- Inversion Count → using merge sort
- Reverse Pairs
- Count of Smaller Elements after Self
- Count Subarrays with 1s > 0s
- Minimum Swaps to Sort I & II

} IMPORTANT.

① Count Inversions / Inversion Count.

Eg: arr:	8	5	3	4	1	6	2
idx:	0	1	2	3	4	5	6

Count no. of (i, j) such that $i < j$ and $arr[i] > arr[j]$

Eg: from above

(5, 8) ✗ → idx[5] > idx[8]

8 5	5 3	4 1
8 3	5 4	4 2 → 2
8 4	5 1	6 2 → 1
8 1	5 2 → 4	
8 6	3 1	
8 7 → 6	3 2 → 2	

Total = 15

Brute force → $O(N^2)$

Optimized approaches

→ Merge Sort → $O(N \log N)$ Easy

→ Fenwick Tree

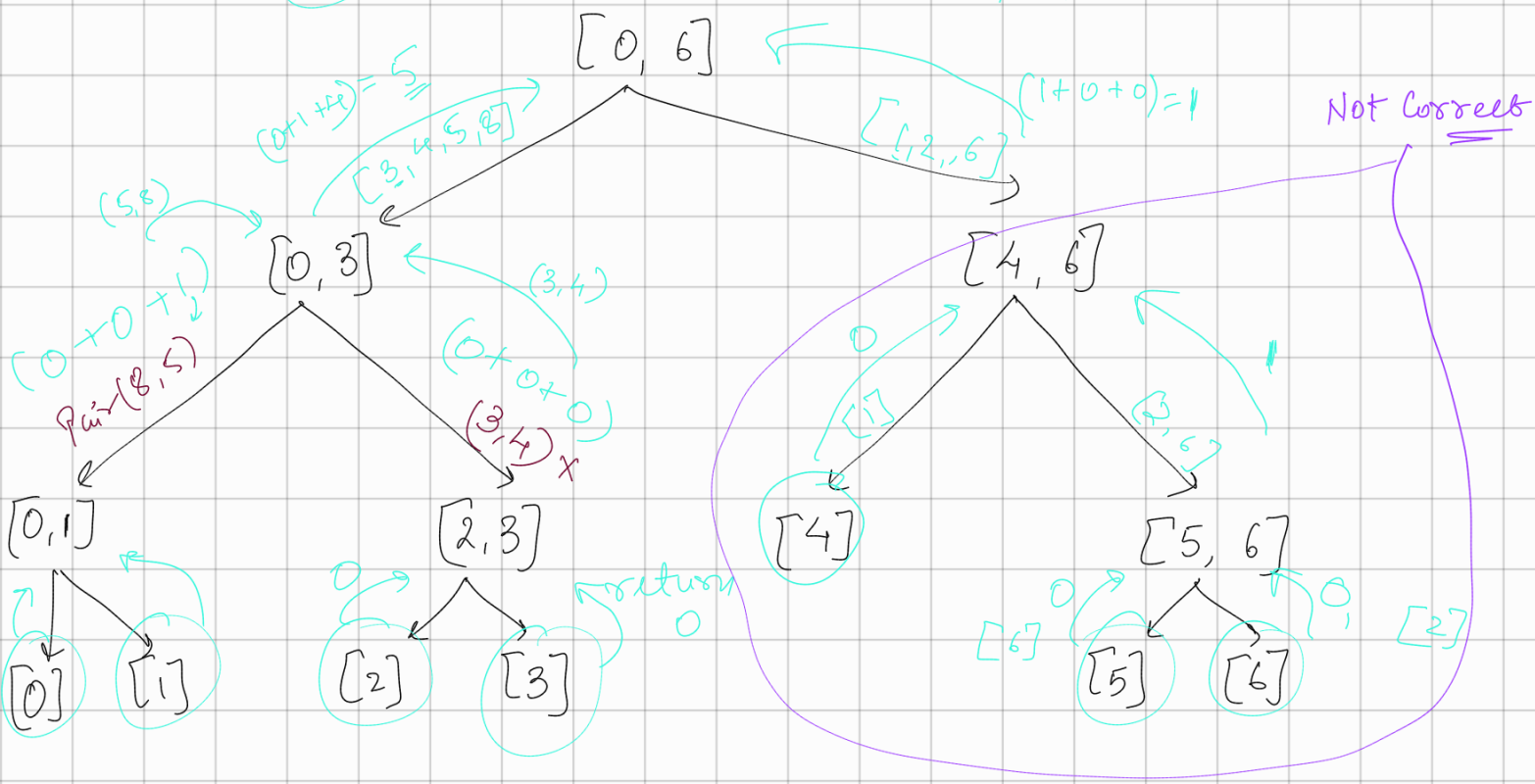
→ Policy Based Data Structure

→ Self Balancing BST, Red-Black Tree

* Using Merge Sort

8	5	3	4	1	6	2
0	1	2	3	4	5	6

(15) = (9+5+1), [1, 2, 3, 4, 5, 6, 8]



