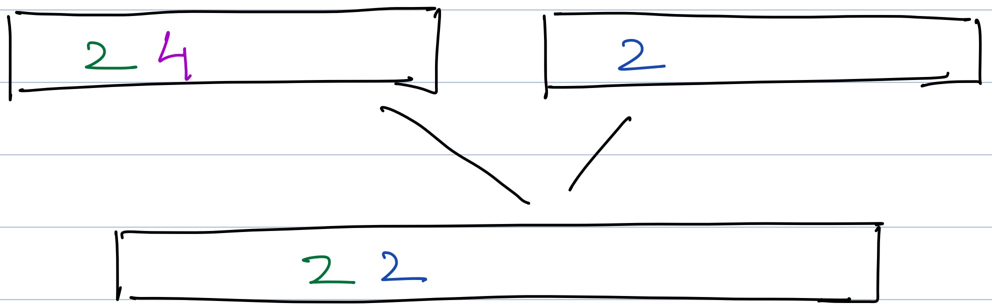
Is Merge Sort Stable ?



If both elements are equal choose from 1st subarray..

Merge Sort is stable

Merge Sort is not inplace

```
merge ( A, s, m, e) {

    int temp [ e-s+1]
    int p1 ⇒ s
    int p2 ⇒ m+1 , int p3 ⇒ 0.

    while ( P1 ≤ m && p2 ≤ e) {
                                        → comp(arr[P1],
                                               arr[P2])
        if (arr [P1] ≤ arr [P2]) {
            temp [P3] = arr [P1];
            P3++, P1++;
        } else {
            temp [P3] = arr [P2];
            P3++, P2++;
        }
    }
    while ( P1 ≤ m)
        temp[P3] = arr[P1]; P3++, P1++;
    while (P2 ≤ c)
        tem [P3] = arr [P2] ; P3++ P2++

    // Copy from temp
}       to array.
```

Comparator Basics

```
#    list<int> l

#    sort(l)
       ↓
     sort(l, comp);  → Define the comparison logic.

bool comp(int a, int b) {

   // return true if you want a first
   // return false if you want b first

}
```

Q1 Sort the array according to the number of factors each element has:

Ex1 : arr[] =

0	1	2	3	4	5	6
3	9	4	6	12	10	7
2	3	3	4	6	4	2

↓

3	7	9	4	6	10	12

```
bool comp (int a, int b) {

    int fact_a => no_of_factors (a);
    int fact_b => no_of_factors (b);

    if ( fact_a ≤ fact_b) {
            return true;
    } else
            return false

}
```

$$\boxed{\text{sort (arr, comp);}}$$

Q2 Given 2 Arrays A[N] & B[M], calculate no of pairs i,j, such that A[i] > B[j].

Ex1

$$A[3] = \begin{array}{|c|c|c|} \multicolumn{1}{c}{0} & \multicolumn{1}{c}{1} & \multicolumn{1}{c}{2} \\ \hline 7 & 3 & 5 \\ \hline \multicolumn{1}{c}{3} & \multicolumn{1}{c}{2} & \multicolumn{1}{c}{2} \end{array} \qquad B[3] = \begin{array}{|c|c|c|} \multicolumn{1}{c}{0} & \multicolumn{1}{c}{1} & \multicolumn{1}{c}{2} \\ \hline 2 & 0 & 6 \\ \hline \end{array}$$

$$\left[\begin{array}{l} (0,0),\ (0,1),\ (0,2) \\ (1,0),\ (1,1) \\ (2,0),\ (2,1) \end{array}\right] > \boxed{7}$$

Approach 1: Brute Force

Two loops ⇒ $O(n \times m)$.

