Sorting 1 : CountSort & MergeSort

Question

find the smallest no. that can be formed by rearranging the digits of a given no. in an array. $0 <= A[i] <= 9$

A = [3 2 4 1] → [1 2 3 4]

A = [6 3 4 2 7 2 0] → [0 2 2 3 4 6 7]

Soln : sorting in ascending order

Inbuilt sorting : TC = O(NlogN)

output will always look like :

00.... 11.... 22.... 33 ..... 88.... 99....

freq(0) freq(1) ...

A = [9 8 9 7 1 1 9 0 0 2 9 1 3 7]
(0 1 2 3 4 5 6 7 8 9 10 11 12 13)

f = [2 3 1 1 0 0 0 2 1 4]
(0 1 2 3 4 5 6 7 8 9)

f(i) → freq of i(0-9) in the array

f = [2 3 1 1 0 0 0 2 1 4]
(0 1 2 3 4 5 6 7 8 9)

↳ 0 0 1 1 1 2 3 7 7 8 9 9 9 9

Count Sort

```
fi, f(i) = 0  // len = 10

for (i=0 to n-1) {
    f[A(i)]++
}
```

calculating frequency

N iterations

O(N)

```
for (d = 0 to 9) {
    for( i = 1 to f(d)) {
        print(d)
    }
}
```

sort array using frequency

iteration → 10×N

or

N ✓

O(N)

d	i	iteration
0		f(0) +
1		f(1) +
2		⋮
⋮		⋮
9		f(9) +

= f(0) + f(1) + ... + f(9) = N

total TC = $O(N)$

SC = $O(10) = O(1)$

What if $0 <= A[i] < 10^9$ → Is counting sort possible?

length of freq array $= 10^9$

int array

4B

$4B \times 10^3 \times 10^3 \times 10^3$ (10^9)

~4KB

~4MB

~4GB (Array space)

[$10^6 - 10^7$]

↓

Memory Limit Exceeded error (MLE)

What if $-9 <= A[i] <= 9$ → Is counting sort possible?

len = [-9, 9] = 19 Yes ✓

A = [-2 3 8 -4 -2 3 0]
(indices 0 1 2 3 4 5 6)

f = [1 | 2 | 1 | 2 | 1]
(indices 0 1 2 ... 6 | 7 8 9 10 | 11 12 ... 16 | 17 18)
-9 -8 -7 | -3 -2 -1 0 | 1 | 2 3 | 7 | 8 | 9

```
// len = 19
for (i=0 to n-1) {
    ~~f[A[i]]++~~ f[A[i]+9]++   // f[A[i] - minElement]
}                               // A[i]-(-9) = A[i]+9

for (d = 0 to (len-1)) {
    for( i = 1 to f[d]) {
        ~~print(d)~~ print(d-9)   // print(d + minElement)
    }                             // d+(-9) = d-9
}
```

