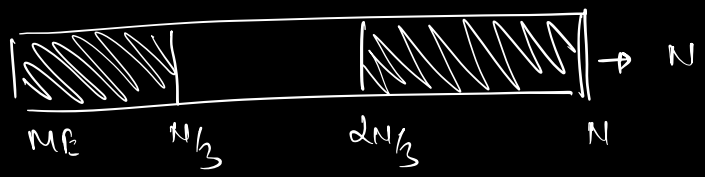
: Majority element :- (freq > N/3).



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
arr ⇒	3	4	3	2	3	2	2	2	3	2	3	3	4	4	5	2

L = 16.

ME → freq > N/3 minm freq ⇒ 16/3 + 1 = 5 + 1 = 6

count of 3 ⇒ 6 (ME)

remaining length ⇒ 16 - 6 = 10.

count of 2 = 6 (ME) 6 > 16/3

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
ex ⇒	3	~~2~~	~~3~~	~~5~~	3	3	~~3~~	~~6~~	~~7~~	1	3	14	8	0	9	1	3	7	6	12

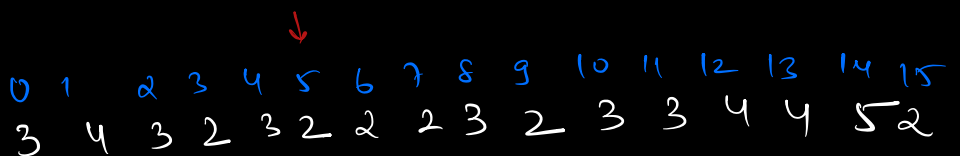
length = 20

minm freq ⇒ > 20/3 ⇒ > 6 ⇒ 7.

count of 3	len	
7 > 20/3	20 → ME	remove 3 distinct
6 > 17/3	17 → ME	
5 > 14/3	14 → ME	removed

* After removing 3 distinct elements, ME remains the same

ME1 ME2
C1 C2



ME1 = 3 ME2 = 2 ⇒ after finding most probable MEs, verify by
C1 = 2 C2 = 1

1000 → pseudo

C1--
C2--

TC ⇒ O(N)
SC ⇒ O(1)

Sorting

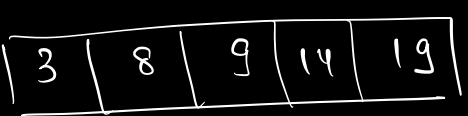
Agenda:- i) why sorting
ii) problems using sorting
iii) comparators

Sorting :- arranging ~~numbers~~ data in ~~asc/desc~~ specific order according to a factor.

Uses ⇒ eCommerce ⇒ sort by price
sort by most sold

Interviewbit ⇒ sort question by toughness
sort question by company

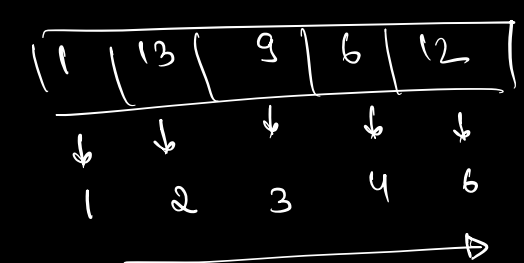
Sort a deck of cards ⇒ i) By color
ii) By type
[heart, diamond, clubs, spade]
iii) By value
iv) By face value

⇒ 

⇒ sorted in asc by value

⇒ 

⇒ sorted in desc by value

⇒ 

⇒ Sorted in asc by no. of factors.

Sorting :- helps a lot in searching.

Q. Given an array of size N, you can remove one array element at a time. Cost of removal ⇒ sum of all elements in arr. just before removal, remove all elements by minm cost.

A = 2, 1, 4

remove 2 ⇒ (2+1+4) = 7

A = 1, 4

remove 1 ⇒ (1+4) = 5

A = 4

remove 4 ⇒ 4 ⇒ 4

(16)

A = 2, 1, 4

remove 4 ⇒ (2+1+4) = 7

A = 1, 2

remove 2 ⇒ (1+2) = 3

A = 1

remove 1 ⇒ (1) ⇒ 1

(11)

Quiz A = 4, 6, 1.

remove ⇒ 6 ⇒ (4+6+1) = 11

[4,1] → remove 4 ⇒ [4+1] = 5

[1] ⇒ remove 1 ⇒ 1

17

OP = 17.

