

Solution: Merge Sorted Array

Let's solve the Merge Sorted Array problem using the K-way Merge pattern.

We'll cover the following

- Statement
- Solution
 - Naive approach
 - Optimized approach using k-way merge
 - Step-by-step solution construction
 - Just the code
 - Solution summary
 - Time complexity
 - Space complexity

Statement

Given two sorted integer arrays, *nums1* and *nums2*, and the number of data elements in each array, *m* and *n*, implement a function that merges the second array into the first one. You have to modify *nums1* in place.

Note: Assume that *nums1* has a size equal to $m + n$, meaning it has enough space to hold additional elements from *nums2*.


Constraints:

- `nums1.length = m + n`
- `nums2.length = n`
- $0 \leq m, n \leq 200$
- $1 \leq m + n \leq 200$
- $-10^3 \leq \text{nums1}[i], \text{nums2}[j] \leq 10^3$


Solution

So far, you've probably brainstormed some approaches and have an idea of how to solve this problem. Let's explore some of these approaches and figure out which one to follow based on considerations such as time complexity and any implementation constraints.

Naive approach

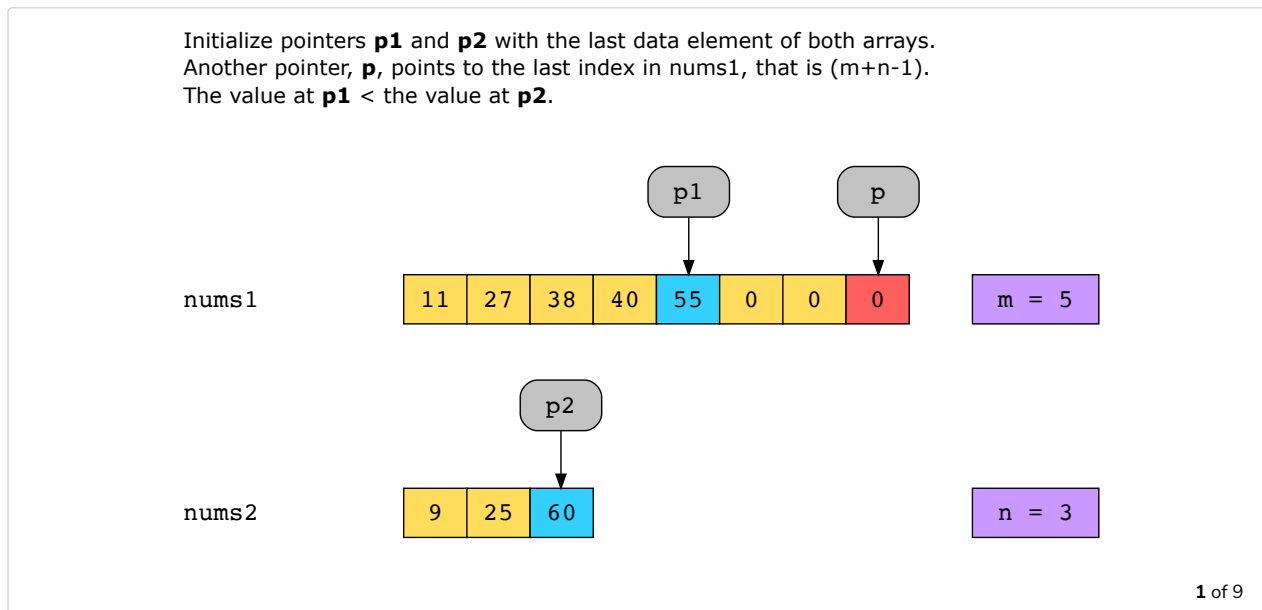
The naive approach here is to append the second list to the first—at a cost of $O(n)$ —and then sort it using quick sort—at a cost of $O((m + n) \log(m + n))$ —for an overall time complexity of $O((m + n) \log(m + n))$. 

Optimized approach using k-way merge

Since we have two sorted arrays to merge, this problem is the simplest illustration of the k-way merge pattern. 

With the k-way merge approach, we iterate over our given arrays using two pointers and merge them in place. The time complexity for this is $O(m + n)$, which is more efficient than the $O((m + n) \log(m + n))$ complexity of the naive approach.

We can solve this problem by comparing both arrays from the end and populating the result in the `nums1` array. The slides below illustrate how we would like the algorithm to run:



Note: In the following section, we will gradually build the solution. Alternatively, you can skip straight to [just the code](#).

Step-by-step solution construction

First, we initialize two pointers, `p1` and `p2`, that point to the last element of each array. `p1` is initialized to `m - 1`, the index of the last data element in `nums1`, and `p2` is initialized to `n - 1`, the index of the last data element in `nums2`.

