Merge two sorted arrays with O(1) extra space

We are given two sorted array. We need to merge these two arrays such that the initial numbers (after complete sorting) are in the first array and the remaining numbers are in the second array. Extra space allowed in O(1).

Example:

```
Input: ar1[] = {1, 5, 9, 10, 15, 20};
       ar2[] = {2, 3, 8, 13};
Output: ar1[] = {1, 2, 3, 5, 8, 9}
        ar2[] = {10, 13, 15, 20}
```

We strongly recommend you to minimize your browser and try this yourself first.

This task is simple and O(m+n) if we are allowed to use extra space. But it becomes really complicated when extra space is not allowed and doesn't look possible in less than O(m*n) worst case time.

The idea is to begin from last element of ar2[] and search it in ar1[]. If there is a greater element in ar1[], then we move last element of ar1[] to ar2[]. To keep ar1[] and ar2[] sorted, we need to place last element of ar2[] at correct place in ar1[]. We can use Insertion Sort type of insertion for this. Below is algorithm:

```
1) Iterate through every element of ar2[] starting from last
   element. Do following for every element ar2[i]
    a) Store last element of ar1[i]: last = ar1[i]
    b) Loop from last element of ar1[] while element ar1[j] is
       smaller than ar2[i].
          ar1[j+1] = ar1[j] // Move element one position ahead
          j--
    c) If any element of ar1[] was moved or (j != m-1)
          ar1[j+1] = ar2[i]
          ar2[i] = last
```

In above loop, elements in ar1[] and ar2[] are always kept sorted.

Below is C++ implementation of above algorithm.

```
// C++ program for implementation of Sieve of Atkin
#include <bits/stdc++.h>
using namespace std;

// Merge ar1[] and ar2[] with O(1) extra space
void merge(int ar1[], int ar2[], int m, int n)
{
    // Iterate through all elements of ar2[] starting from
    // the last element
    for (int i=n-1; i>=0; i--)
    {
        /* Find the smallest element greater than ar2[i]. Move all
           elements one position ahead till the smallest greater
           element is not found */
        int j, last = ar1[m-1];
        for (j=m-1; j >= 0 && ar1[j] > ar2[i]; j--)
            ar1[j+1] = ar1[j];

        // If there was a greater element
        if (j != m-1)
        {
            ar1[j+1] = ar2[i];
            ar2[i] = last;
        }
    }
}

// Driver program
int main(void)
{
    int ar1[] = {1, 5, 9, 10, 15, 20};
    int ar2[] = {2, 3, 8, 13};
    int m = sizeof(ar1)/sizeof(ar1[0]);
    int n = sizeof(ar2)/sizeof(ar2[0]);
    merge(ar1, ar2, m, n);

    cout << "After Merging \nFirst Array: ";
    for (int i=0; i<m; i++)
        cout << ar1[i] << " ";
    cout << "\nSecond Array: ";
    for (int i=0; i<n; i++)
        cout << ar2[i] << " ";
   return 0;
}
```

Output:

```
After Merging
First Array: 1 2 3 5 8 9
Second Array: 10 13 15 20
```

Illustration:

```
Initial Arrays:
   ar1[] = {1, 5, 9, 10, 15, 20};
   ar2[] = {2, 3, 8, 13};

After First Iteration:
   ar1[] = {1, 5, 9, 10, 13, 15};
   ar2[] = {2, 3, 8, 20};
// 20 is moved from ar1[] to ar2[]
// 13 from ar2[] is inserted in ar1[]

After Second Iteration:
   ar1[] = {1, 5, 8, 9, 10, 13};
   ar2[] = {2, 3, 15, 20};
// 15 is moved from ar1[] to ar2[]
// 8 from ar2[] is inserted in ar1[]

After Third Iteration:
   ar1[] = {1, 3, 5, 8, 9, 10};
   ar2[] = {2, 13, 15, 20};
// 13 is moved from ar1[] to ar2[]
// 3 from ar2[] is inserted in ar1[]

After Fourth Iteration:
   ar1[] = {1, 2, 3, 5, 8, 9};
   ar2[] = {10, 13, 15, 20};
```

// 10 is moved from ar1[] to ar2[]
// 2 from ar2[] is inserted in ar1[]

The worst case time complexity of code/algorithm is O(m*n). The worst case occurs when all elements of ar1[] are greater than all elements of ar2[].