Quiz ⇒ 3 5 1 -3

remove 5 ⇒ [3+5+1+(-3)] = 6

[3 1 -3] ⇒ remove 3 ⇒ [3+1+(-3)] = 1

[1 -3] ⇒ remove 1 ⇒ [1+(-3)] = -2

[-3] ⇒ remove -3 ⇒ [-3] ⇒ -3

2

array ⇒ a b c d

remove a ⇒ a+b+c+d

[bcd] remove b ⇒ b+c+d

[cd] remove c ⇒ c+d

[d] remove d ⇒ d

a+2b+3c+4d

total cost ⇒ $a + 2b + 3c + 4d$

minimise

a → maximum index of array

d → minimum index value of array

a b c d ⟶ sort desc.

a → max

d → min

arr ⇒

0	1	2	3
a_0	a_1	a_2	a_3

⟶ Sort Desc

cost

a_0 ⇒ $a_0 + a_1 + a_2 + a_3$ ⇒ pf[3]

a_1 ⇒ $a_1 + a_2 + a_3$ ⇒ pf[3] - pf[0]

a_2 ⇒ $a_2 + a_3$ ⇒ pf[3] - pf[1]

a_3 ⇒ a_3 ⇒ pf[3] - pf[2]

maxm

minm

prefix sum technique

- Sort(A) // desc order
- find pfsum
- find cost for each element removal

In built sorting function :

TC ⇒ $O(N \log N)$

SC ⇒ depends on algo used for sorting

$O(1) \rightarrow O(N)$

TC ⇒ $O(N \log N) + O(N) + O(N)$

↓ sort ↓ ppsum ↓ cost

$\simeq O(N \log N)$

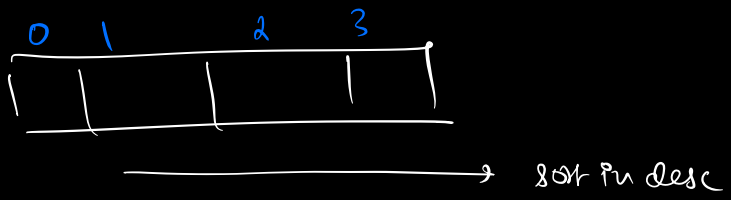
SC ⇒ $O(N)$

↓ ppsum

: Contribution technique

a b c d

cost = a + 2b + 3c + 4d



cost = A[0] + 2 A[1] + 3 A[2] + 4 A[3]

i^{th} ⇒ (i+1) * A[i]

pseudo

```
sort(A)   // desc
cost = 0
for( i=0; i<N; i++) {
    cost = cost + (A[i] x (i+1))
}
return ans;
```

TC ⇒ $O(N \log N) + O(N)$

($O(N \log N)$ ← sort, $O(N)$ ← cost)

$\simeq O(N \log N)$

SC ⇒ depends on sorting algo.

break ⇒ 10 mins | 2ST 8:35 PM

⇒ a b c d

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

→ sort in desc

a max

d is min

Amazon, Zeta

Q2. Given N elements, count no. of Noble integers present in array. [distinct elements]

Noble integer: A[i] is a noble int. if the no. of array element less A[i] == A[i]

arr ⇒ [1 -5 3 5 -10 4]

#less 2 1 3 5 0 4

O/P = 3

quiz arr ⇒ [-3 0 2 5]

#less ⇒ 0 1 2 3

O/P = 1

quiz arr ⇒ [-10, 1, 2, 3, 100]

#less 0 1 2 3 4

O/P = 3

→ Sorted & no dup. i is no. of elements lesser than A[i]

0 1 2 3

-3 0 2 5

→ Sort (asc)

#

left wala is less than 2

```
if (i == A[i])
    // Noble integer
```

elements present in left of 2 = 2 (1)

pseudo

```
Sort(A)  // asc
ans = 0
for(i=0; i<N; i++) {
        if(A[i] == i)
                ans++
}
return ans;
```

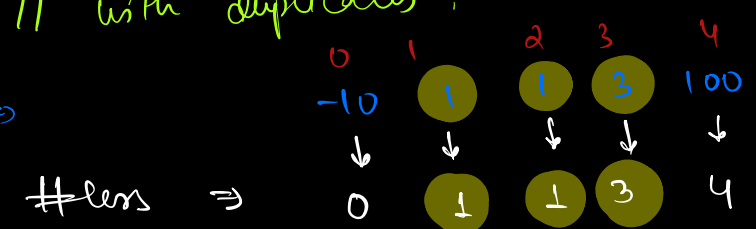
TC => O(N log N) + O(N)

~ O(N log N)