Approach 2 :

1) Sort B array. ⇒ $O(m\log m)$

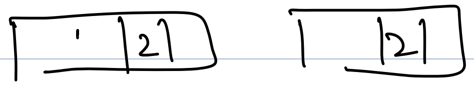
2) For each element in A, do binary search in B ⇒ $O(n\log m)$

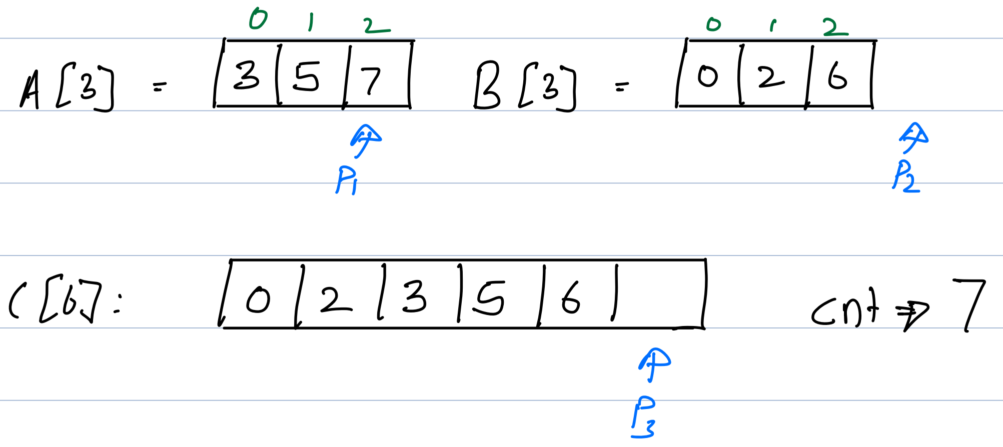
TC: $O(m\log m + n\log m)$

Approach 3

1) Sort A array

2) Sort B array





Case 1: When we pick from 2^{nd} array.

cnt ⇒ cnt + no of element left to be proceeded in A

↳ $n - P_1$

CASE2: When we pick from 1^{st} array.

cnt remains same

TC: $(n \log n + m \log m + n + m)$

Q3 Given A[N], find no of pairs i,j such that
i < j && A[i] > A[j].

$\boxed{n \Rightarrow 10^5}$

Ex1 : arr[] =

0	1	2	3	4	5	6	7	8	9
10	3	8	15	6	12	2	18	7	1
6	2	4	5	2	3	1	2	1	0

⇒ 26

Brute Force : TC: $O(n^2)$

SC: $O(1)$:

$\boxed{arr}$

0	1	2	3	4	5	6	7	8	9
10	3	8	15	6	12	2	18	7	1

10	3	8	15	6

A

12	2	18	7	1

B

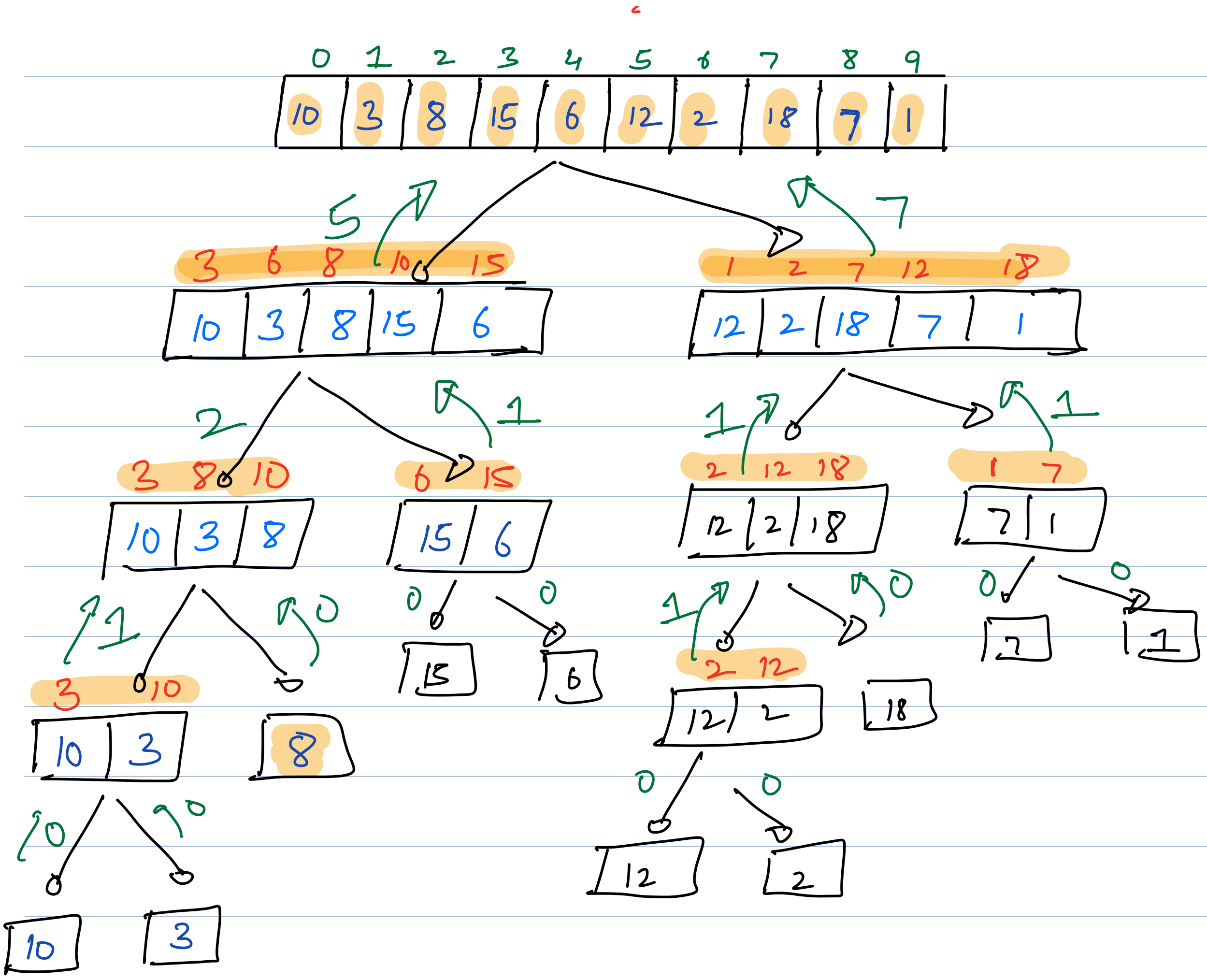
Pairs in arr ⇒ $\overset{5}{(\text{Pairs in A})}$ + $\overset{7}{(\text{Pairs in B})}$

+ $\underset{14}{(\text{Pairs b/w A \& B})}$

↳ during merge function.

C ⇒ $\boxed{0}$

→ $\boxed{26}$



0 1 2 3 4 5 6 7 8 9
10 3 8 15 6 12 2 18 7 1
5
3 6 8 10 15
10 3 8 15 6
7
1 2 7 12 18
12 2 18 7 1
2
3 8 10
10 3 8
1
6 15
15 6
1
2 12 18
12 2 18
1
1 7
7 1
1
3 10
10 3
0
8
0
15
0
6
1
2 12
12 2
0
18
0
7
0
1
0
10
0
3
0
12
0
2

10:40

Pseudo Code

```
int mergeSort ( int arr[], int s, int e)

    if (s==e)
        return 0;

    int mid => $\frac{(s+e)}{2}$

    int l => mergeSort (arr, s, mid);
    int r => mergeSort (arr, mid+1, e);

    return (l+r+ merge (arr, s, mid, e));

}
```

```
int  merge ( A, s, m ,e) {

    int  temp [ e-s+1]
    int  p1 = s
    int p2 = m+1 , int p3 = 0.        int c = 0;

    while ( p1 <= m && p2 <= e) {

          if  (arr [p1] <= arr [p2]) {
                temp [p3] = arr [p1];
                 p3++, p1++;
          } else {
                temp [p3] = arr [p2];
                p3++, p2++;   c = c + (m-p1+1);
          }
}
    while ( p1 <= m)
           temp[p3] = arr[p1]; p3++, p1++;
    while (p2 <= c)
           tem [p3] = arr [p2] ; p3++ p2++

  // Copy from temp
       to array.
    return c;
}
```