Java

```
43     {12, 34, 45, 56, 67, 78, 89, 99, 0}
44     };
45     int[][] nums2 = {
46         {32, 49, 50, 51, 61, 99},
47         {7},
48         {1, 2, 3, 4},
49         {-45, -99},
50         {100}
51     };
52     int k = 1;
53     for (int i = 0; i < m.length; i++) {
54         System.out.print(k + ".\tnums1: [");
55         for (int j = 0; j < nums1[i].length - 1; j++) {
56             System.out.print(nums1[i][j] + ", ");
57         }
58         System.out.println(nums1[i][nums1[i].length - 1] + "], m: " + m[i]);
59         System.out.print("\tnums2: [");
60         for (int j = 0; j < nums2[i].length - 1; j++) {
61             System.out.print(nums2[i][j] + ", ");
62         }
63         System.out.println(nums2[i][nums2[i].length - 1] + "], n: " + n[i]);
64         mergeSorted(nums1[i], m[i], nums2[i], n[i]);
65         System.out.println("\tMerged list: None");
```

```

66     System.out.println(PrintHyphens.repeat("-", 100));
67     k += 1;
68 }
69 }
70 }

```



Initializing the pointers

Next, we initialize another pointer called `p`, which points to the `m + n - 1` index of `nums1`. We use the pointer `p` to traverse the `nums1` array from the end. If the value at `p1` is greater than the value at `p2`, the result at `p` is equal to the value at `p1`. We then decrement `p1` to move it one element back.

Java

```

1  class MergeSorted {
2
3      public static String printArraywithMarkers(int[] arr, int pValue) {
4          String out = "[";
5          for (int i = 0; i < arr.length - 1; i++) {
6              if (i == pValue) {
7                  out += "«";
8                  out += String.valueOf(arr[i]) + "»" + ", ";
9              }
10             else {
11                 out += String.valueOf(arr[i]) + ", ";
12             }
13         }
14         if (pValue == arr.length - 1) {
15             out += "«" + arr[arr.length - 1];
16             out += "»";
17         }
18         else {
19             out += arr[arr.length - 1];
20             out += "】";
21         }
22         return out;
23     }
24     public static String printArraywithMarkers(int[] arr, int iValue, int jValue) {
25         String out = "[";
26         for (int i = 0; i < arr.length - 1; i++) {
27             if (i == iValue || i == jValue) {
28                 out += "«";

```



Traversing the array

If the value at `p1` is less than the value at `p2`, the result at `p` is set equal to the value at `p2`. We decrement the pointer for the list that the entry was copied from.

The traversal continues until `nums2` is merged with `nums1`.

Java

```

1  class MergeSorted {
2
3      public static String printArraywithMarkers(int[] arr, int pValue) {
4          String out = "[";
5          for (int i = 0; i < arr.length - 1; i++) {
6              if (i == pValue) {
7                  out += "«";
8                  out += String.valueOf(arr[i]) + "»" + ", ";
9              }
10             else {
11                 out += String.valueOf(arr[i]) + ", ";
12             }
13         }
14         if (pValue == arr.length - 1) {

```



```

14     if (pvalue == arr.length - 1) {
15         out += "«" + arr[arr.length - 1];
16         out += "»";
17     }
18     else {
19         out += arr[arr.length - 1];
20         out += " ";
21     }
22     return out;
23 }
24 public static String printArrayWithMarkers(int[] arr, int iValue, int jValue) {
25     String out = "[";
26     for (int i = 0; i < arr.length - 1; i++) {
27         if (i == iValue || i == jValue) {
28             out += "«";

```



Merging the Array

Just the code

Here's the complete solution to this problem:

Java

```

1  class MergeSorted {
2
3      public static int[] mergeSorted(int[] nums1, int m, int[] nums2, int n) {
4          int p1 = m - 1;
5          int p2 = n - 1;
6
7          for (int p = m + n - 1; p >= 0; p--) {
8              if (p2 < 0) {
9                  break;
10             }
11             if (p1 >= 0 && nums1[p1] > nums2[p2]) {
12                 nums1[p] = nums1[p1];
13                 p1 -= 1;
14             }
15             else {
16                 nums1[p] = nums2[p2];
17                 p2 -= 1;
18             }
19         }
20         return nums1;
21     }
22
23     // Driver code
24     public static void main(String args[]) {
25         int[] m = {9, 2, 3, 1, 8};
26         int[] n = {6, 1, 4, 2, 1};
27         int[][] nums1 = {
28             {22, 23, 25, 41, 44, 47, 56, 61, 105, 0, 0, 0, 0, 0, 0}

```



Solution summary

To recap, the solution to this problem can be divided into the following parts:

1. Initialize two pointers that point to the last data elements in both arrays.
2. Initialize a third pointer that points to the last index of *nums1*.
3. Traverse *nums1* from the end using the third pointer and compare the values corresponding to the first two pointers.
4. Place the larger of the two values at the third pointer's index.
5. Repeat the process until the two arrays are merged.



Time complexity

The time complexity is $O(n + m)$, where n and m are the counts of initialized elements in the two arrays.

Space complexity

The space complexity is $O(1)$ because we only use the space required for three indices.

[← Back](#)

Merge Sorted Array

[Next →](#)

Kth Smallest Number i...



Mark as
Completed

