Q → Find the **smallest number** that can be formed by **re-arranging the digits** of the given number in an array.

[# digits <= 10^5] [0 <= A[i] <= 9]

	0	1	2	3	4	5	6	7	8
A = [	5	6	3	5	4	2	0	8	5]
↳	0	2	3	4	5	5	5	6	8

A = [5 3 3 2 4 2] → sort in asc. order

↳ 2 2 3 3 4 5

Sorting Algo.

TC = O(N log (N))

SC = O(1)

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

→ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓

Freq. array F[i] = frequency of i

size = 10

	0	1	2	3	4	5	6	7	8	9
F = [	1	0	1	2	2	3	2	0	1	0]

```
for i → 0 to (N-1) {
    F[A[i]] ++
}
```

TC = O(N)

SC = O(A[i]) = O(10) = O(1)

L → R

```
for d → 0 to 9 {
    for i → 1 to F[d] {
        print(d)
    }
}
```

TC = O(N)

Count Sort TC = O(N) SC = O(A[i]) → O(1)

If A[i] <= 10^9, can we use count sort?

No, ∵ SC will be high ⇒ MLE error.

Usual array max length → 10^6 to 10^7

If -5 <= A[i] <= 5, can we use count sort?

Yes

	0	1	2	3	4	5	6	7
A =	2	-3	0	2	-1	-5	-3	2

Range = [min max] → (max - min + 1)

5 - (-5) + 1 = 11

	-5	-4	-3	-2	-1	0	1	2	3	4	5
	0	1	2	3	4	5	6	7	8	9	10
F =	1	0	2	0	1	1	0	3	0	0	0

```
for i → 0 to (N-1) {
    F[A[i] - minA] ++
}
```

A[i] = -3 → index = 2

smallest → 0

A[i] - smallest → index in F[]

-5

A[i] - (-5) = A[i] + 5

```
for x → minA to maxA {
    for i → 1 to F[x - minA] {
        print(x)
    }
}
```

TC = O(N) SC = O(A[i])

Q → Given an integer array where all odd elements are sorted & all even elements are sorted. Sort the array.

	0	1	2	3	4	5	6	7	8
A = [	2	5	4	8	11	13	10	15	21]

Sorting Algo → TC = O(N log(N)) → Find sol. with TC < O(N log(N))

	0	1	2	3	4	5	6	7	8
A = [	2	5	4	8	11	13	10	15	21]

[2 4 8 10] [5 11 13 15 21]

	0	1	2	3	4	5	6	7	8
A = [	2	4	5	8	10	11	13	15	21]

Steps → 1) Divide the array into 2 parts, one containing all even elements & other containing all odd elements. → TC = O(N)

SC = O(N)

2) Merge two sorted arrays into one.

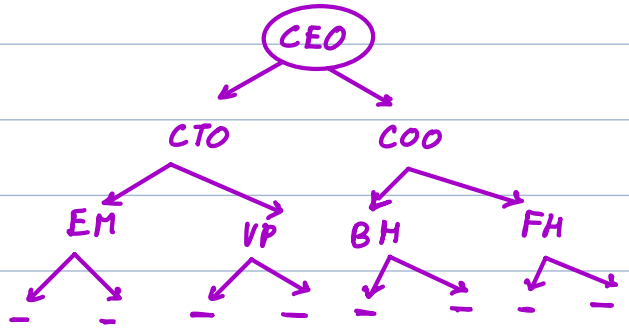
A[N], B[M] C[N+M]

