Sorting 1: Count Sort & Merge Sort

COUNT SORT

Question :- Find the smallest number that can be formed by rearranging the digits of the given number in an array. Return the smallest number in the form of array.

Note:- Given array will contain Single Digits only.

Ex1

A[] = { 6, 3, 4, 2, 7, 2, 1 }

Ans = { 1, 2, 2, 3, 4, 6, 7 }

Ex2

A[] = { 4, 2, 7, 3, 9, 0 }

Ans = { 0 2 3 4 7 9 }

Brute force

↳ Sort the array

TC → O(N log N)

OPTIMIZATION

↳ we can form freq array of Range 0 – 9

Ex A[] = { 6, 3, 4, 2, 7, 2, 1 }

freq = { 0 1 2 2 1 1 0 1 1 0 0 }

0 1 2 3 4 5 6 7 8 9

iterate & form ans

ans = { 1, 2, 2, 3, 4, 6, 7 }

PSEUDO CODE

```
int freq[10];
for ( i -> 0 to (n-1)) {
        int val = arr[i];
        freq[val] ++;
}

for ( i -> 0 to 9) {
        for ( j -> 1 to freq[i]) {
                Print (i);
        }
}
```

TC → O(N + N)

≒ O(N)

SC → O(1)

Q Will Count Sort work if the range of A[i] is more than 10^9 ?

$A = \{ 2, 3, 10^9, 10^7, 999 \}$

↳ So, NO

↑ Max [So, we need to create freq array of range $[0 - (10^9)]$

Explanation

Integer → 4 Bytes

freq $[10^9]$ → Space $[4 \times 10^9]$

≈ 4GB

Since 4 GB required, Hence Not possible.

Ily freq $[10^6] = 10^6 * 4$

≈ 4 MB.

⇒ COUNT SORT ON -ve NUMBERS

Implement Count Sort for an array Containing -ve nos.

Eg :- $A = [-2, 3, 8, 3, -2, 3]$

Here, Smallest value = -2

Largest value = 8

Range = $8 - (-2) + 1$

$= 8 + 2 + 1$

$= 11$

Let, Create freq array

freq : { 2 0 0 0 0 3 2 0 0 0 1 }

0 1 2 3 4 5 6 7 8 9 10

if, val = arr[i] // = 2

freq [val - Smallest Value]

index of arr[i] in freq array.

PSEUDO CODE

Smallest, Largest] → TODO

freq [Largest - Smallest + 1];

for (i → 0 to (n-1)) {

    f[A[i] - Smallest]++;

}

for (i → 0 — (freq.length - 1)) {

    for (j → 1 to freq[i]) {

        print (i + Smallest);

    }

}

TC → O (max (N, Range))

SC → O (Range)

Question 2 :- Given an integer array where all odd elements are sorted & all even elements are sort. Sort the entire array.

Ex :- A[] = {2, 5, 4, 8, 11, 13, 10, 15, 21}

Ans = {2 4, 5, 8, 10, 11, 13, 15, 21}

Brute force

↳ Sort the entire array.

TC → O(N log N)

↳ OPTIMIZATION

Ex :- A[] = {2, 5, 4, 8, 11, 13, 10, 15, 21}

odd Count → 5

even Count → 4

Create Seperate Array →

o: | 5 | 11 | 13 | 15 | 21 |

e: | 2 | 4 | 8 | 10 |

CQ → How to use the two seperate array to form the final array ?

odd → | 5 | 11 | 13 | 15 | 21 | (o)

even → | 2 | 4 | 8 | 10 | (e)

Ans → | 2 | 4 | 5 | 8 | 10 | 11 | 13 | 15 | 21 | (i)

PSEUDO CODE

```
void merge (A[]){
    int N = A.length;
    int odd[n1];        // count of odd
    int even[n2];       // count of event   //TODO
    int e=0, o=0;
    for(i -> 0 to (N-1)){
        if (A[i]%2==0){
            even[e] = A[i];
            e++;
        } else {
            odd[o] = A[i];
            o++;
        }
    }

    e=0, o=0, i=0
    while(e<n2 && o<n1){
        if (odd[o] > even[e]){
            A[i] = even[e];
            e++;
        } else {
            A[i] = odd[o];
            o++;
        }
        i++;
    }
```

```
while (e < n2) {
    A[i] = even[e];
    e++; i++;
}

while (o < n1) {
    A[i] = odd[o];
    o++, i++;
}
```

