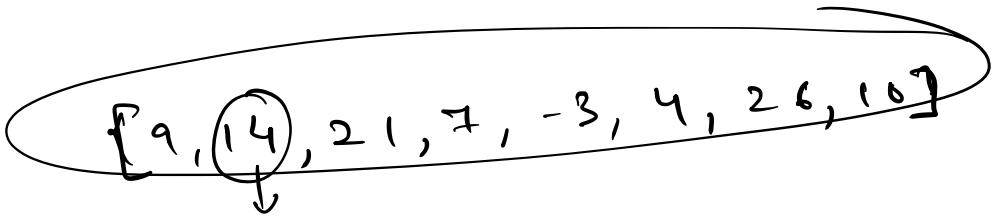
Brute force

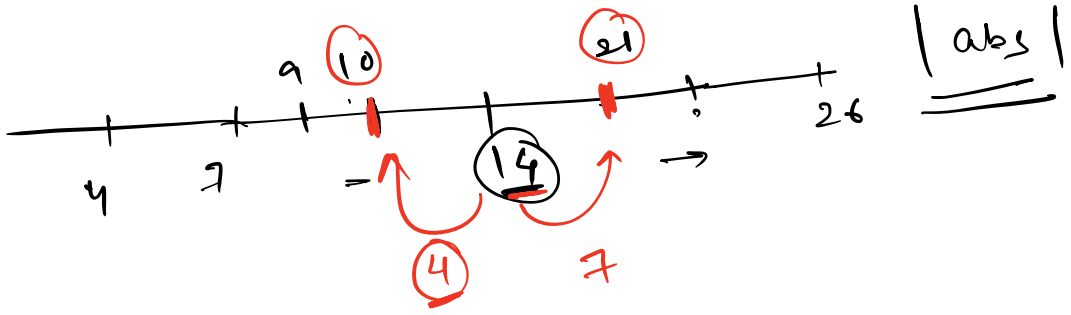
→ Check for every pair of no's.

```
ans = INT_MAX;
for (i=0; i<n; i++){
    for (j=0; j<n && j!=i; j++)
    {
        ans = min(ans, arr[i]-arr[j])
    }
}
```

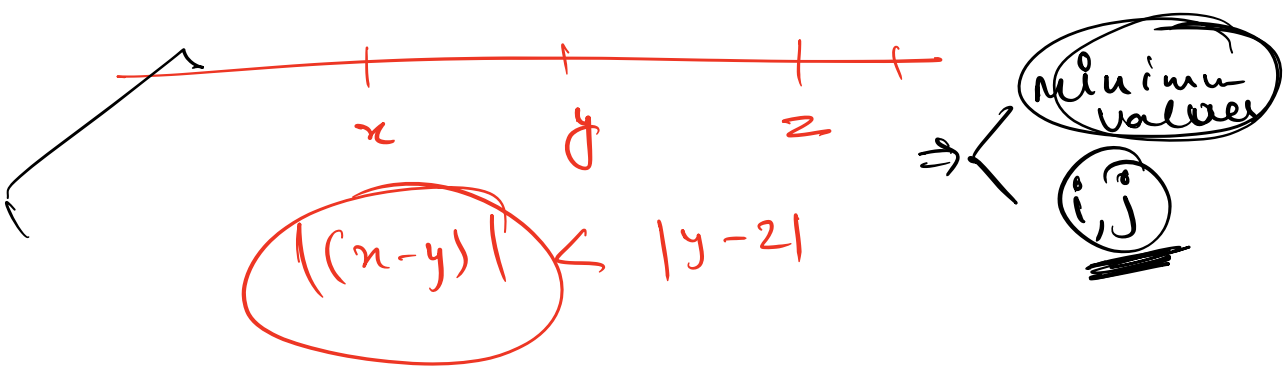
T.C ⇒ $O(N^2)$

S.C ⇒ $O(1)$





↓ SORT



→ Sort the array

→ find & compare the differences of adjacent values.

