Brute force

for merging arr1 of m size & arr2 of n size.
Create a third array of size (m+n)
Run pointers on both arrays
Copy the smaller element of the two, increment that pointer.

T.C → O(n+m) S.C → O(n+m)

Optimised Solution : (Based on Insertion sort)

Traverse through the first array & compare with second arrays first element

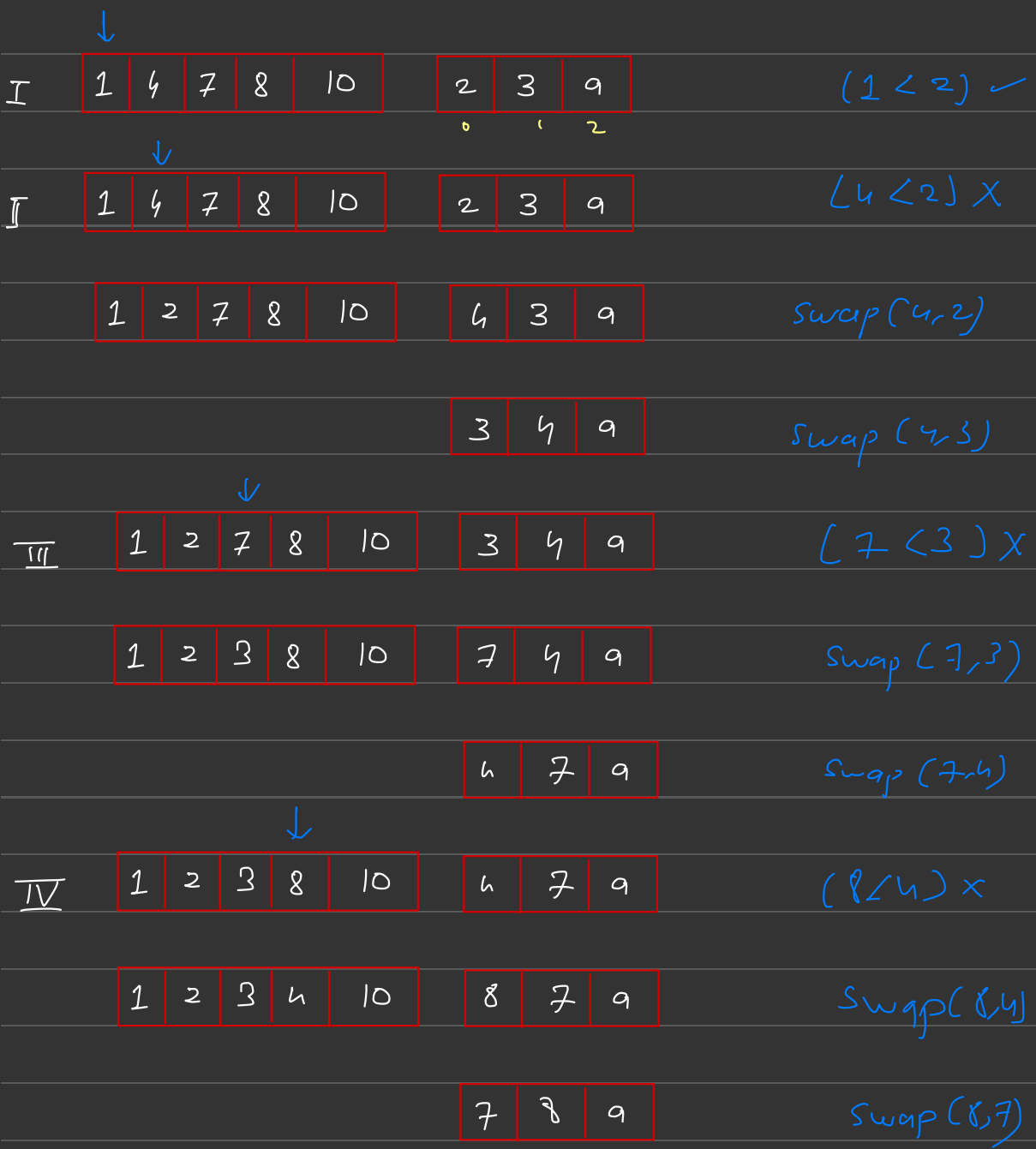
∵ arr2 is sorted, arr2[0] is the smallest element. If arr1[i] > arr2[0], arr1[i] is in wrong position

Hence swap them

```
if(arr1[i] > arr2[0]) {
    swap(arr1[i], arr2[0]);
```

After swap, make sure that arr2 is still sorted, by changing the position of newly swapped element.

```
int k = 0
while (k < n-1 && arr2[k] > arr2[k+1]) {
    swap(arr2[k], arr2[k+1]);
    k++; }
}
```


I
1 4 7 8 10
2 3 9
0 1 2
(1 < 2) ✓
II
1 4 7 8 10
2 3 9
(4 < 2) X
1 2 7 8 10
4 3 9
swap (4, 2)
3 4 9
swap (4, 3)
III
1 2 7 8 10
3 4 9
(7 < 3) X
1 2 3 8 10
7 4 9
swap (7, 3)
4 7 9
swap (7, 4)
IV
1 2 3 8 10
4 7 9
(8 < 4) X
1 2 3 4 10
8 7 9
swap (8, 4)
7 8 9
swap (8, 7)

arr 1 = | 1 | 2 | 3 | 4 | 7 |

arr 2 = | 8 | 9 | 10 |

T.C → O(n x m)

Traversing arr 1

arranging arr 2

S.C → O(1)

Best Solution : (Gap Method)