Ask a Question

Introduction to K-way Merge

Let's go over the K-way Merge pattern, its real-world applications, and some problems we can solve with it.

We'll cover the following...

About the pattern

The **K-way merge** pattern is an essential algorithmic strategy for merging K sorted data structures, such as arrays and linked lists, into a single sorted data structure. This technique is an expansion of the standard merge sort algorithm, which traditionally merges two sorted data structures into one.

> **Note:** For simplicity, we'll use the term lists to refer to arrays and linked lists in our discussion.

To understand the basics of this algorithm, first, we need to know the basic idea behind the K-way merge algorithm. The K-way merge algorithm works by repeatedly selecting the smallest (or largest, if we're sorting in descending order) element from among the first elements of tl K input lists and adding this element to a new output list (with the same data type as the inputs). This process is repeated until all elements from all input lists have been merged into the output list, maintaining the sorted order.

Now, let’s take a closer look at how the algorithm works. The K-way merge algorithm comprises two main approaches to achieve its goal, both leading to the same result.

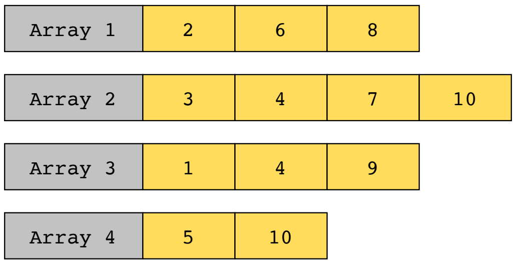
Here, we’ll discuss one of the approaches, which uses a minheap:

Using a min heap

1. Insert the first element of each list into a min heap. This sets up our starting point, with the heap helping us efficiently track the smallest current element among the lists.
2. Remove the smallest element from the heap (which is always at the top) and add it to the output list. This ensures that our output list is being assembled in sorted order.
3. Keep track of which list each element in the heap came from. This is for knowing where to find the next element to add to the heap.
4. After removing the smallest element from the heap and adding it to the output list, replace it with the next element from the same list the removed element belonged to.
5. Repeat steps 2–4 until all elements from all input lists have been merged into the output list.

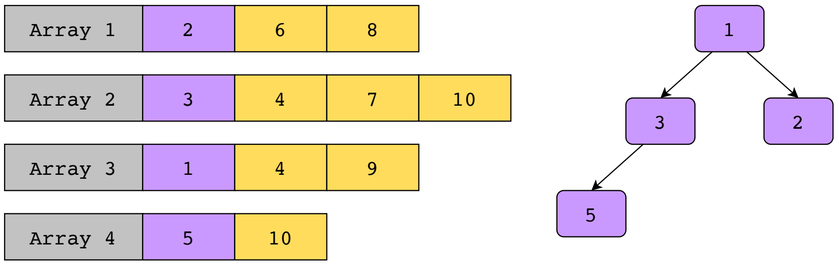
The slides below illustrate an example of using this approach with arrays:

Traverse all the arrays, starting from the first element of each array.



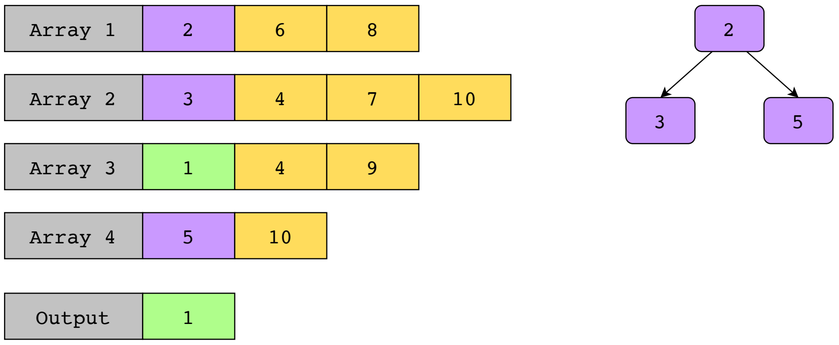
1 of 23

For the first elements, we simply push them into a min heap.



