INTRODUCTION TO SORTING

Sorting Algo.

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

- Questions
- Private Chat

Introduction

inc, dec, custom

Sorting => Arranging data in an orderly manner based on parameter

① arr[5] = [1, 3, 5, 11, 14] increasing order

② arr[5] = [14, 11, 5, 3, 1] decreasing order

③ arr[5] = [0, 0, -1, -1, 1, 1, 1]

order { [0, -1, 1] } If this is the order than sorted — custom

④ arr[7] = [1, 5, 3, 9, 6, 10, 12] ???

1 2 2 3 4 4 6

=> sorted based on no of factors

(Q) Why do we need sorting?

- easy access for data => Algorithms

- reduce the time of access =>

Sort N numbers => O(NlogN) most efficient

Sorting - 1, -2 } Adv

In-built sorting also.

✓ Syntax.

Python	sorted/sort.
C++	sort(_,_,_)
C	qsort(_,_,_)
Java	sort()

Problem 1:
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]

① Understand
② Observations (before performing opn) = TC, SC.
③ pseudocode
④ Implement
⑤ Test
✓ Time & Space Complexities.

eg1 ⇒ [2, 1] → cost = 2+1 = 3

[1] → cost = 1

3+1 = 4

min cost = 4

i) Remove 2 ⇒ 2+1 = 3
ii) Remove 1 ⇒ 1 = 1
3+1 = 4

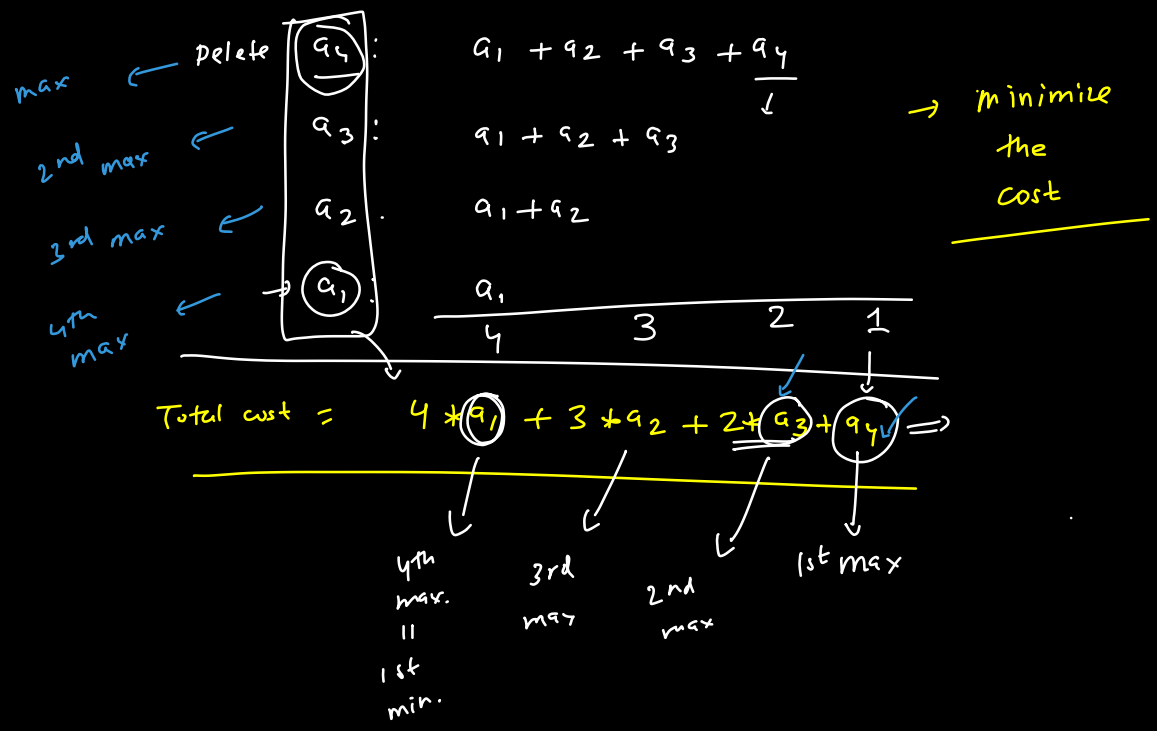
i) Remove 1: 1+2 = 3
ii) Remove 2: 2
3+2 = 5

eg2 [2, 1, 4]

