# Sorting :

## Today's Content:

- Count Sort
- Merge 2 sorted arrays
- Merge Sort
- Inversion Count
- Stable Sort.

**Question**: Find the smallest number that can be formed by rearranging the digits of the given number.

**ex**: N = 6342721 N = 427390

O/P = 1223467 O/P = 023479

**Idea 1**: Convert N into array and sort.

T.C = O(N log N) S.C = O(N)

**Optimised**:

1) Create a freq map (10 indexed array)
2) Go through each digit in N and count the frequency.
3) Go through 0-9 index in the array and print the element according to the frequency.

T.C = O(N) S.C = O(N)

```
func smallestNumber(int num) {
    arr[] // TODO - Number num to array
    int freq[10];
    for(int i=0; i<arr.length; i++){
        freq[arr[i]]++;
    }
    for (int i=0; i<10; i++){
        while (freq[i] >0){
            print(i);
            freq[i]--;
        }
    }
}
```

## COUNT SORT

num= 6342721

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

O/P = 1223467

# Disadvantages of Count Sort

[3, 2, 1 , 999]

{ 1 : 1
3 : 1
2 : 1
999 : 1 }

- Takes more time when range of each element is range
- In standard machines, only 10 MB is allocated.

int [$10^9$] → 4 B * $10^9$

= 4 GB (RAM)

You can't create a $10^9$ element array.

# Count Sort on Negative Numbers

A = [ -2 , 3 , 8 , 3 , -2 , 3 ]

Range = [-2, 8] → 8 - (-2) + 1 = 11 elements.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| freq = | ~~1~~2 | | | | | ~~1~~~~2~~3 | | | | | 1 |
| | -2 | -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

freq[arr[i] - smallest element].

$8 - x = 10$

$8 - 10 = x$

$x = \boxed{-2}$

```
func  smallestNumber (int num) {
      arr []  // TODO - Number num to array
      int  freq[Range];
      for (int  i=0;  i<arr.length ;  i++){
           freq [arr[i] - min_ele ]++ ;
      }
      for (int  i=0;  i<10;  i++){
           while ( freq[i] > 0){
                print(i + min_ele);
                freq[i]--;
           }
      } }
```

## COUNT SORT

**Question:** Merge two sorted arrays such that the result is also sorted.

ex: A = [1, 5, 6, 9]    B = [2, 4, 8]

O/P = [1, 2, 4, 5, 6, 8, 9]

## Brute Force Approach :

- Merge the arrays and sort.

  TC = $O(N \log N)$

## Optimised

i ↓ i ↓ ↓ i ↓    j ↓ j ↓ j ↓ j ↓

A = [1, 5, 6, 9]    B = [2, 4, 8]

