Leetcode 88 - Merge Sorted A

Merge 2 sorted array without any extra space.

1. arr1[] = { 1, 2, 4, 5, 6 }, arr2[] = { 7, 8, 9 }
   0 1 2 3 4 — 0 1 2

Soln:- Will take 2 pointer to merge array

And, also will take 3rd array

arr3[] = { 1, 2, 4, 5, 6, 7, 8, 9 }
0, 1, 2, 3, 4, 5, 6, 7

Size of array = m+n

∵ m = arr1.length;
n = arr2.length;

By brute force approach, if initially will not think much then we will take temporary array. Copy the elements from both the array, sort the array and copy back the elements to nums 1.

```java
class Solution {
    public void merge(int[] nums1, int m, int[] nums2, int n) {

        //Brute force approach - Copy elements of nums1 and nums2 into temporary array , then copy back the sorted elements into nums1

        int[] temp = new int[m+n];

        for(int i=0 ; i <m ; i++){
            temp[i] = nums1[i];
        }

        for(int i=0 ; i < n ; i++){
            temp[m+i] = nums2[i];
        }

        Arrays.sort(temp);

        for(int i=0; i< m+n ; i++){
            nums1[i] = temp[i];
        }

        System.out.println(Arrays.toString(nums1));

    }
}
```

TC

Copying elements - $O(m+n)$

Sorting array - $O((m+n)\log(m+n))$

Copying back - $O(m+n)$

Overall - $O((m+n)\log(m+n))$

SC - $O(m+n)$

Here we have used for loop, that's why it's taking much TC & SC. We can solve it by while loop as well.

Optimized App - Two Pointer

Eg. nums1 = [1, 2, 3, 0, 0, 0], m = 3
nums2 = [2, 5, 6], n = 3

Output - [1, 2, 2, 3, 5, 6]

nums1 = [1, 2, 3, 0, 0, 0] , nums2 = [2, 5, 6]

i = 0 i = 1 i = 2 (nums1)

i = 0 i = 1 i = 2 (nums2)

i, pointer = m - 1

j, pointer = n - 1

Output, nums1 = [1, 2, 3, 2, 5, 6]

i = 0 i = 1 i = 2 i = 3 i = 4 i = 5

m n.

int k = m + n - 1 = 6 - 1
= 5

> We will start from the back to avoid shifting elements.

> We compare elements from nums1 and nums2, placing the larger element at the end.

> If any elements from nums2 are left, we copy them.

> If nums1 has remaining elements, they

are already in the correct position.

Dry run

nums1 = [1, 2, 3, 0, 0, 0], nums2 = [2, 5, 6]

nums 1 = [1, 2, 3, 0, 0, 0]

Iteration 1, nums1[2] = 3, nums2[2] = 6

6 > 3, so place 6 at nums1[5] = 6

nums 1 = [1, 2, 3, 0, 0, 0] → [1, 2, 3, 0, 0, 6]

Iteration 2, nums1[2] = 3, nums2[1] = 5

5 > 3, so place 5 at nums1[4] = 5

[1, 2, 3, 0, 0, 6] → [1, 2, 3, 0, 5, 6]

Iteration 3, num1[2] = 3, nums2[0] = 2

3 > 2, so place at 3 nums1[3]

[1, 2, 3, 0, 5, 6]

[1, 2, 3, 3, 5, 6]

Iteration 4, nums1[1] = 2, nums2[0] = 2

2 <= 2, So place 2 at nums1[2]

[1, 2, ~~3~~, 3, 5, 6]

2

[1, 2, 2, 3, 5, 6]

Now, j move to -1 (nums2 is exhausted) and k to 1

So, num2 is exhausted (j<0)

And, nums1 has the remaining elements in place, so the process is done.

```java
class Solution {
    public void merge(int[] nums1, int m, int[] nums2, int n) {
        int i = m-1; // Pointer for nums1's last non-zero element
        int j = n-1; // Pointer for nums2's last element
        int k = m + n -1 ; // Pointer for placing elements in nums1 from the end

// Merge from the end to the start
        while(i >= 0 && j >= 0){

            if(nums1[i] > nums2[j]){
                nums1[k--] = nums1[i--];   // Place the larger element at the end
            }else{
                nums1[k--] = nums2[j--];
            }
        }
// If nums2 has remaining elements, copy them
        while(j>=0){
            nums1[k--] = nums2[j--];
        }

        System.out.println(Arrays.toString(nums1));

    }
}
```

TC - O(m+n)
SC - O(1)