Merge Sorted Array

Given two sorted arrays **arr1[]** and **arr2[]** of sizes **n** and **m** in non-decreasing order. Merge them in sorted order without using any extra space. Modify arr1 so that it contains the first N elements and modify arr2 so that it contains the last M elements.

Input:

n = 4, arr1[] = [1 3 5 7]

m = 5, arr2[] = [0 2 6 8 9]

Output:

arr1[] = [0 1 2 3]

arr2[] = [5 6 7 8 9]

Explanation:

After merging the two

non-decreasing arrays, we get,

0 1 2 3 5 6 7 8 9.

Naïve Approach: time = O(m+n) ; space=O(m+n);

```
class Solution {
   public void merge(int[] nums1, int m, int[] nums2, int n) {
int a[]=new int[m+n];
      int i=0,j=0,l=0;
      while(i<m&&j<n)
      {
         if(nums1[i]<=nums2[j])
         {
            a[l++]=nums1[i++];


         }
         else
         {
```

```
            a[l++]=nums2[j++];
          }
        }
        while(i<m)
        {
          a[l++]=nums1[i++];
        }
        while(j<n)
        {
          a[l++]=nums2[j++];
        }
        for(int k=0;k<a.length;k++)
        {
          nums1[k]=a[k];
        }
}
}
```

Better Approach: time = (m*n) space=O(1)

It uses insertion sort technique

```
class Solution {
    public void merge(int[] nums1, int m, int[] nums2, int n) {
      int l=0;
      int r=0;
        while(l<m&&r<n)
        {

          if(nums1[l]>nums2[r])
          {
```

```
            int t=nums1[l];
            nums1[l]=nums2[r];
            nums2[r]=t;
            insertElement(nums2);

        }
        l++;
    }
    while(r<n)
    {
        nums1[l++]=nums2[r++];
    }

}
public void insertElement(int[] arr)
{
    int e=arr[0];
    for(int i=1;i<arr.length;i++)
    {
        if(e>arr[i])
        {
            arr[i-1]=arr[i];
        }
        else
        {
            arr[i-1]=e;
            return;
        }
```

```
        }
        arr[arr.length-1]=e;
    }
}
```

Efficient Approach: time =O(Nlog(N)) space: O(1)

It uses gap method

```
class Solution
{
    //Function to merge the arrays.
    public static void merge(long arr1[], long arr2[], int n, int m)
    {

        int gap=n+m;
        for( gap=nextGap(gap);gap>0;gap=nextGap(gap))
        {

            for(int i=0;i<(m+n);i++)
            {
                int j=i+gap;
                //j reach out of the m+n
                if(j>=m+n)
                break;
                //if i and j in the first array
                if(i<n&&j<n)
                {
                    if(arr1[i]>arr1[j])
                    {
                        swap(arr1,arr1,i,j);
```

```
            }
        }
        //if i in first array and j in second array
        else if(i<n&&j>=n)
        {
            if(arr1[i]>arr2[j-n])
            swap(arr1,arr2,i,j-n);
        }
        //if i and j both are in second array
        else if(i>=n&&j>=n)
        {
            if(arr2[i-n]>arr2[j-n])
            {
                swap(arr2,arr2,i-n,j-n);
            }
        }
      }
   }
}
public static void swap(long[] a,long[] b,int i,int j)
{
    long l=a[i];
    a[i]=b[j];
    b[j]=l;
}
public static int nextGap(int gap)
{
    if(gap==0||gap==1)
```

```
        {
            return 0;
        }
        //gap%2 is to get the celing value
        return gap/2+gap%2;
    }
}
```