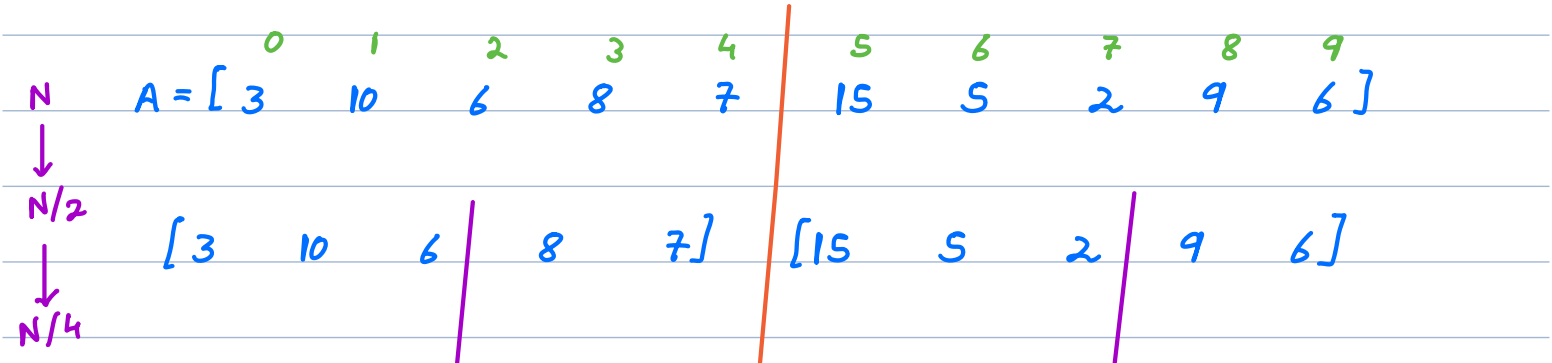
```
i = 0 , j = 0 , k = 0
while (i < N && j < M) {
    if (A[i] <= B[j]) {
        C[k] = A[i]    i++    k++      // C[k++] = A[i++]
    }else {
        C[k] = B[j]    j++    k++
    }
}
while (i < N) {
    C[k] = A[i]    i++    k++
}
while (j < M) {
    C[k] = B[j]    j++    k++
}
return C                    TC = O(N+M)
```

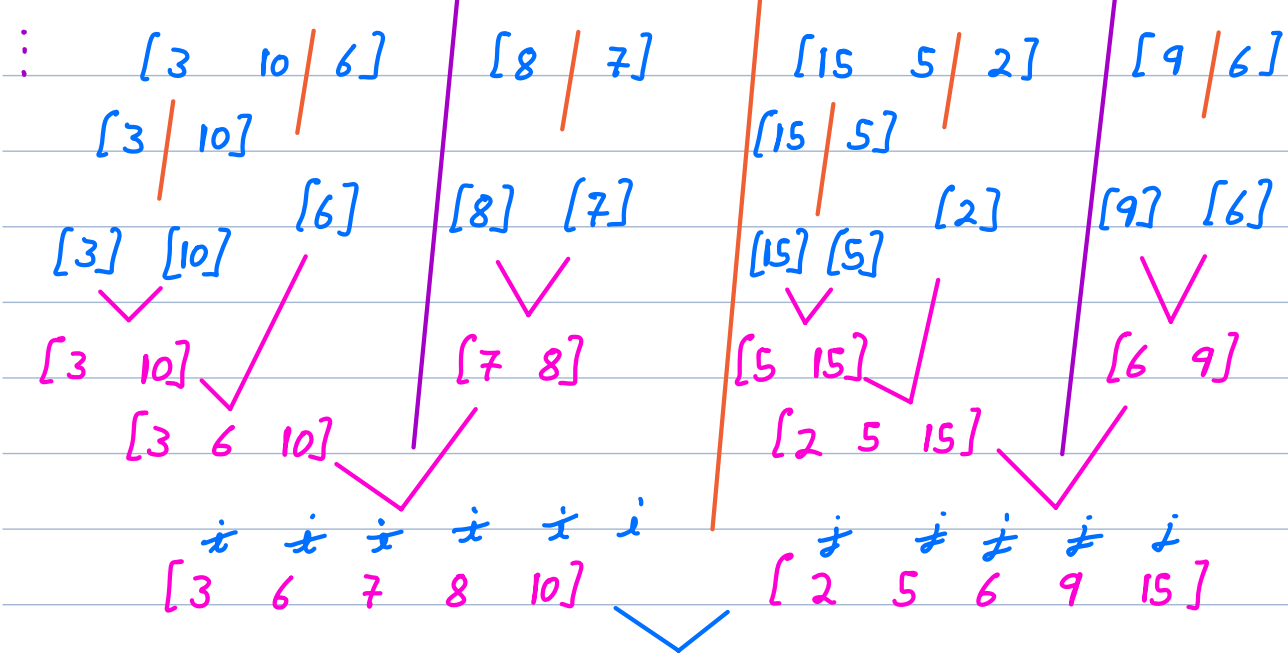
Overall → TC = O(N) SC = O(N)

Merge Sort

Divide & Conquer







A = [2 3 5 6 6 7 8 9 10 15]

```
void sort(A[], l, r) {        // l: 0, r: N-1
    if (l >= r) return
    mid = (l+r)/2
    sort(A, l, mid)
    sort(A, mid+1, r)
    merge(A, l, mid, r)
}
```

merge → TC = O(N) SC = O(N)

l — mid (mid+1) — r

TC = O(N log(N)) SC = O(N + log(N)) → O(N)

Q → Given 2 integer array A & B. Find the count of pairs (i,j) s.t A[i] > B[j].