O(NlogN) ↔ sort (A)

O(N) ← for (i=1; i<n; i++)

ans = min (ans, arr[i] - arr[i-1]);

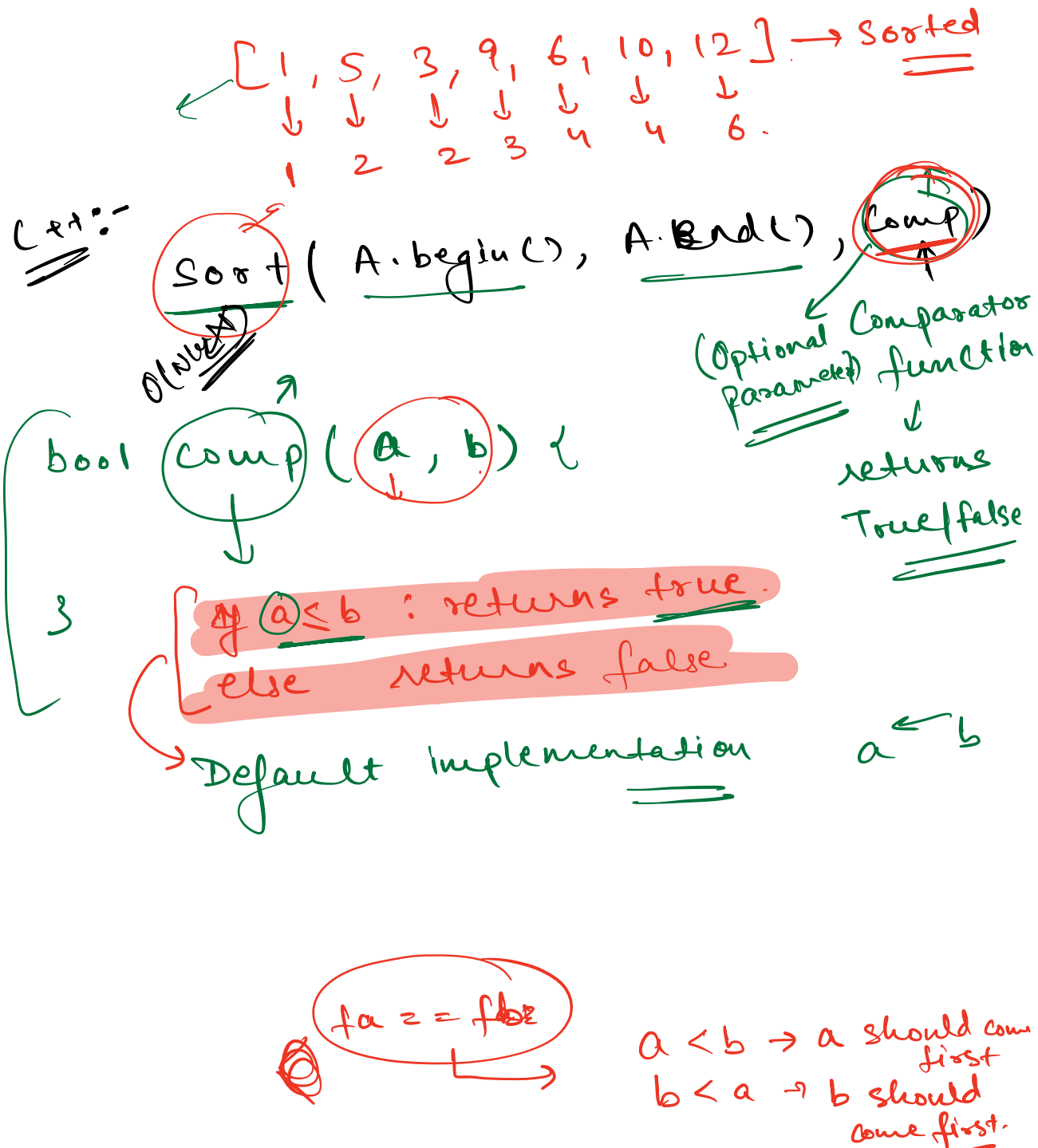
}

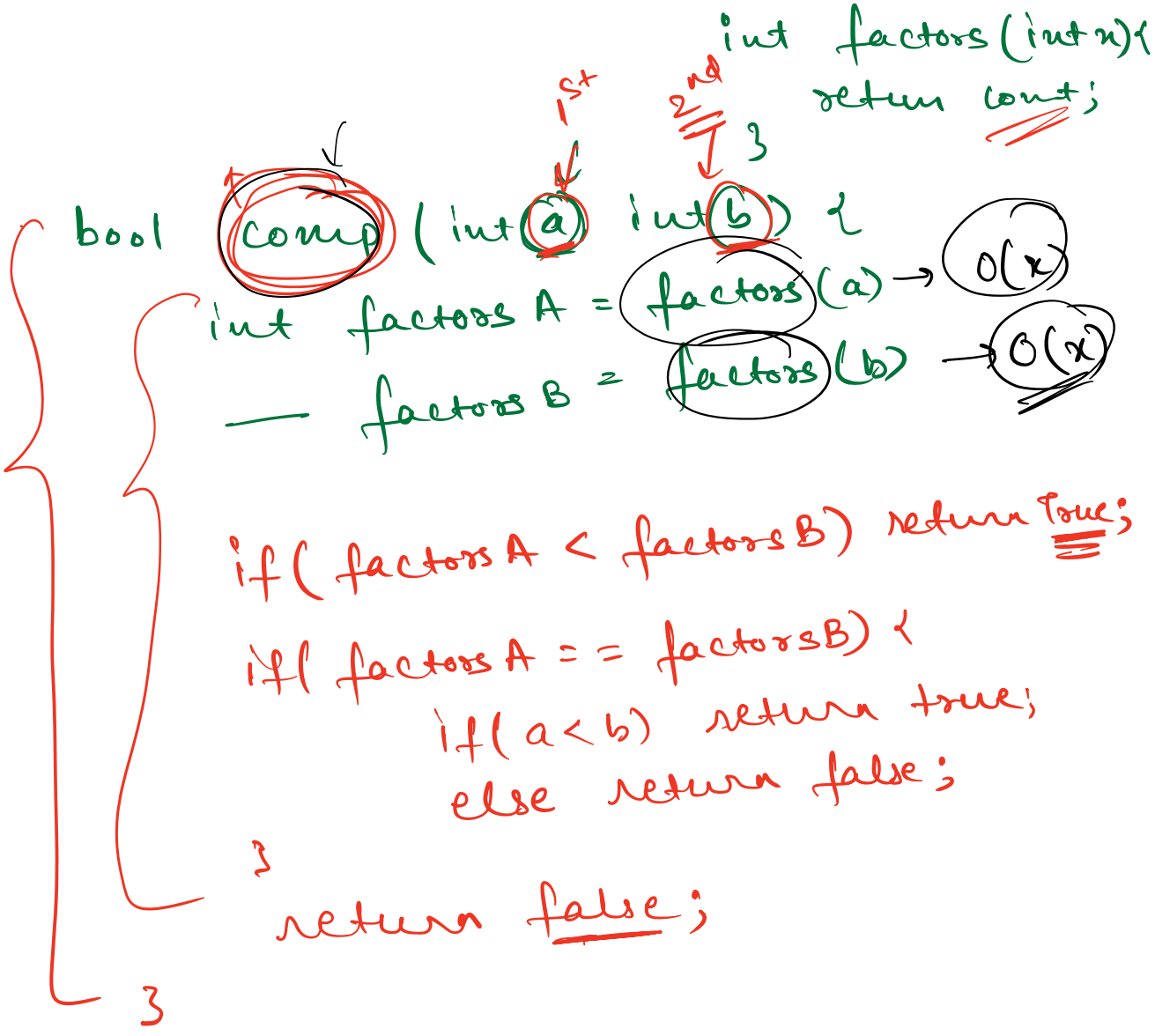
O(NlogN), O(1)

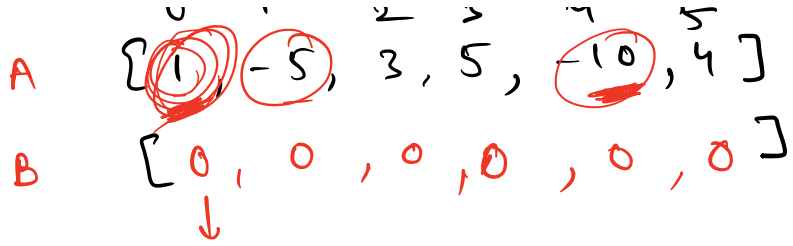
Introduction to Comparator

Sort the given array of N elements based on no of factors without using any extra space.

Example (already sorted): [1, 5, 3, 9, 6, 10, 12]




int factors(int n){
return count;
}
1st
2nd
bool comp(int a, int b) {
int factorsA = factors(a) → O(x)
factorsB = factors(b) → O(x)
if(factorsA < factorsB) return true;
if(factorsA == factorsB) {
if(a<b) return true;
else return false;
}
return false;
}
sort(A, A.end, comp);
↳ sort the Array based on the
no. of factors!


A
[1, -5, 3, 5, -10, 4]
B
[0, 0, 0, 0, 0, 0]