in this case minElement = -9

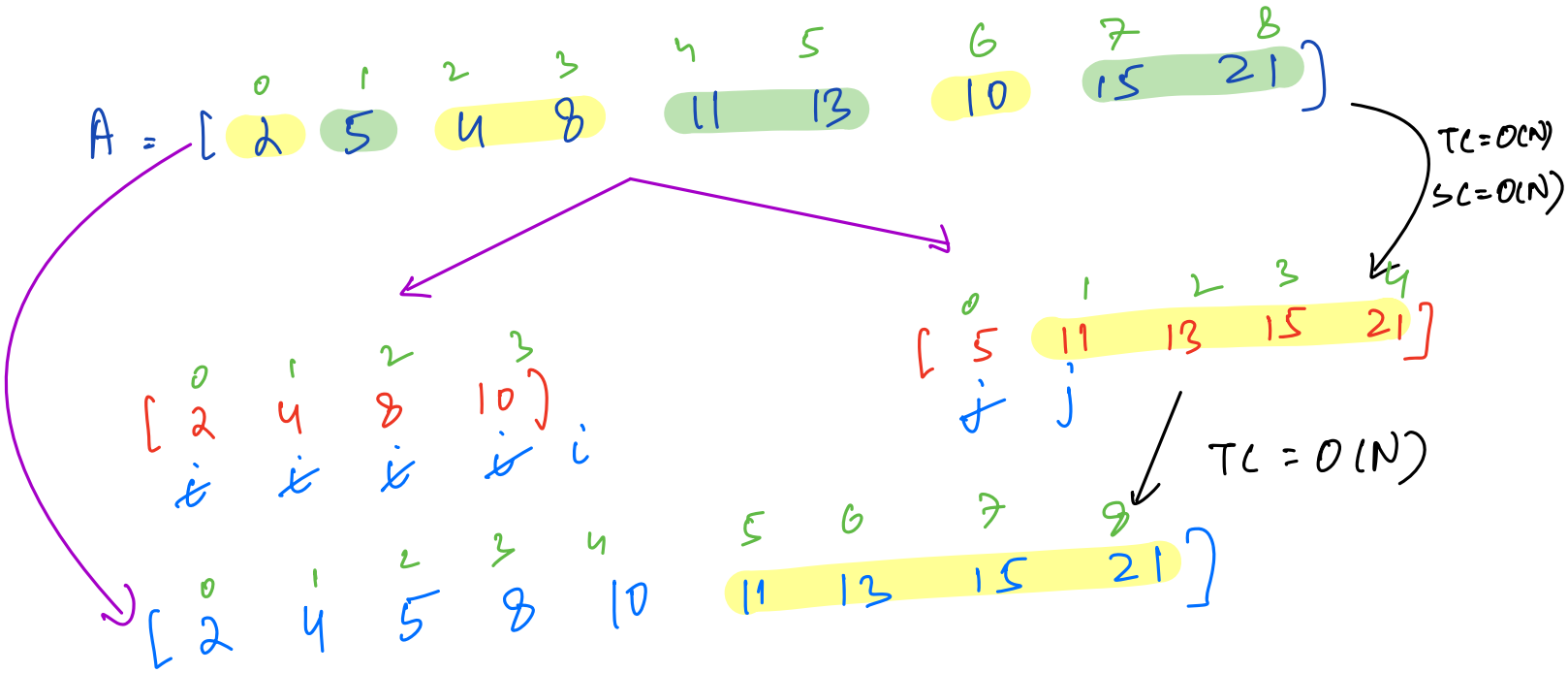
TC = O(N+N) = O(N)

SC = O(range of array)

(max(A[i]) - min(A[i])) + 1) <= 10^6

use countsort

Question Given an integer array where all odd elements are sorted & all even elements are sorted, sort the array.



Merge 2 sorted array of size N & M

$TC = O(N+M)$

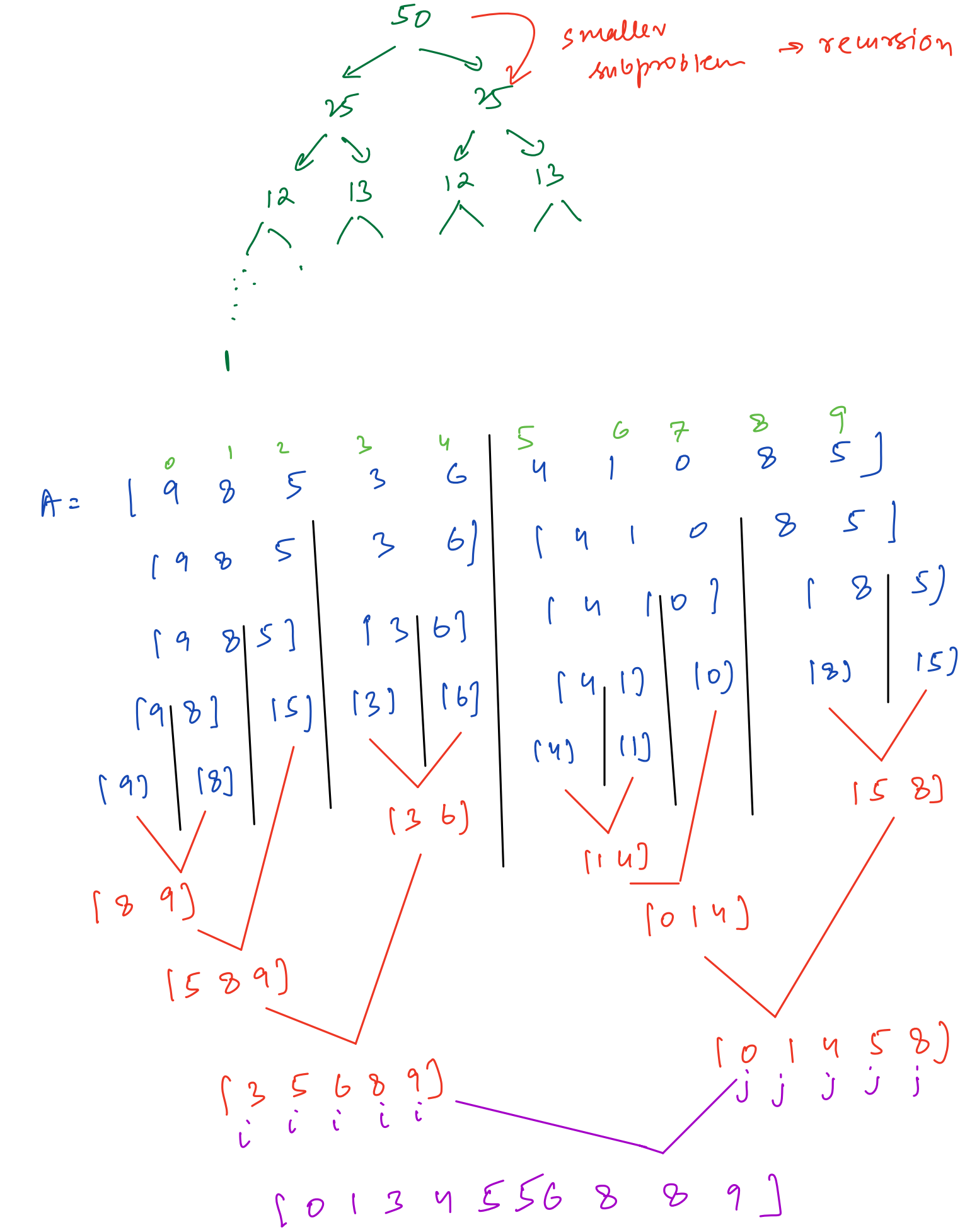
$SC = O(N+M)$

```
int[] merge( A[], N, B[], M) {
    ans [N+M]
    i=0, j=0, K=0
    while ( i<N && j<M ) {
        if (A[i] <= B[j]) {        → maintains stable sort
            ans[K] = A[i]
            i++
        }
        else {
            ans[K] = B[j]
            j++
        }
        K++
    }
    while ( i<N ) {
        ans[K] = A[i]
        K++, i++
    }
    while ( j<M ) {
        ans[K] = B[j]
        K++, j++
    }
    return ans
}
```