Q Sort the array.

arr[7] =

0	1	2	3	4	5	6
1	1	2	4	4	1	3

0	1	2	3	4	5
0	3	1	1	2	0

= O(n)

⇓

1 1 1 2 3 4 4 ⇒ O(n)

$0 \leq arr \leq 10^5$

int freq $[10^5+1]$

```
for (int i=0 ; i<n ; i++) {
        freq [arr[i]]++
}
```

O(n)

```
for (int i = 0; i<=k ; i++){
    int cnt => freq [i];
    for (int j=0; j<cnt ; j++) {
         print (i);
    }
}
```

O(n+k)

Tc : O(n+k)

Sc : O(k)

TC :

i	j	inner loop
0	cnt[0]	cnt[0]
1	cnt[1]	cnt[1]
⋮		
k	cnt[k]	cnt[k].

Inner loop ⇒ cnt[0] + cnt[1] + ... cnt[k]

⇒ n

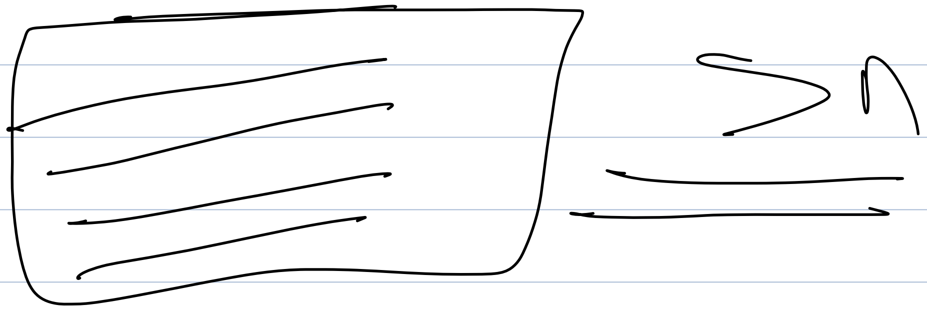
```
int j = 0;

for (i=0 ; i<k; i++) {

    while (j<100) {
        print ("hello");
        j++;
    }
}
```

i	j	inner
0	100	100
1	0	0
2	0	0
3	0	0
⋮	0	0
⋮	0	0
k	0	0

$\boxed{\text{Inner loops} \Rightarrow 100}$ ⇒ O(k+100)

Time ⇒ outer loop iteration + inner loops iteration.

for (int i=0; i<n ; i++) {



}

$0 \le arr[i] \le 10^5$ $k \Rightarrow 10^5$

$\boxed{n \Rightarrow 10} \Rightarrow \boxed{10}$

<u>Case 1</u>

$n \Rightarrow 10$

$k \Rightarrow 10^5$

n is insignificant

<u>Case 2</u>

$n = 10^5$

$k \Rightarrow 10$

$1 \le arr \le 10^{18}$ X $k = 10^{18}$

↳ no of distinct elements.

$\boxed{-10 \le arr \le 10}$

$k \Rightarrow 20$

$k < 10^6$

↳ Then only counting sorting is valid.

$-10 \Rightarrow 0$

$-9 \Rightarrow 1$

$-8 \Rightarrow 2$

⋮

$10 \Rightarrow 20$

$n \Rightarrow 100$
$k \Rightarrow 10^5$

$n \Rightarrow 10^5$
$k \Rightarrow 10^5$

⇒ Merge =

⇒ Count

$(n \log n) > (n+k)$ || $O(n) \ge O(k)$

TC

SC