A = [7 3 5] (0, 1, 2)

B = [2 0 6]

(7, 2) (7, 0) (7, 6)

(3, 2) (3, 0)

(5, 2) (5, 0)

Ans = 7

Bruteforce → ∀ i, j check A[i] > B[j]. TC = O(N*M)

SC = O(1)

Sol → Sort the array. ✓

A = [3 5 7] B = [0 2 6]

ans = 0 3 6 7 [0 2 3 5 6 7]

→ select B[j] ⇒ ans += # remaining elements in A

[i N-1] → N-1 - i + 1 ⇒ N-i

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

= O(N log(N) + M log(M)) SC = O(N+M)

Q → Find the # pairs (i, j) s.t i < j & A[i] > A[j].

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

i	j
0	1
0	2
0	4
2	4
3	4

Ans = 5

A = [5 2 6 1] (0 1 2 3)

i	j
0	1
0	3
1	3
2	3

Ans = 4

A = [5 3 1 4 2] (0 1 2 3 4)

i → 0 0 0 0 1 1 3

j → 1 2 3 4 2 4 4

Ans = 7

ans = 0

Bruteforce → for i → 0 to (N-2) {

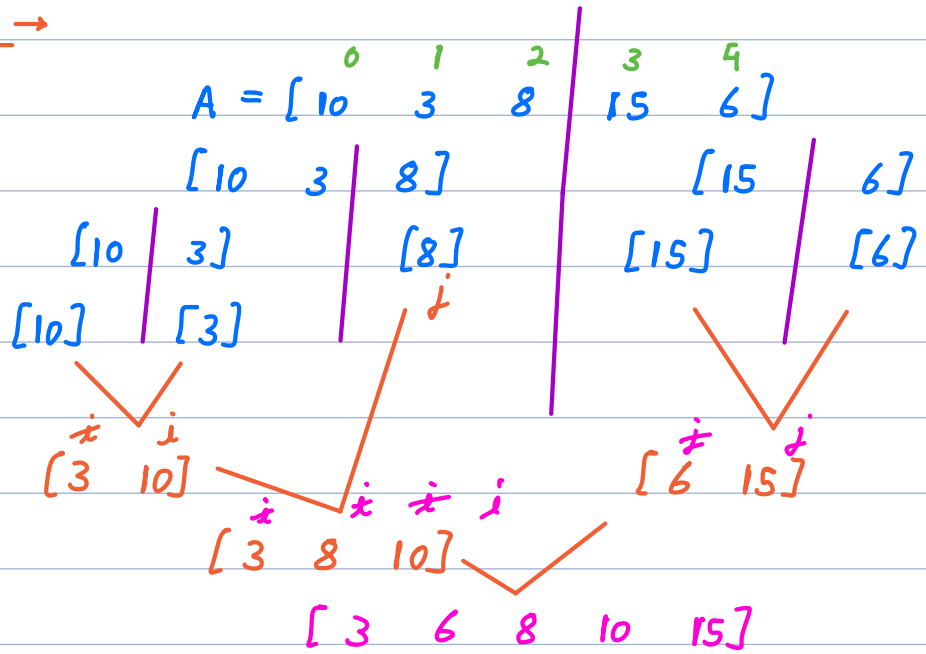
for j → (i+1) to (N-1) {

if (A[i] > A[j]) ans++

}

} return ans TC = O(N^2) SC = O(1)

Sol →



(10, 3) (8, 6)

(10, 8) (10, 6)

(15, 6)

TC = O(N log(N)) SC = O(N)

select from right half ⇒ ans += # remaining elements in left

l — mid mid - i

Stable Sort

while sorting relative order of equal elements should not change.

A = [8 7_1 10 5 7_2 9]

Sort → 5 7_1 7_2 8 9 10

Relative order of equal data becomes critical if original data has a defined order.

Ensure stability in code → For equal elements use index to compare.