SC => depends on sorting algo

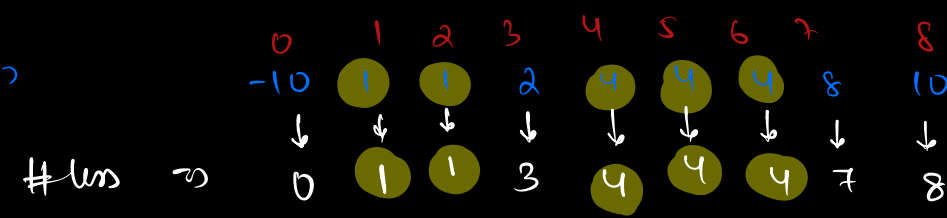
// with duplicates :-

Quiz 4 =>

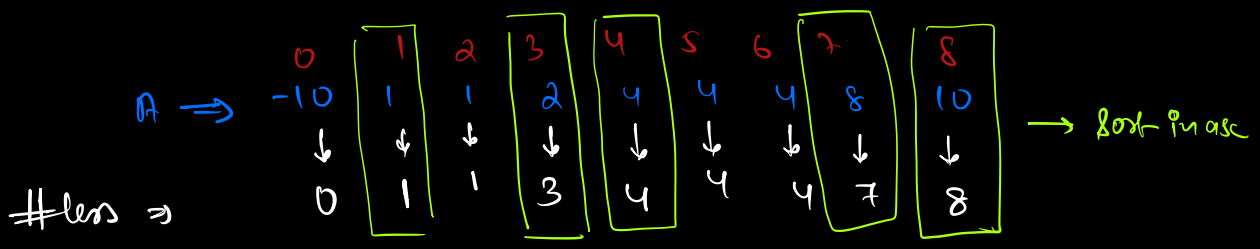


o/p = 3

Quiz 5 =>



o/p = 5



Pseudo How

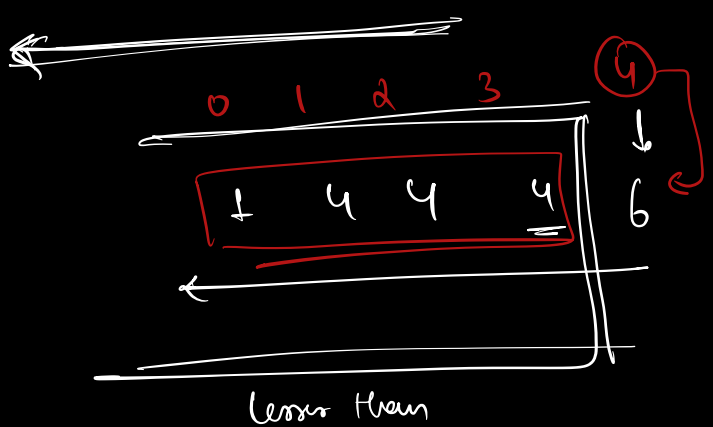
i) arr[i] == arr[i-1]

elements lesser than a[i] = elements lesser than a[i-1]

ii) arr[i] != arr[i-1]

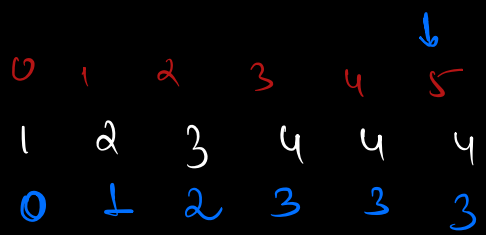
elements lesser than a[i] = i

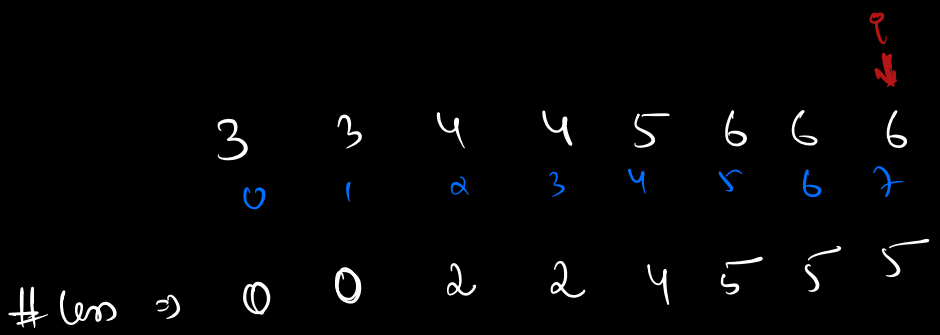
a[i-1] ✗ a[i]



5

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





part of DSA

⇒ when to use sort() ⇒ C/C++ func

⇒ implement sorting ⇒ implement