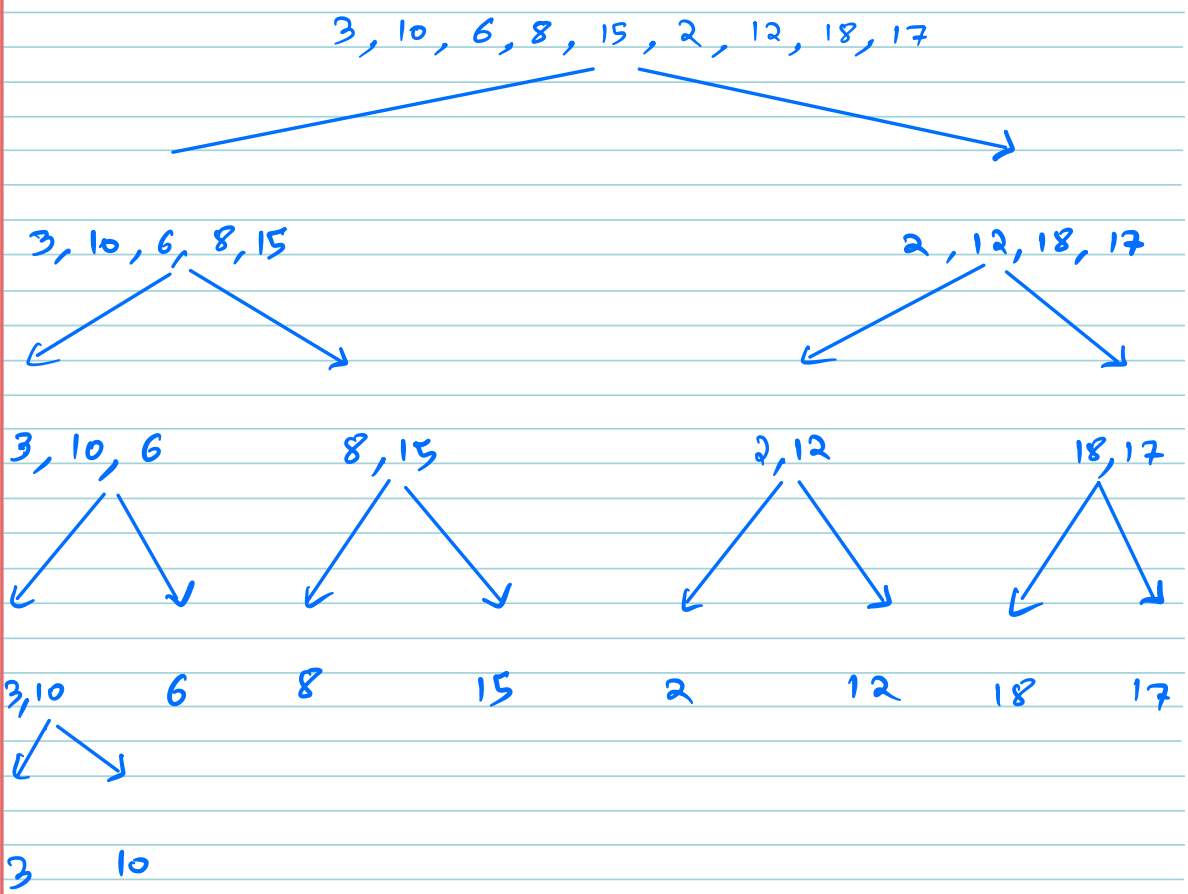
Quiz :- Iteration for Merging 2 arrays ?

↳ 2 * N

TC → O(N)