at [0,3] → [5 8] [3,4]

liw = 1, riw = 0

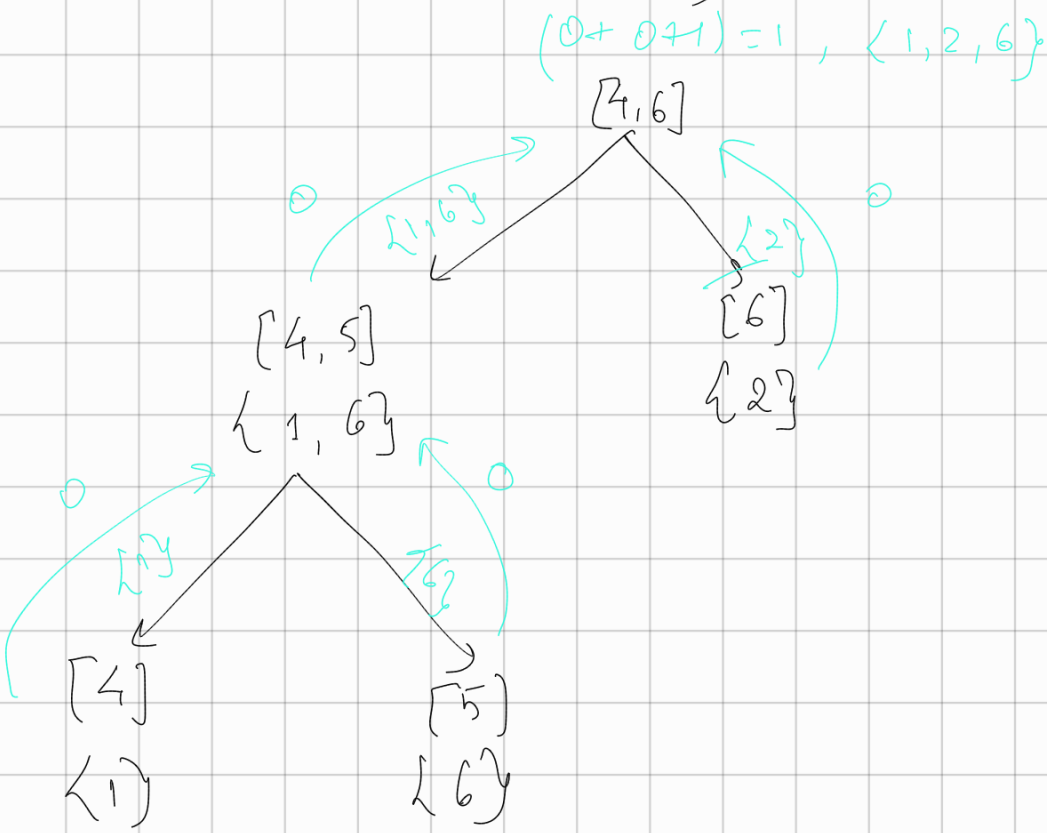
at idx = 0, 0 ⇒ we can tell no. of possible pairs are: [5,3] & next of 5 >= 5, so one more pair = [8,3] → This we can conclude at the first index of both arrays.

after this ⇒ [5, 8] [3, 4] ∵ 3 < 5, ptr2 ++ → 2 possible pairs. [5,4] & ∵ next of 5 >= 5, [8,4]

ptr1 ↑ ptr2 ↑

So, we get the correct count at [0,3] which is (4)
But, we also have to add the (lInv + rInv) to the answer. So at [0,3], the answer will be:

(1 + 0 + 4) = 5



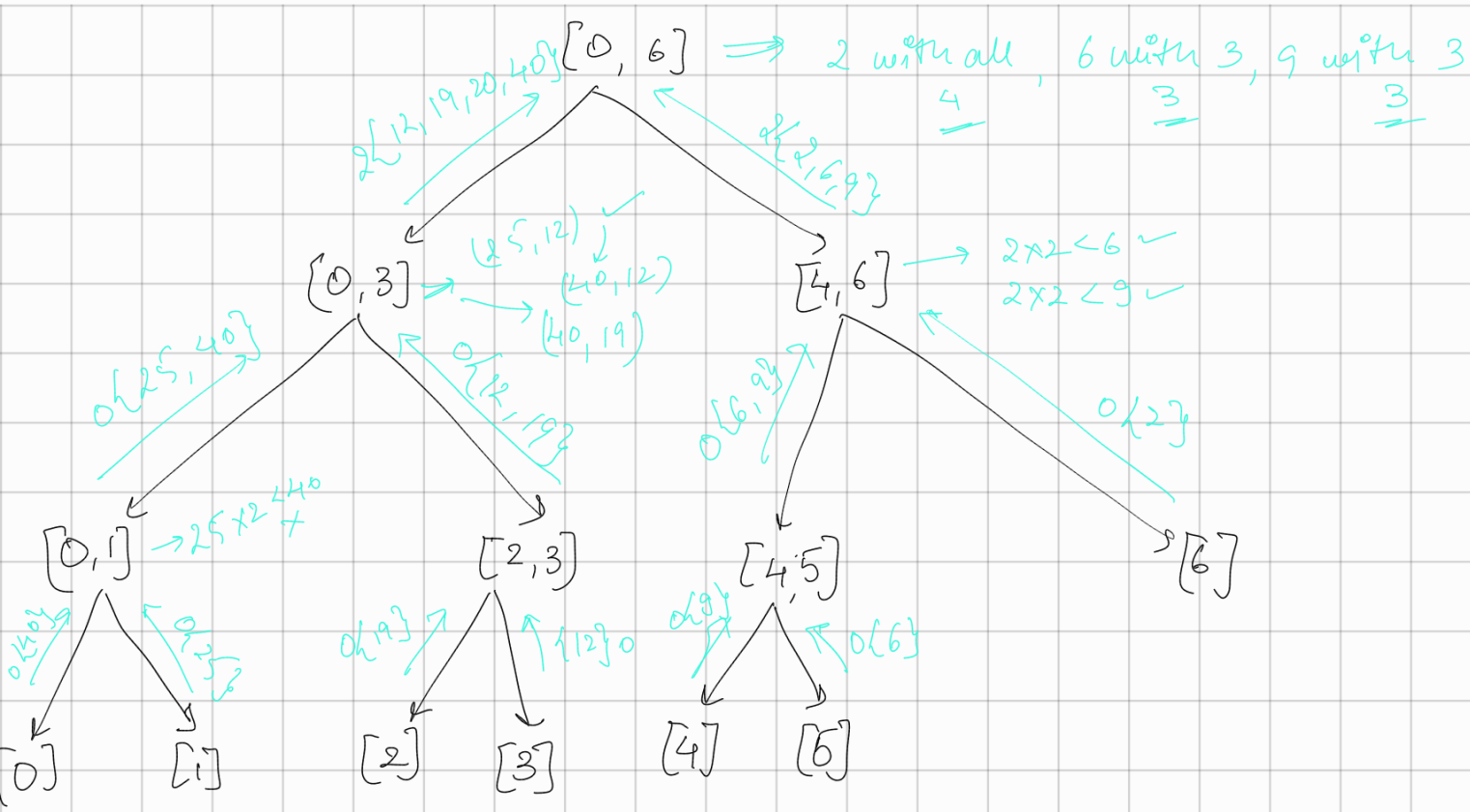
∴ CODE is same as mergeSort
except when (nums[ptr2] > nums[ptr1])
count += (arr - ptr1 + 1);

(2) Reverse Pairs

count pairs where $i < j$ & $a[i] > 2 \times a[j]$

Here, find the pairs first & then merge

Eg:	40	25	19	12	9	6	2
idx	0	1	2	3	4	5	6



③ Count Winning Streaks → GOOGLE. — Same as ↓

④ Count Subarrays with more 0's than 1's (LeetCode 2031)

count(1) > count(0)

recursion $O(2^N)$ → $O(N^3)$ → $O(N^2)$ → $N \log N$

$O(N^3)$: 3 loops

$O(N^2)$: optimized subarrays (2 loops) { prefix }

$N \log N$: inv. count { prefix }

to check, count(1) − count(0) > 0

Eg: [0 | 1 1 0 | 1 1 1 0]

