Date:

Boyer-Moore Voting Algorithm

0 1 2 3 4 5
[2, 2, 1, 1, 2, 2]

TC → O(n)
SC → O(n) → Hashtable (key, value)

{ 2 → 4 → 2
  1 → 2 majority element.

Example 1:

0 1 2 3 4 5 6
[2, 2, 1, 1, 1, 2, 2]

Majority Element = $\frac{n}{2} = \frac{6}{2} = 3$

Candidate = 2 → 1 → 2
Count = 0 1 2 1 0 1 2 1 0 1

Count value keeps on updating if same value is found & decrements with new value.
Candidate value updates only when count value is 0.
We use count only to check if it is > 0 then we have greater chance of finding majority element.

Date:

Example 2:

nums = [2, 3, 1, 3, 3]

Candidate = None → 2 → 3 → 4 → 3

Count = 0 1 0 1 0 1 0 1 2

If you find any value not repeating & near to zero then you decrement the value to zero & then increment after updating the candidate value.

Example 3:

[2, 3, 7, 3, 4]

Candidate = None → 2 → 3 → 7 → 3 → 4

count = 0, 0 1, 0 1, 0 1, 0 1

No. Majority element ≥ 4

To be majority element, frequency should be greater than n/2

Eg:- n = 8 then $\frac{n}{2} = \frac{8}{2} = 4$

frequency > 4

Date:

Count of number of inversions in an Array: (Interview Question)

↳ inversion { $i < j$ → indexes, nums(i) > nums(j) }

0	1	2	3	4	5	6	7
70	50	60	10	20	30	80	15
i	j						

n = 8

70	50
0	1
i	j

Inversion property ⇐

$i < j \Rightarrow 0 < 1$ (Indexes)

nums(i) > nums(j) ⇒ 70 > 50

70 → 50, 60, 10, 20, 30, 15
50 → 10, 20, 30, 15
60 → 10, 20, 30, 15
10 → ~~X~~
20 → 15
30 → 15
80 → 15
15 → X

Total count = 17 inversions.