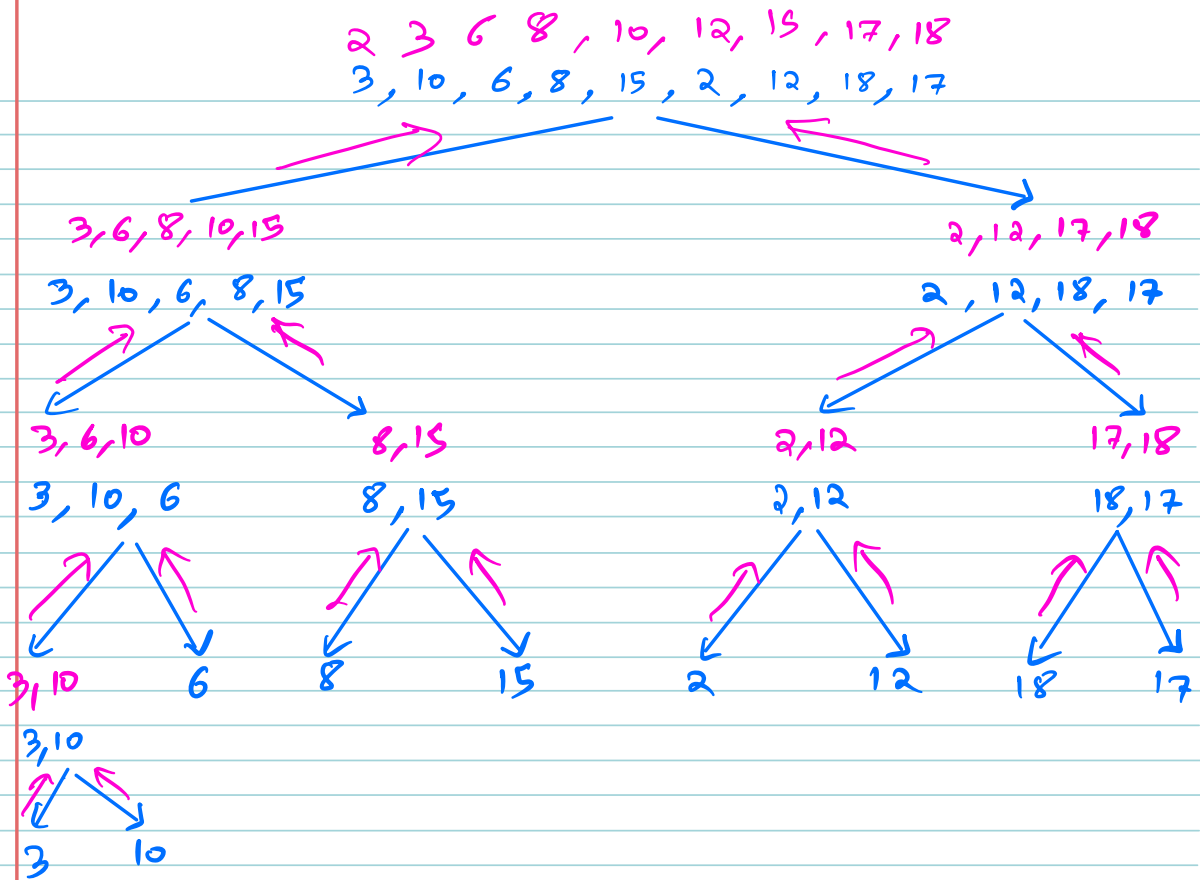
SC → O(N)

Merge Sort



★ Single element which we get after splitting will always be sorted.

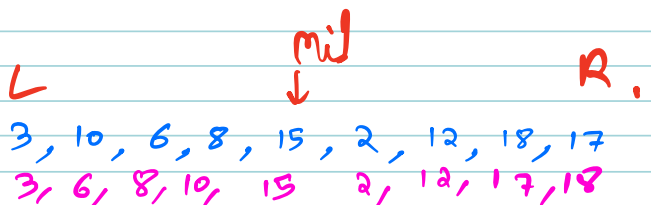
★ Can we apply the concept of merge from single elements.



Q By looking at Diag what implementation technique is coming out intuitively ?

↳ Recursion

- → We get Problem
- → Divided into sub-problems.
- → Use solution of sub-problems to build find solution.