k ↘

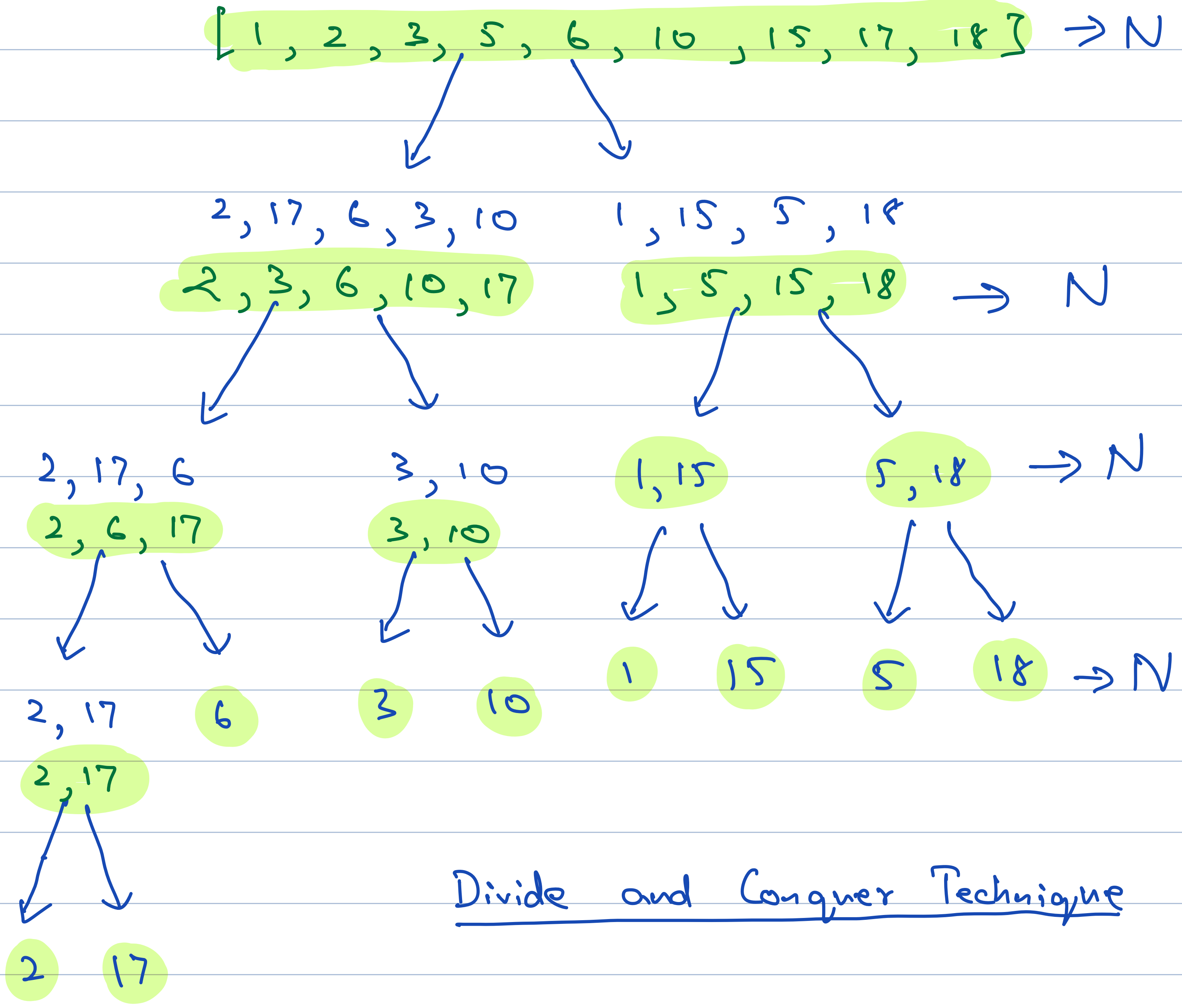
res = [1, 2, 4, 5, 6, 8, 9]

T.C = $O(N)$    S.C = $O(1)$

```
func merge (int[] A, int[] B) {
      i= 0 ; j= 0;   k=0;
      res[N+M];
      while (i<A.length && j< B.length){
            if (A[i] <= B[j]) {
                  res[k]= A[i];
                  i++ ; k++;
            }
            else {
                  res[k] = B[j];
                  j++ ; k++;
            }
      }
      while (i < A.length) {
            res[k] = A[i];
            i++ ; k++;
      }
      while (j < B.length) {
            res[k] = B[j];
            j++; k++;
      } return res; }
```

# Merge Sort

|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| arr = | 2 | 17 | 6 | 3 | 10 | 1 | 15 | 5 | 18 |



## Divide and Conquer Technique

$$N \Rightarrow \frac{N}{2} \rightarrow \frac{N}{4} \rightarrow \cdots\cdots 1 \Rightarrow \log N$$

$$T.C = O(N \log N)$$

```
func mergeSort (int[] arr){
    if (arr.length == 1) return arr;
    mid = N/2;
    A[N/2]; B[N/2];
    for (int i=0; i<=mid; i++){
        A[i] = arr[i];
    }
    k=0;
    for(int i=mid+1; i<N; i++){
        B[k] = arr[i];
        k++;
    }
    A = mergeSort(A);
    B = mergeSort(B);
    return merge(A, B);
}
```

S.C = Space at each recursion + No. of levels.

= $N + \log N$

S.C = $O(N)$

**Question:** Given two arrays A & B, find the number of pairs such that A[i] > B[j]

ex: A = [7, 3, 5] → N
B = [2, 0, 6] → M

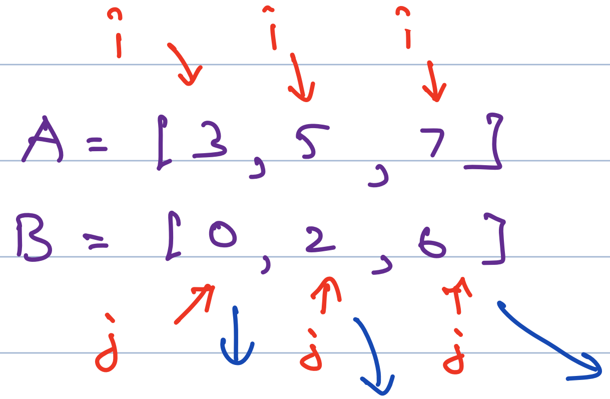
O/P = [(7, 2) (7, 0) (7, 6) (3, 2) (3, 0) (5, 2) (5, 0)] ⇒ 7

**Brute Force Approach**

- Take 2 loops and check if A[i] > B[j].