$TC = O(N+M)$

$SC = O(N+M)$

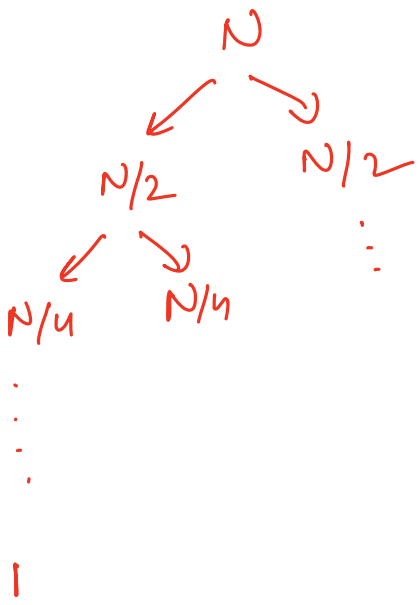
Merge sort


50
25
25
12
13
12
13
smaller subproblem
→ recursion
A =
0 1 2 3 4 5 6 7 8 9
9 8 5 3 6 4 1 0 8 5
[9 8 5 3 6]
[4 1 0 8 5]
[9 8 5]
[3 6]
[4 1 0]
[8 5]
[9 8]
[5]
[3]
[6]
[4 1]
[0]
[8]
[5]
[9]
[8]
[4]
[1]
[8 9]
[5 8 9]
[3 6]
[1 4]
[0 1 4]
[3 5 6 8 9]
[0 1 4 5 8]
i i i i i
j j j j j
[0 1 3 4 5 5 6 8 8 9]

```
void mergesort(A[], l, r) {
    if(l>=r) return
    mid = (l+r)/2
    mergesort(A, l, mid)
    mergesort(A, mid+1, r)
    merge(A, l, mid, r)
}
```

(l: 0, r: N-1)

merge(A, l, mid, r) → TC = O(N), SC = O(N)



levels = $\log_2(N)$

TC per level = O(N)

total TC = $O(N \log N)$

SC = $O(N + \log N) = O(N)$

because of recursion stack size

Question

Given an array, find count of inversion pairs.

inversion pair (i, j) ⇒ i<j && A[i] > A[j]