2 3 6 8, 10, 12, 15, 17, 18

PSEUDO CODE

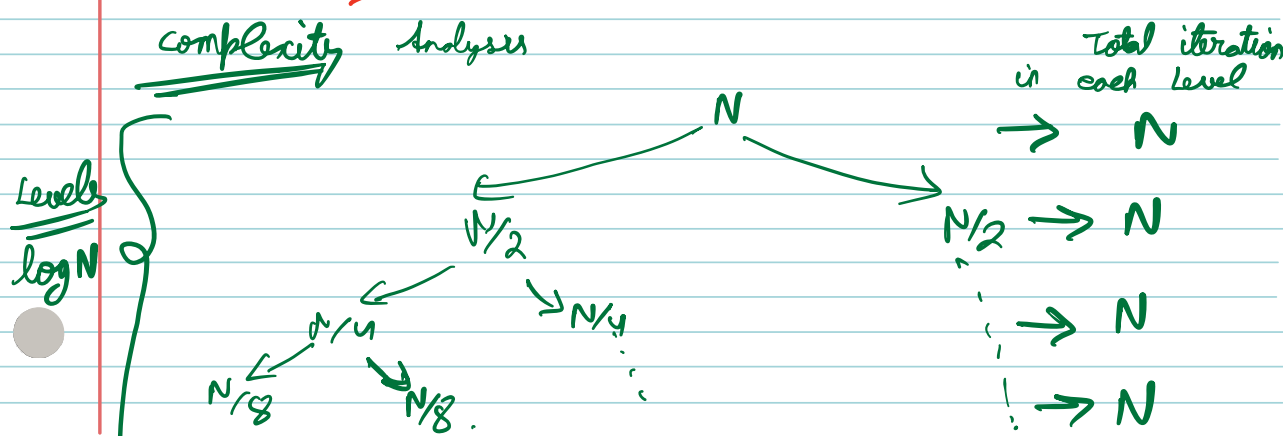
```
void mergeSort (A, L, R){
    if (L == R) { return; }
    int mid = (L+R)/2;
    mergeSort (A, L, mid);
    MergeSort (A, mid+1, R);
    Merge (A, L, mid, R);
}

void merge (A, L, mid, R){
    int N = R-L+1;
    int N1 = mid-L+1;
    int N2 = R-mid;

    Left[N1], Right[N2];
    for (i -> 0 to (N1-1)){
        Left[i] = A[i];
    }
    int ind = 0
    for (i -> (mid+1) to (N-1)){
        Right[idx] = A[i];
        idx++
    }
```

```
idx = 0, i = 0, j = 0
while (i < N1 && j < N2) {
    if (Left[i] < Right[j]) {
        A[idx] = Left[i];
        i++;
    } else {
        A[idx] = Right[j];
        j++;
    }
    idx++;
}
while (i < N1) {
    A[idx] = Left[i];
    idx++; i++;
}
while (j < N2) {
    A[idx] = Right[j];
    idx++; j++;
}
}
```

Complexity Analysis



TC → $O(N \log N)$

N → merging, $\log N$ → levels.

Recursive Way

Relation => $T(N) = 2T(\frac{N}{2}) + N$

SC → $O(\log N + N) \approx O(N)$

$\log N$ → Recursion stack, N → merging

Question 4 :- Given two array, A[n] & B[m]

Calculate number of pairs i, j such that A[i] > B[j].

Ex :- A[3] = {7, 3, 5}

B[3] = {2, 0, 6}

Ans → 7 (7,2) (7,0) (7,6) (3,2) (3,0) (5,2) (5,0)

Brute force

↳ 2 Nested loop & form all pairs & check.

TC → $O(N * M)$ | SC → $O(1)$

+3+3+1

↳ OPTIMIZATION

A[3] = { 3 5 7 } (i → 7)

B[3] = { 0 2 6 } (j →)

APPROACH :- ① Sort the 2 arrays.

② if (A[i] > B[j]), then all the remaining elements in A from i'th index will be greater than B[j].

[i, j] ↳ (3,0) (5,0) (7,0)

then j ++ ; & retry.

③ if (A[i] < B[j]), i++ & repeating step 2.

TC → O(N log N + M log M + N+M)

- N log N ↑ Sorting
- M log M ↑ Sorting
- N+M ↓ Counting Pairs

question :- Given an A[n], calculate no of pairs [i,j] such that $i < j$ && $A[i] > A[j]$, i & j are index of array.

Ex :- A[5] = { 10, 3, 8, 15, 6 }

Ans => 5

i	<	j	A[i]	>	A[j]
0	,	1	A[0]	>	A[1]
0	,	2	A[0]	>	A[2]
0	,	4	A[0]	>	A[4]
2	,	4	A[2]	>	A[4]
3	,	4	A[3]	>	A[4]

Quiz 2 :- Consider the following array: [5, 2, 6, 1]. calculate the inversion count for this array.

↳ 4

Quiz 3 :- Consider the following array: [5, 3, 1, 4, 2]. calculate the inversion count for this array.

