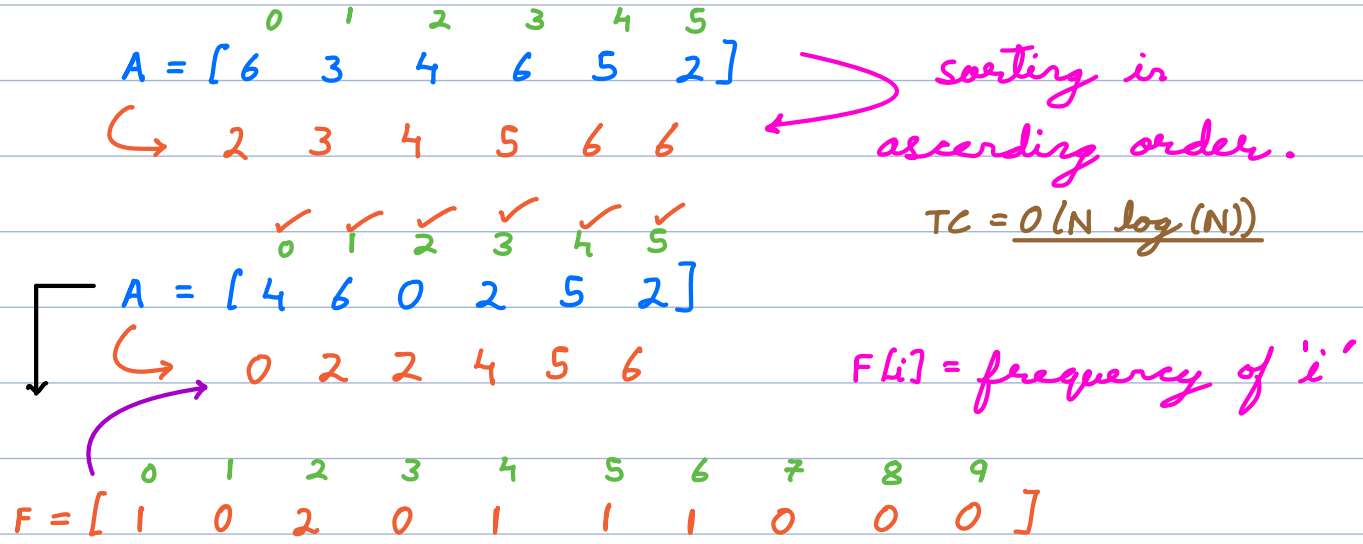
Q → Find the smallest number that can be formed by rearranging the digits in the given array.

$0 <= A[i] <= 9$




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

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

Count Sort

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

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

If $A[i] <= 10^9$ ⇒ MLE error (SC is very high)

Works for range ~ 10^6 to 10^7

Sort array where $-5 <= A[i] <= 5$.

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

x →	-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	0	0	2	1	0	1	2	1	0

index = x - min

o/p → -5 -1 -1 0 2 3 3 4

= x - (-5) = x + 5

```
// find minimum of A[] → m
for i → 0 to (N-1) {
    F[A[i] - m] ++
}

for i → -5 to 5 { // Range
    id = i - m
    for j → 1 to F[id] { // # times an element is present
        print (i)        // for (i = 1; i <= F[id]; i++)
    }
}
```

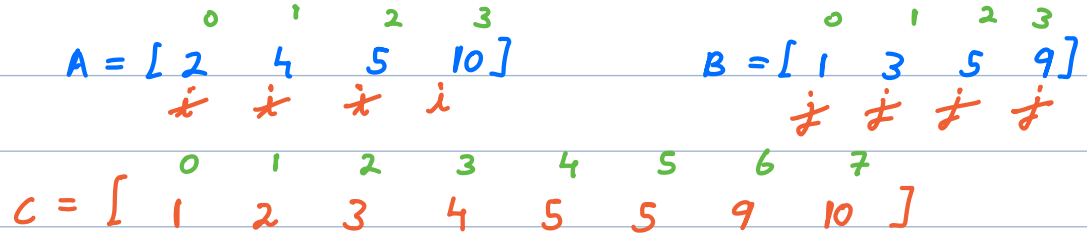
TC = O(N) SC = O(|A[i]|)

Q → Merge two sorted arrays into a single sorted array.