Obs 1 * cost depends on the order of removals

2, 1, 4	1, 2, 4	4, 2, 1	An element does not contribute after it is removed.
2+1+4	1+2+4	(4)+2+1	
1+4	2+4	2+1	
4	4	1	
Total cost: 2+2+12 = 16	1+4+12 = 17	4+4+3 = 11	

Hint $A = [a_1, a_2, a_3, a_4]$



→ minimize the cost

⇒ Sorted in descending order

$(i+1) * arr[i]$

$A = [2, 1, 4]$

id 0 1 2

$[4, 2, 1]$

ans = 0

4 × ans = 4 * 1 = 4

2 × ans += (2 * 2 = 4)

ans = 8

1 × ans += (3 * 1 = 3)

= 11

A = [4, 6, 1]

0 1 2

[6, 4, 1]

6*1 + 4*2 + 1*3 = 17

---

ans = 0;

Tc: O(NlogN) ← sort A in descending order; ⟹

O(NlogN)

O(N) for(i=0; i<N; i++){

ans += ((i+1) * A[i]);

}

return ans;

Sc: O(1)

Problem 2:
Noble Integer

strict <

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:
1. • Given that elements are distinct. — no duplicates

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

Amazon, Facebook, Google

1

[-10, 1, 3]

1 element is < 1.

[-10, 1, 1, 3]

<= 1 (not same)
< 1

A = [3, -1, 0, 2]

[3, 0, 1, 2]

count < count of elements < arr[i]

[1, 2, 3, 4, 5, 7]

count<: 0, 1, 2, 3, 4, 5

c

arr[i] = c[i]

eg 2

idx.

count

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

0 1 2 3 4 5

O(N)

```
int countLesser(vector<int> arr, int x) {
    int res = 0;
    for(i=0; i<N; i++){
        if(arr[i] < x) {
            res++;
        }
    }
    return res;
}
```

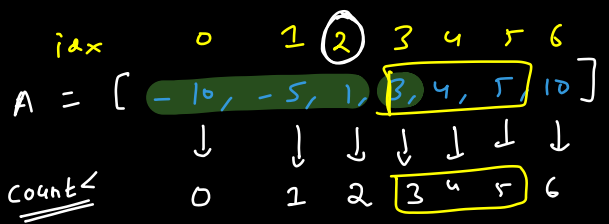
→ Brute force

```
res = 0;
for(i=0 to N){
    if(arr[i] == countLesser(arr, arr[i]))
        res++;
}
```

3

O(N^2)

we can get this in O(1) once we've sorted arr.



⇒ No duplicates

① sort in asc. order — O(NlogN)

② count = 0 — O(1)

③ for (i=0; i<N; i++){ — O(N)

    if (i == arr[i]) count ++; — O(1)

}

④ return count; — O(1)

TC : O(NlogN)

SC : O(1)

