Merge Two sorted arrays

BRUTE FORCE :

Making use of a third array and two pointers to compare the values. The value of whichever element out of two arrays is less is inserted into ans array. But the question requires without extra space.

```
#include <bits/stdc++.h>

vector<int> ninjaAndSortedArrays(vector<int>& arr1, vector<int>& arr2, int m, int n) {
  // Write your code here.
  vector<int> ans(m+n,0);
  int ind=0,i=0,j=0;
  while(i<m && j<n)
  {
    if(arr1[i]<arr2[j])
    {
      ans[ind]=arr1[i];
      ind++;
      i++;
    }
    else
    {
      ans[ind]=arr2[j];
      ind++;
      j++;
    }
  }
  while(i<m)
  {
    ans[ind]=arr1[i];
    i++;ind++;

  }
  while(j<n)
  {
    ans[ind]=arr2[j];
    j++;ind++;
  }
  return ans;
}
```

- Time Complexity : O(M+N)
- Space Complexity : O(M+N).

Approach 2 :

Taking advantage of sorted array traversing arr1 backwards and arr2 forward. If arr1 value is greater then swap the values of arr1 and arr2 else break from loop.

Now the arr1 contains all smaller elements and arr2 contains all larger elements but they are not sorted so performing sort on individual arrays.

```
#include <bits/stdc++.h>

vector<int> ninjaAndSortedArrays(vector<int>& arr1, vector<int>& arr2, int m, int n) {
  // Write your code here.
  int left=m-1,right=0;
  while(left>=0 && right< n)
  {
    if(arr1[left]>arr2[right])
    {
      swap(arr1[left],arr2[right]);
    }
    else
    {
      break;
    }
    left--;
    right++;
  }
  sort(arr1.begin(),arr1.begin()+m);
  sort(arr2.begin(),arr2.end());
  int ind=0;
  for(int i=m;i<m+n;i++)
  {
    arr1[i]=arr2[ind++];
  }
  return arr1;
}
```

- Time Complexity : O(min(m,n)) + O(mlog(m))+ O(nlogn)
- Space complexity :O(1)

Approach 3 :

Using gap algorithm - a technique used in shell sort.

We find gap using ceil(m+n/2). We maintain two pointers left which is placed at the starting of 1st array and right which is placed at gap distance from left if the elem pointing by left is greater then we swap. Perform the algo until gap==1 (One iteration will run for gap=1 then 1/2 gives agian 1 and this continues so we need to break when gap =1 is repeated again.

m+n%2 in gap is done to perform ceiling function.

```
#include <bits/stdc++.h>

vector<int> ninjaAndSortedArrays(vector<int>& arr1, vector<int>& arr2, int m, int n) {
  // Write your code here.
  int left,right,gap=(m+n/2)+(m+n%2);
  while(gap>0)
  {
    left=0;
    right=left+gap;
    while(right<m+n)
    {
      if(left<m && right>=m)
      {
        if(arr1[left]>arr2[right-m])
          swap(arr1[left],arr2[right-m]);
      }
      else if(left>=m)
      {
        if(arr2[left-m]>arr2[right-m])
          swap(arr2[left-m],arr2[right-m]);
      }
      else
      {
        if(arr1[left]>arr1[right])
          swap(arr1[left],arr1[right]);

      }
      left++;right++;
    }
    if(gap==1)
      break;
    gap=(gap/2)+(gap%2);

  }
  int ind=0;
  for(int i=m;i<m+n;i++)
  {
    arr1[i]=arr2[ind++];
  }
  return arr1;
}
```

- Time Complexity : O(m+n log(m+n))
- Space Complexity : O(1).

Approach 4 :

using 3 pointers one to make entry to the arr1 that are initialised 0 for new elem,another for arr1 and another for arr2 traversing backwards.

if a greater elem is found at arr1 then replace with that elem and move pointers accordingly. Traversing whole array 1 and 2 until all the elem are compared and placed at their correct position.

```
#include <bits/stdc++.h>

vector<int> ninjaAndSortedArrays(vector<int>& arr1, vector<int>& arr2, int m, int n) {
  // Write your code here.
  int i=m+n-1,j=m-1,k=n-1;
  while(j>=0 || k>=0)
  {
    if(j>=0 &&(k<0 || arr1[j]>=arr2[k]))
      arr1[i--]=arr1[j--];
    if(k>=0 &&(j<0 || arr1[j]<arr2[k]))
      arr1[i--]=arr2[k--];
  }
  return arr1;
}
```