	0	1	2	3
A =	2	4	5	10

	0	1	2	3
B =	1	3	5	9

o/p → 1 2 3 4 5 5 9 10

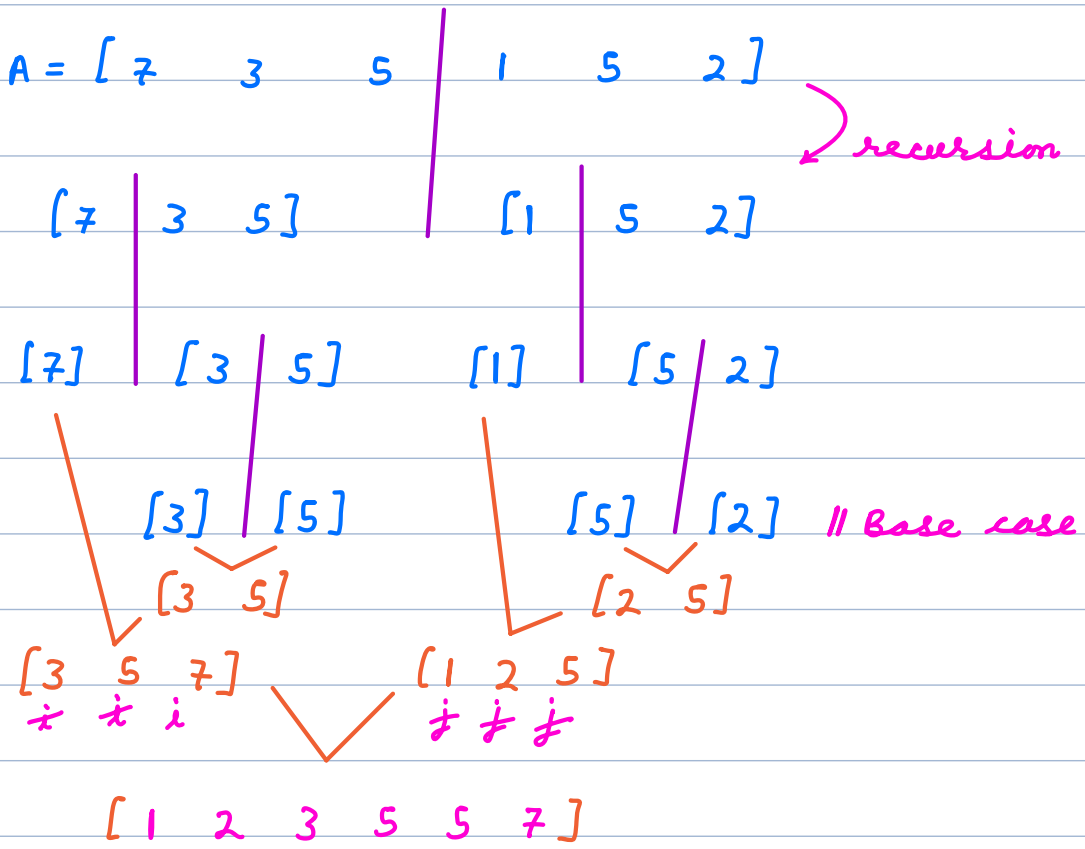



```
int[] merge (A[], B[]) {
    N = A.length        M = B.length
    // C[N+M] → Merged array
    i = 0      j = 0      k = 0
    while (i < N && j < M) {
        if (A[i] <= B[j]) {
            C[k] = A[i]
            i++    k++
        } else {                    // select from right half
            C[k] = B[j]             // cnt += N - i
            j++    k++
        }
    }

    while (i < N) {
        C[k] = A[i]
        i++    k++
    }

    while (j < M) {
        C[k] = B[j]
        j++    k++
    }

    return C
}
```

A[l — mid] A[mid+1 — r]

TC = O(N+M) SC = O(1) A, B → i/p C → o/p

Merge Sort (Divide & Conquer)




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

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

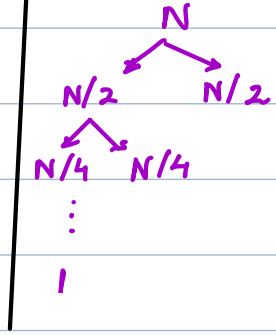
TC = O(N * # levels) ⇒ divide from mid

levels = log (N)

TC = O(N * log (N))

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

merge recursion



Q → Given two **sorted** arrays A[] & B[].
Find # pairs i,j s.t A[i] > B[j].

```
      0  1  2
A = [ 3  5  7 ]
B = [ 0  2  6 ]
```

Ans = 7

i	j
0	0
0	1
1	0
1	1
2	0
2	1
2	2

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

```
      ✓  ✓  ✓
      0  1  2  3
A = [ 1  3  5  7 ]
      ↑  ↑  ↑  i

      ✓  ✓  ✓
      0  1  2
B = [ 0  2  6 ]
      ↑  ↑  ↑
```

cnt = 0 + 4 + 3 + 1
= 8

when selecting B[j]
Ans += # remaining elements in A[]
= A.length − i

```
i = 0   j = 0
while ( i < N && j < M) {
    if (A[i] <= B[j])   i++
    else {  j++
            cnt += N-i
        }
} return cnt
```

TC = O(N + M) SC = O(1)

Q → Given an integer array A, find # pairs s.t i < j & A[i] > A[j].

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

Ans = 5

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

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

Ans = 4

i	j
0	1
0	3
1	3
2	3

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

Ans = 7

Bruteforce → $\forall i, j$ s.t $i<j$ check $A[i] > A[j]$.

TC = $O(N^2)$ SC = $O(1)$

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

[5 3 1] [4 2]

[5 3] [1] [4] [2]

[5] [3] [2 4]

[3 5] [1 3 5]

[1 2 3 4 5]

cnt += # of remaining elements in left half when selecting element from right half.

cnt = 0 + 1 + 2 + 1 + 2 + 1 = 7

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

SC = $O(N)$

Stable Sorting → Relative order of equal elements should not change while sorting.

For equal elements → use index to compare.