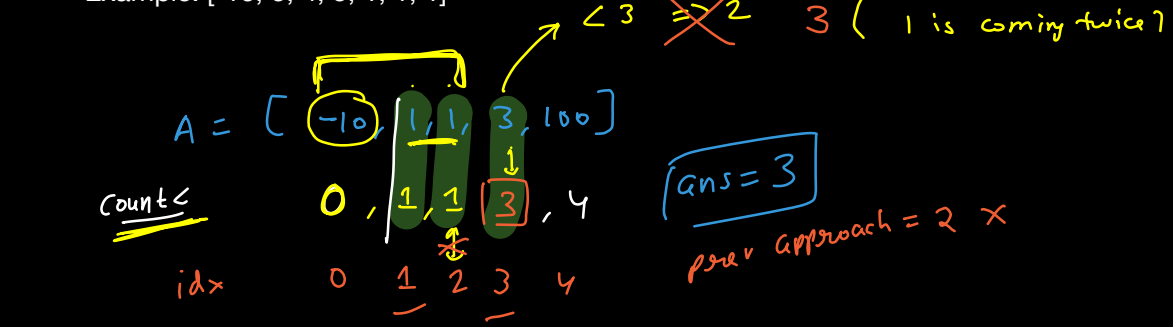
Problem 2 (Variation):
Noble Integer 2

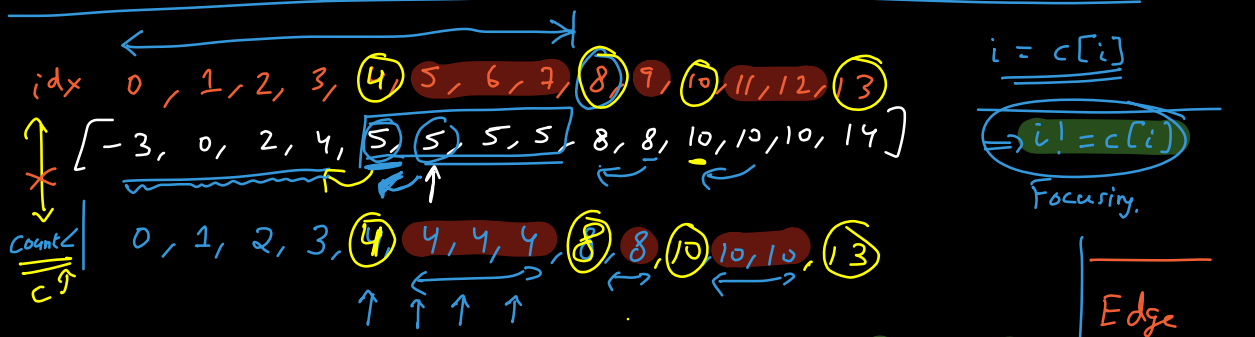
Given N array elements, arr[i] is said to be noble if:
(count of elements < arr[i]) = arr[i]

Count the no of noble integers in the array.

NOTE:
- Given that elements are **NOT distinct.** => can be duplicates

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





1) ✓ Prev solution can be applied when i = c[i]