  T.C = O(N*M)  S.C = O(1)

**Optimised:**



A = [3, 5, 7]

B = [0, 2, 6]

3 pairs 3 pairs 1 pair. = 7 pairs.

Code - TODO  T.C = O(N log N + M log M) S.C = O(1)

**Question : Inversion Count** [INTERVIEW QUESTION]

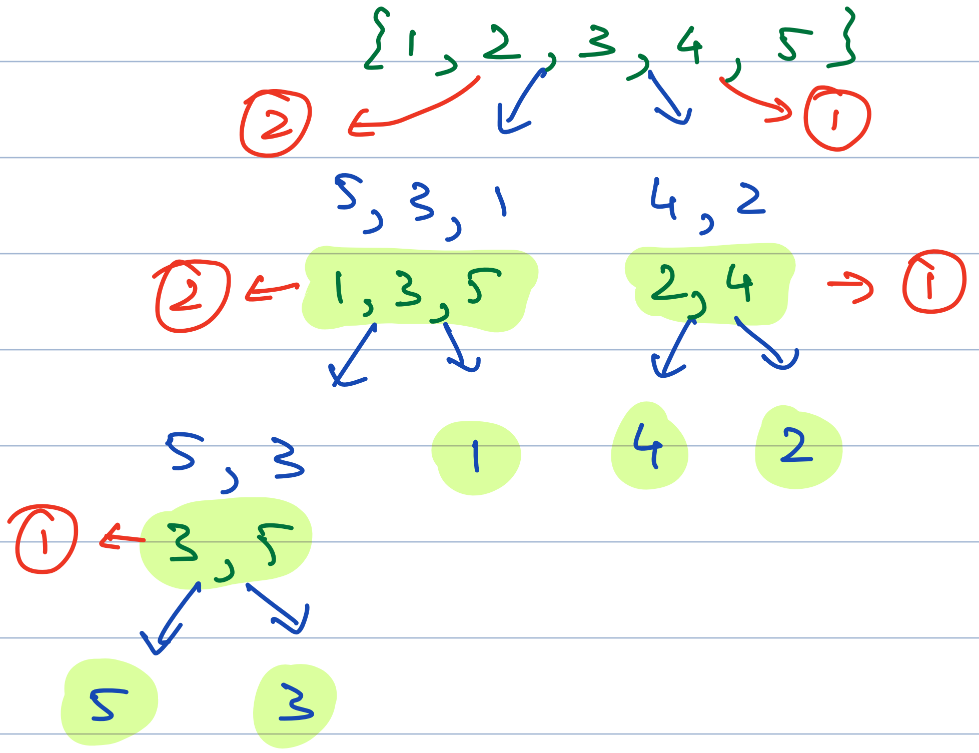
Given an array, find the number of pairs such that i<j and arr[i] > arr[j]

$$\begin{array}{ccccccc} & 0 & 1 & 2 & 3 & 4 & \\ \text{ex: arr} = \{ & 10, & 3, & 8, & 15, & 6 & \} \end{array}$$

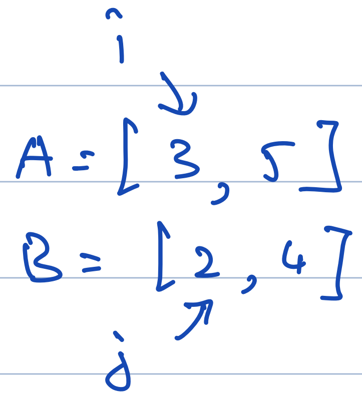
O/P = {(10, 3), (10, 8), (10, 6), (8, 6), (15, 6)}
= 5