idx:	0	1	2	3	4	5	6	7
1 → -1 & 0 → +1								
Prefix Sum:	1	0	-1	0	-1	-2	-3	-2

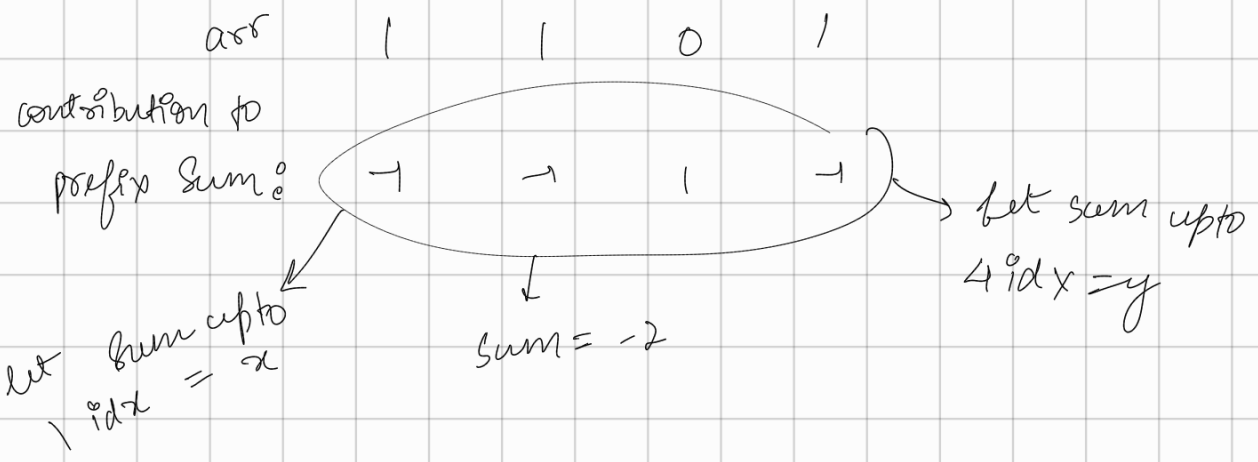
inv. count

inv. count

inv. count ✓

* If we can find an inversion count in prefix sum, the the same index elements in the original array will be a valid subarray

Eg: Subarray idx [1,4] from above



then $y = x - 2$

or $y - x = -2$

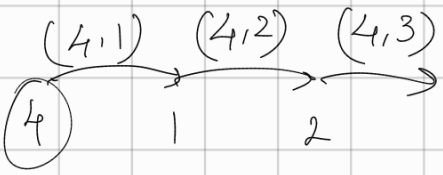
So, here whenever $y - x < 0$ / $y < x$ we can find a valid subarray

So, in essence, we only need to find the inversion count in the prefix sum array

1 0

(4) Minimum Swaps To Sort. (Adjacent)

Adjacent.



→ inv. count = swaps to sort.

Eg:

							Steps
8	5	3	4	1	6	2	6
							+
5	3	4	1	6	2	8	1
							+
5	3	4	1	2	6	8	4
							+
3	4	1	2	5	6	8	2
							+
3	1	2	4	5	6	8	2
1							
1	2	3	4	5	6	8	

Total = 15

(5) Minimum Swaps To Sort (Not Only Adjacent)

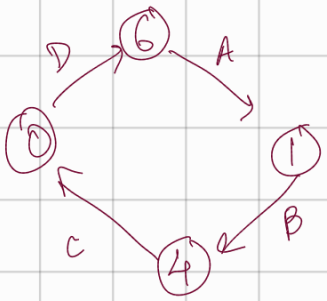
Eg:	8	5	3	4	1	6	2
idx	0	1	2	3	4	5	6
Correct Position	1	2	3	4	5	6	8

On comparing given array & correct array

already in place → do not touch

Number	8	5	3	4	1	6	2
is at	0	1	2	3	4	5	6
Need's to be at:	6	4	2	3	0	5	1
	Edge D	Edge B			Edge C		Edge A

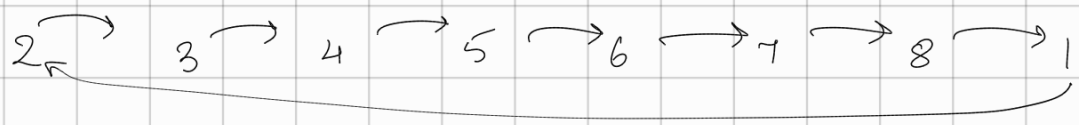
Graph for elements not at correct position



→ Edge A B C D form a cycle

Here, total number of minimum swaps = no. of edges for cycle

Corner case I

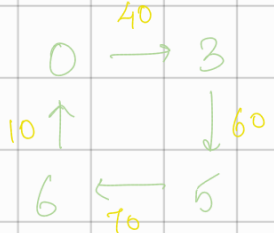


7 swaps

Corner case II

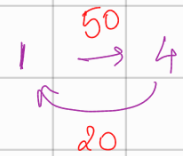
40	50	30	60	20	70	10	80
0	1	2	3	4	5	6	7
10	20	30	40	50	60	70	80

elem	from	to	
40 :	0	→ 3	✓
50 :	1	→ 4	
30 :	2	→ 2	
60 :	3	→ 5	✓
20 :	4	→ 1	
70 :	5	→ 6	✓
10 :	6	→ 0	✓
80 :	7	→ 7	



Cycle 1

4 swaps



Cycle 2

2 swaps.

Total = 6 swaps.

Consider all elements which are not at their correct place.

This is a $O(N \log_2 N + N)$ approach