Edge Cases

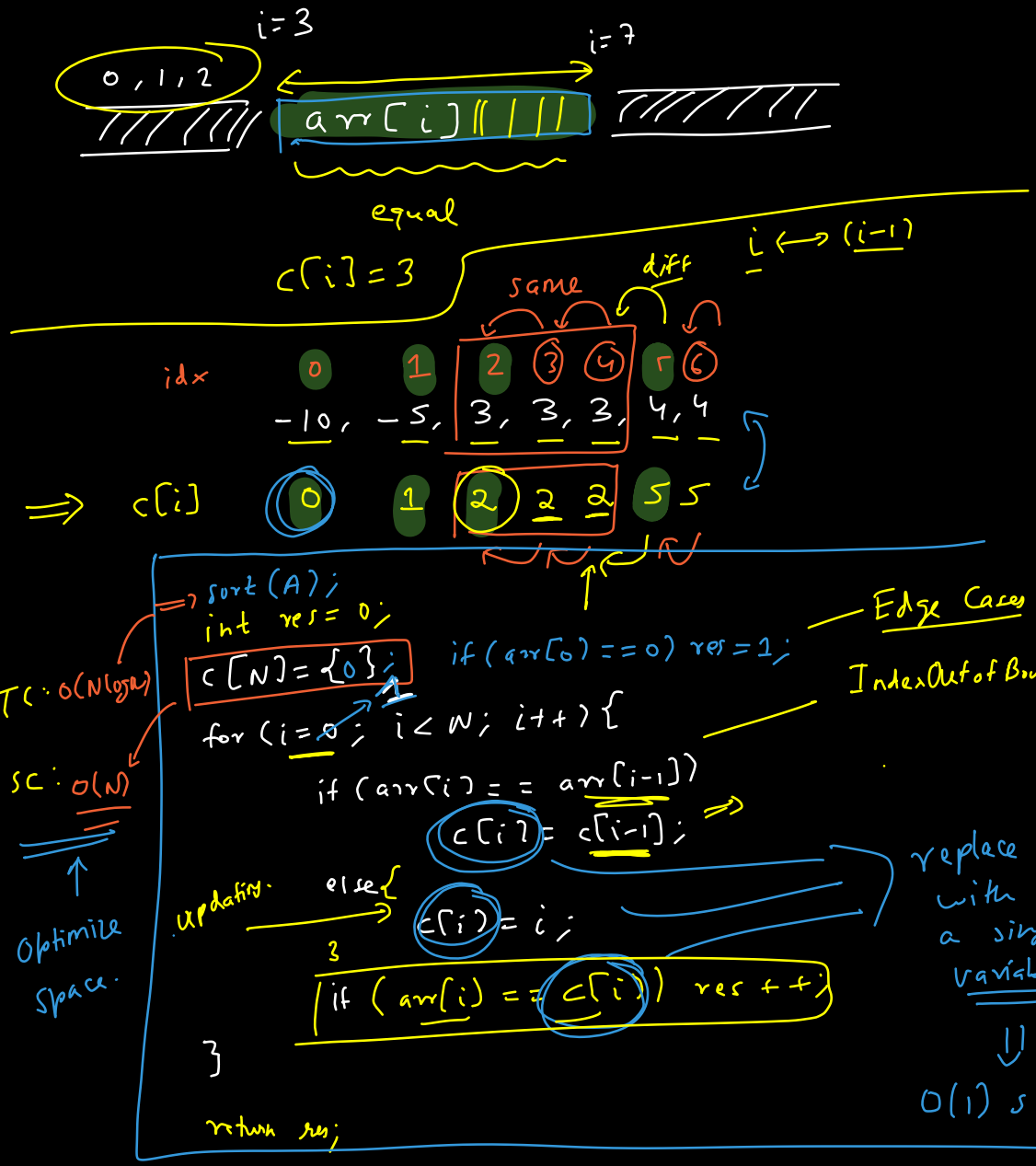
2) ✓ If an element started repeating, then arr[i] = arr[i-1]

Observations

3) if arr[i] != arr[i-1]
c[i] = i

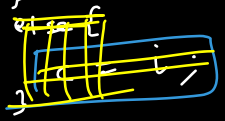
Key observation

4) if arr[i] == arr[i-1]
c[i] = c[i-1]


i=3
i=7
0, 1, 2
arr[i]
equal
c[i]=3
same
diff
i ↔ (i-1)
idx
0 1 2 3 4 5 6
-10, -5, 3, 3, 3, 4, 4
c[i]
0 1 2 2 2 5 5
sort(A);
int res=0;
c[N]={0};
for(i=0; i<N; i++){
if(arr[0]==0) res=1;
if(arr[i]==arr[i-1])
c[i]=c[i-1];
else{
c[i]=i;
}
if(arr[i]==c[i]) res++;
}
return res;
TC: O(NlogN)
SC: O(N)
Optimize Space.
Edge Cases
Index Out of Bounds
updation.
replace with a single variable.
O(1) space!

```
=> sort(A);
int res = 0;
int c = 0;
for (i = 1; i < N; i++) {
    if (arr[i] != arr[i-1]) {
        c = i;
    }
    if (arr[i] == c) res++;
}
return res;
```

if (arr[0] == 0) res = 1;



0 1 2 3 4 5 6 7

-10 1 1 1 9 5 5 1000

0 1 1 2 4 5 5 7

c = ∅ 1 4 5

res = ∅ X X 3 4 5 6

O(N log N)

T

O(1)

S

Problem 3:
Minimise Difference

Given N array elements, find the pair of indices (i, j) such that |arr[i] - arr[j]| is minimised.