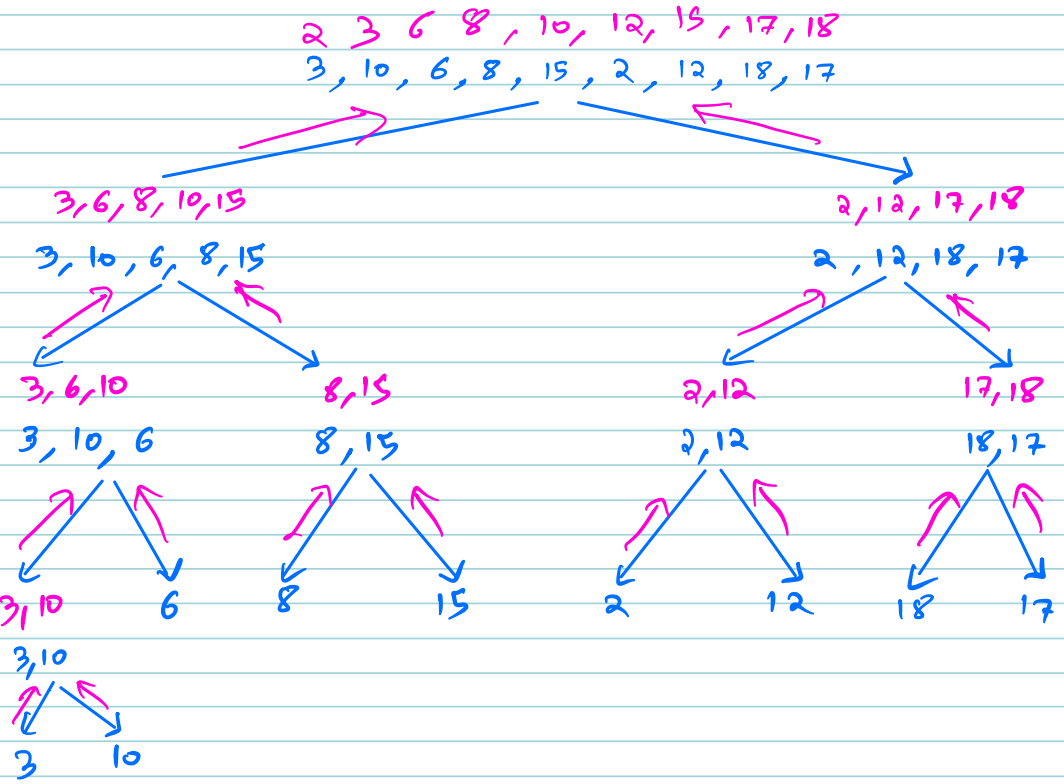
↳ 7

Brute force

↳ Create all the pair & check.

TC → O(N²) | SC → O(1)

↳ OPTIMIZATION



PSEUDO CODE

```
void mergeSort (A, L, R){
    if (L == R) { return; }
    int mid = (L+R)/2;
    mergeSort (A, L, mid);
    MergeSort (A, mid+1, R);
    Merge (A, L, mid, R);
}

void merge (A, L, mid, R){
    int N = R-L+1;
    int N1 = mid-L+1;
    int N2 = R-mid;

    Left[N1], Right[N2];
    for (i -> 0 to (N1-1)){
        Left[i] = A[i];
    }
    int ind = 0
    for (i -> (mid+1) to (N-1)){
        Right[idx] = A[i];
        idx++
    }
```

```
idx = 0, i = 0, j = 0
while (i < N1 && j < N2) {
    if (Left[i] <= Right[j]) {
        A[idx] = Left[i];
        i++;
    } else {
        A[idx] = Right[j];
        j++;
        Inv_Count += (Mid - i + 1);
    }
    idx++;
}
while (i < N1) {
    A[idx] = Left[i];
    idx++; i++;
}
while (j < N2) {
    A[idx] = Right[j];
    idx++; j++;
}
}
```

TC → O(N log N)

SC → O(N)

STABLE SORT

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

Ex:- arr[] = { 6, 5, 3, 5' }

↓ sorting

arr[] = { 3, 5, 5', 6 }

★ If any Sorting Algo maintains relative order of same elements then its stable sorting Algo.

Q → Is Merge Sort Stable?

↳ yes.

Q → why we look whether the Algo is stable sorting Algo or not?

IMPORTANCE

Eg:-

Airport	Normal Line	Priority Line
1	1	3
2	2	5
3 → P	4	6
4		
5 → P		
6 → P		