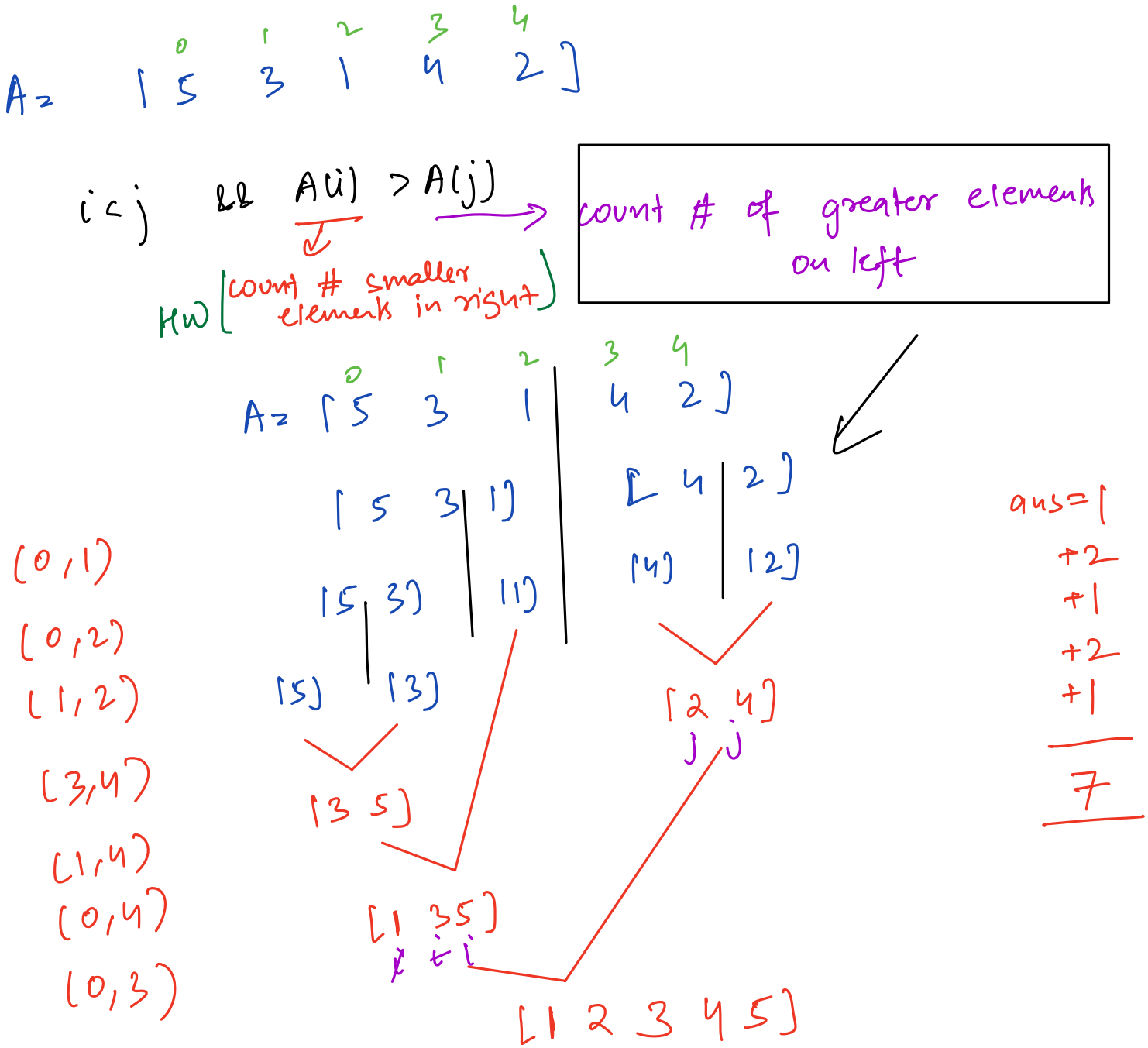
A = [10 3 8 15 6]
0 1 2 3 4

i < j		A[i] > A[j]	
0	1	10 > 3	✓
0	2	10 > 8	✓
0	3	10 > 15	x
0	4	10 > 6	✓
1	2	3 > 8	x
1	3	3 > 15	x
1	4	3 > 6	x
2	3	8 > 15	x
2	4	8 > 6	✓
3	4	15 > 6	✓

ans = 5

Bruteforce ⇒ TC = O(N²)

SC = O(1)



if selecting right side element while merging

ans += # remaining elements in left

TC = O(N log N)

SC = O(N)

0 1 2 3

[5 2 6 1]

ans = 4

(0,1) (1,3)

(0,3) (2,3)

0 1 2 3 4

[5 3 1 4 2]

(0,1) (3,4) (0,4)

(0,2) (1,4) (0,3)

(1,2)

Stable Sorting → Relative order of equal elements should not change while sorting w.r.t a parameter.

A = [6 5 3 5]

↳ [3 5 5 6]

Name	Marks
A	8
B	5
C	8
D	4
E	8

sort w.r.t marks →

D	4
B	5
A	8
C	8
E	8

Inplace Sorting

if no extra space is needed to sort, it is called Inplace sorting.

If $SC = O(1) \Rightarrow$ Inplace