arr = {5, 2, 6, 1}
O/P = {(5, 2), (5, 1) (2, 1) (6, 1)} ⇒ 4

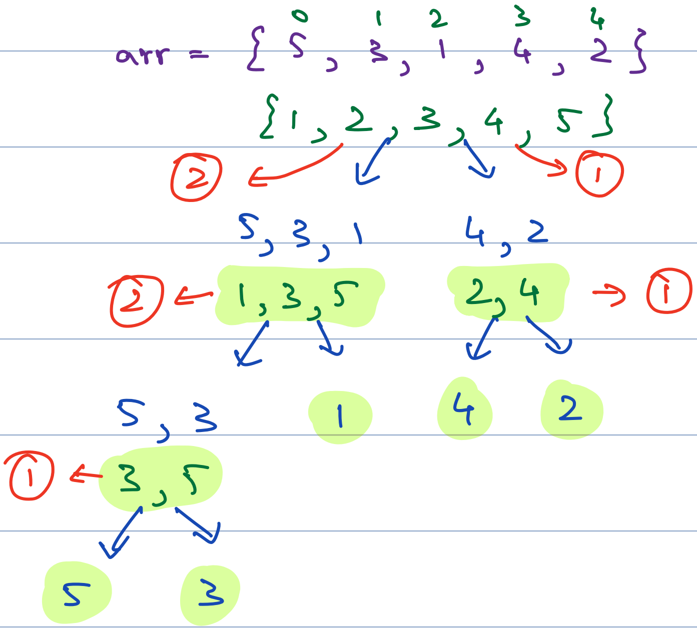
$$\begin{array}{ccccccc} & 0 & 1 & 2 & 3 & 4 & \\ \text{arr} = \{ & 5, & 3, & 1, & 4, & 2 & \} \end{array}$$ ⇒ O/P = 7



```
func merge (int[] A, int[] B){
    i = 0 ; j = 0;  k=0;
    res [N+M];
    while (i<A.length && j< B.length){
        if (A[i] <= B[j]){
            res[k]= A[i];
            i++ ; k++;
        }
        else{
            res[k] = B[j];
            j++ ; k++;
            ans += A.length - i ;
        }
    }
    while ( i < A.length ) {
        res[k] = A[i] ;
        i++ ; k++;
    }
    while (j < B. length) {
        res[k] = B[j] ;
        j++ ; k++;
    } return res; }
```



```
func mergeSort (int[] arr){
    if (arr.length == 1) return arr;
    mid = N/2 ;
    A[N/2] ; B[N/2];
    for (int i=0 ; i<=mid ; i++){
        A[i] = arr[i];
    }
    k = 0 ;
    for(int i = mid+1 ; i<N ; i++ ){
        B[k] = arr[i];
        k++;
    }
    A = mergeSort(A);
    B = mergeSort(B);
    return merge(A, B);
}
int ans =0; //+1+2+1+2+1
↳ GLOBAL
```



# Stable Sort

- Relative order of equal elements should not change while sorting.

arr = { 1 , 5 , 3 , 5 }

ans = { 1, 3 , 5 , 5 { → Stable sort

ans = { 1, 3 , 5 , 5 { → Not stable.

| Airport | Normal | Priority |
|---|---|---|
| 1 | 1 | 4 |
| 2 | 2 | 7 |
| 3 | 3 | |
| 4 → Business | 5 | |
| 5 | 6 | |
| 6 | | |
| 7 → Business | | |

Amazon Prime → Bhargav

Non-Prime → Puneet, Aniket , Rishi

② ① ③

# Doubts

```
interface A {
}
interface B extends A {
}
interface C extends B {
}
```

```
interface/class extends A, B {
```

→ Not possible

→ Multiple Inheritance

→ Interface

```
class D implements A, B {
```

→ Possible.

$A = [\,1, \cdot - -$ 1000 Times, $5 \cdots$ 500 times,

$2 \ldots$ 2000 times, $4 - - - \cdot$ 100 times $]$

$A[i] \rightarrow [1, 5] \longrightarrow$ Range of the value.

$T.C = O(N) = 2^{*}3600$ iterations using Count Sort.

$T.C = O(N \log N) = 3600 \log 3600 = 36000$ iterations

↳ Using merge sort.

$A = [\,1, 500, 1000, 200, 10000\,]$

$A[i] \rightarrow [1, 10000]$

$T.C \rightarrow$ Using Count Sort $\rightarrow 5 + 10000$ iterations

$T.C \rightarrow$ Using merge sort $\rightarrow 5 \log 5 = 15$ iterations

$A = [\,3\,,\ 6\,,\ \boxed{15}\,,\ 16\,]$

① ② ② ① → Prime co-factors.

1) Get all primes.
2) Get SPF (Smallest prime factor) of all values till 16.
3) For every A[i] → (A[i] / SPF) → Find its SPF.
   ↳ Increment count if SPF is unique.
4) Maintain ans & count_ans. If cur_count >= count_ans → Update ans, count_ans
5) Return ans.