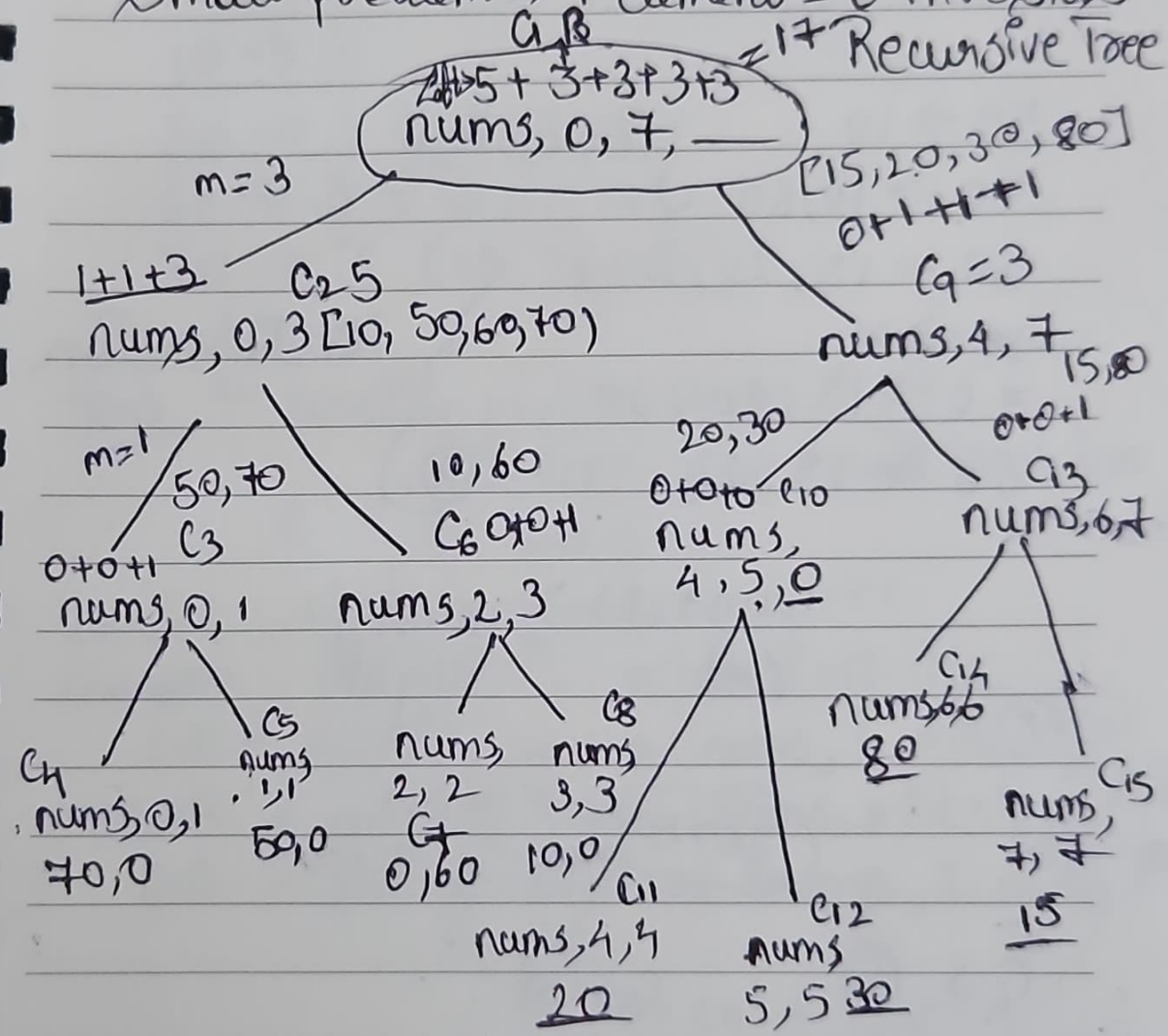
Date:

Brute force Approach:
2 for loops (for i & j)
$O(n^2)$ T.C

Divide And Conquer Approach

0	1	2	3	4	5	6	7
70	50	60	10	20	30	80	15

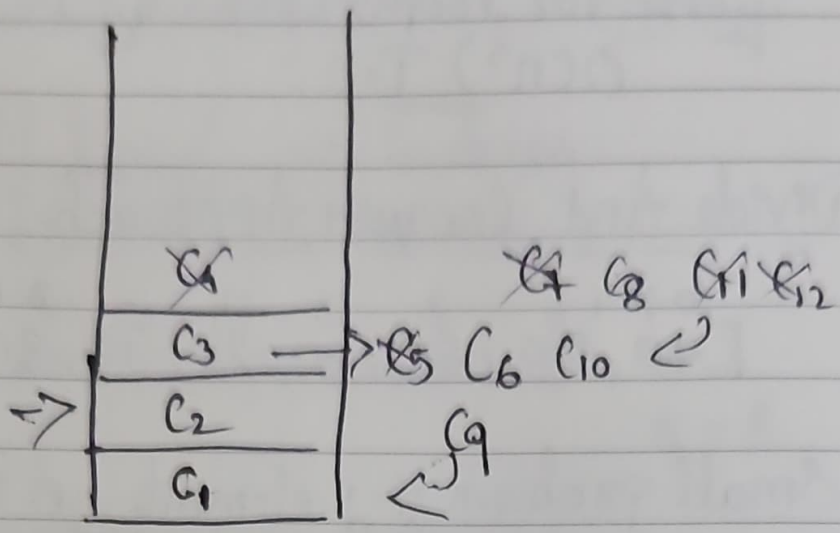
Small problem ⇒ 1 element = 0 inversions



70 → 50 (valid inversion)
60 → 10 → 1 inversion

Date:

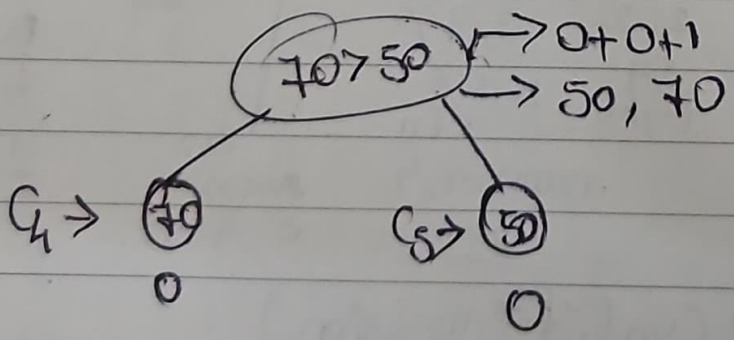
Stack → Recursion



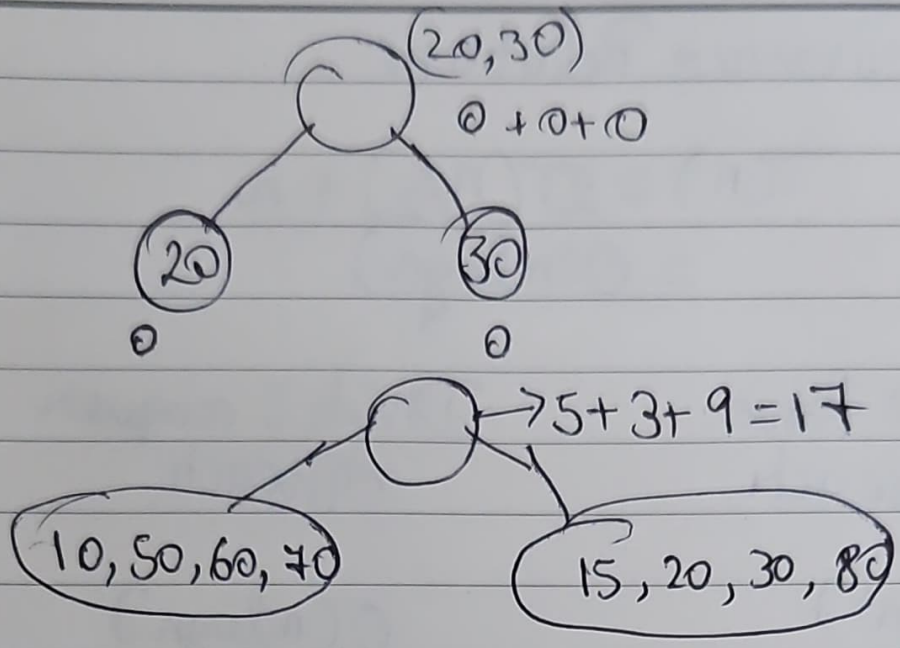
50 → 10 } Inversions
70 → 10, 60 }
50 → 15, 20, 30
10 → X
70 → 15, 20, 30
60 → 15, 20, 30 → (3)

C1 Inversion

nums(i) > nums(j)
Left Right



Date:



10 → X

50 → 3 (15, 20, 30)

60 → 3 (15, 20, 30)

70 → 3 (15, 20, 30)

Pseudocode:

T(n) ← countInversion(nums, i, j);

if i == j: → small problem

C ← count = 0

Divide & Conquer ← return nums(?) → C

else; mid = i + (j-1)/2

cntL ← countInversion(nums, i, mid) (T/2)

cntR ← countInversion(nums, mid+1, j) (T/2)

cnt = mergeProcedure(nums, i, mid, j) n

→ cnt = cntL + cntR + current inversions

return cnt → inversions

Date:

Recurrence Relation:

$$T(n) = 2T(n/2) + n$$
$$= O(n \log n)$$

Brute force Approach	Divide & conquer Approach
$O(n^2)$	$O(n \log n)$
	↓
	Optimized way