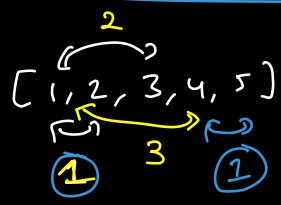
Return this minimum value as your answer.

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

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

|x|

|3 - 5| = |-2| = 2



|9 - 10| = 1

Brute Force: Go over all pairs and find the min diff.

```
ans = INT_MAX
for i = 0 to N-1          ==>  Fix an outer i
    for j = i+1 to N-1
        ans = min(ans, |arr[i] - arr[j]|)
return ans
```

TC: $O(N^2)$

SC: $O(1)$

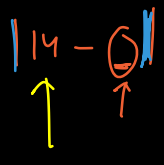
Observation

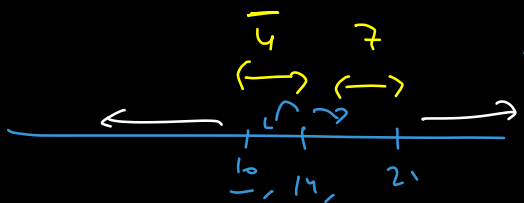
Ex 1

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

Nearest larger element

Nearest smaller

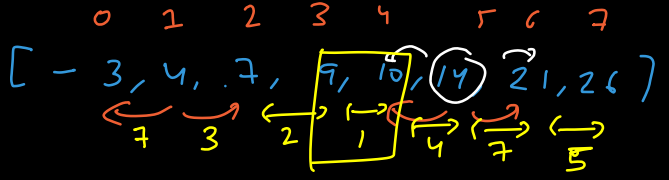




minimize the distance

- get the closest elements.

Sort A



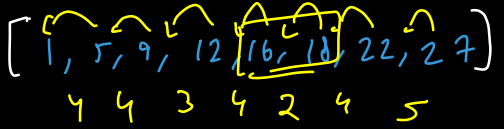
Find the min diff b/w any 2 adjacent elements.

Pseudo-code

TC: O(NlogN)

SC: O(1)

```
res = INT_MAX
Sort(A) in asc. order          → NlogN
for(i=1; i<N; i++){            → N
    res = min(res, arr[i]-arr[i-1]);
}
return res;
```

✓ 

Introduction to Comparator → allows us to compare elts based on some parameter while sorting

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]

A = [1, 2, 4, 10, 5, 6, 3]

numFactors 1, 2, 3, 4, 2, 4, 2

O = [1, 2, 3, 5, 4, 6, 10]

If 2 elements have same num of factors, smaller num should come first.

```
int factors(int a){
    return cnt
}
```

FG given!

Sort(begin, end, Comp)

```
bool comp( A, B ){ // If A should come before B return true
                   // otherwise false
    int cA = factors(A);
    int cB = factors(B);
    if ( cA < cB ) return true;
    if ( cA == cB and A < B ) return true;
    return false; // → B comes before A
}
```

CA = CB →

CA < CB → T

CA > CB → F

Recap & Doubts

Q) How comparator used in compiler code?

operator overloading.

sort A[I=] (arr, st, end, comp)

internally { if (A < B) cmp(A,B) } swap(A,B)

Comparator

A = [1, 7, 6, 5]

factors = [1, 2, 4, 2]

[1, 5, 7, 6]

sort (begin, end, comp)

A[i] < A[j]

cmp(A[i], A[j]) = true

comp(a, b)

returns true if a should come before b.

otherwise false.

```
bool comp( int a, int b) {
    int cA
        cB
        .
        .
        .
}
```

① Problem Solving + Learning — Both required.

② Competitive Programming
- more about Maths & Observations
- identifying the right patterns ⇒ practice
- reduce to a similar problem
- speed
- accuracy.

→ Interviews

time

③ When you're ready for interviews. → Communication
- clarifying.

Readiness →
- able to identify patterns
- solve problems.
- quickly.
- explain your ideas.

- Mock
- Interviews.

↓

- Track No. of problems category.

Graphs
DP.

1% ⇒ 37.