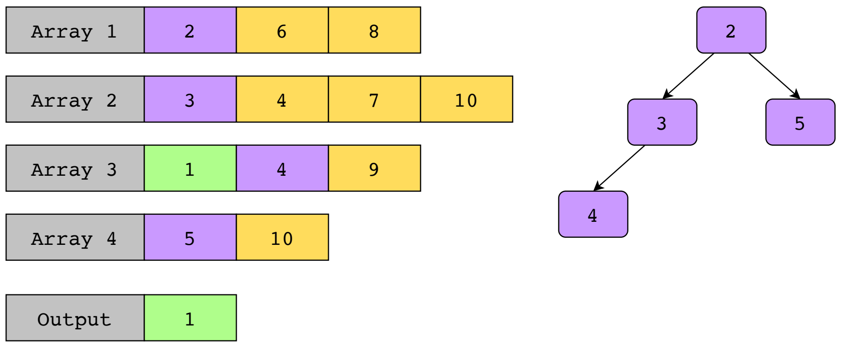
2 of 23

Pop the top of the min heap, that contains the smallest element, and add it to the **output** array.



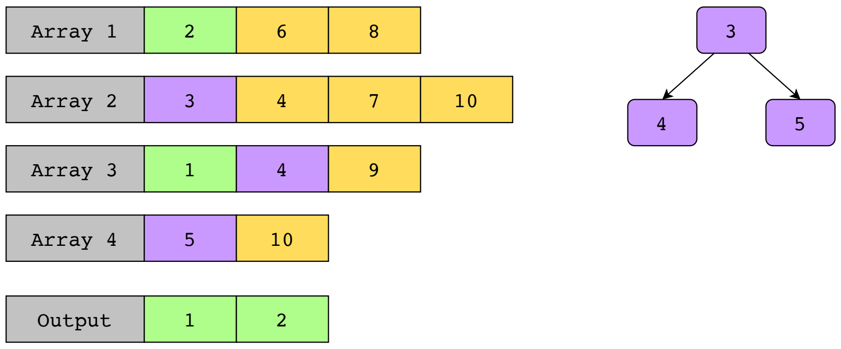
3 of 23

The pointer for the array that contains the previously popped element will be moved forward. The new element that this pointer points to will be pushed on to the heap.



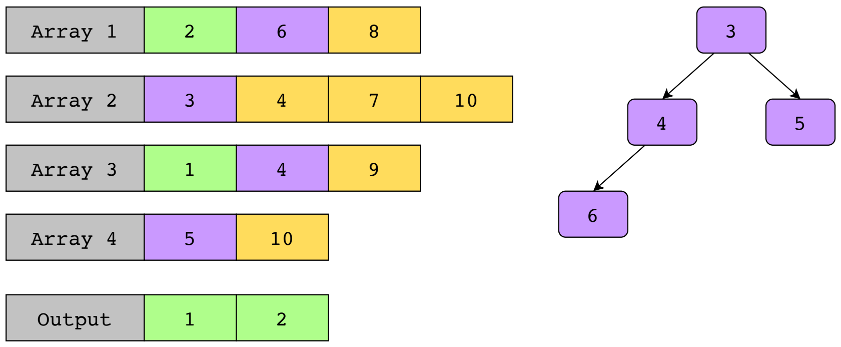
4 of 23

Pop the top of the min heap, that contains the smallest element, and add it to the **output** array.



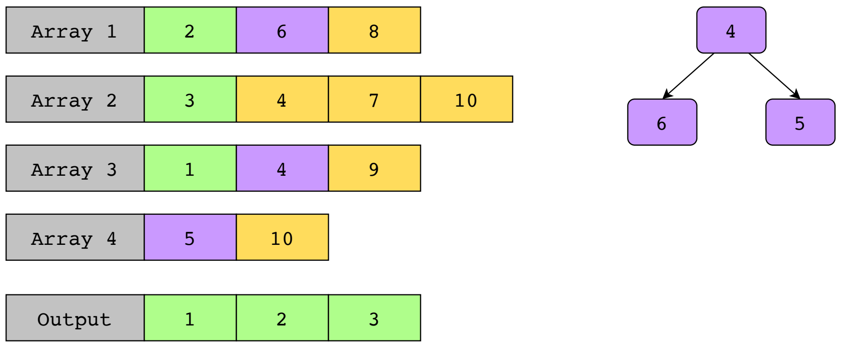
5 of 23

The pointer for the array that contains the previously popped element will be moved forward. The new element that this pointer points to will be pushed on to the heap.



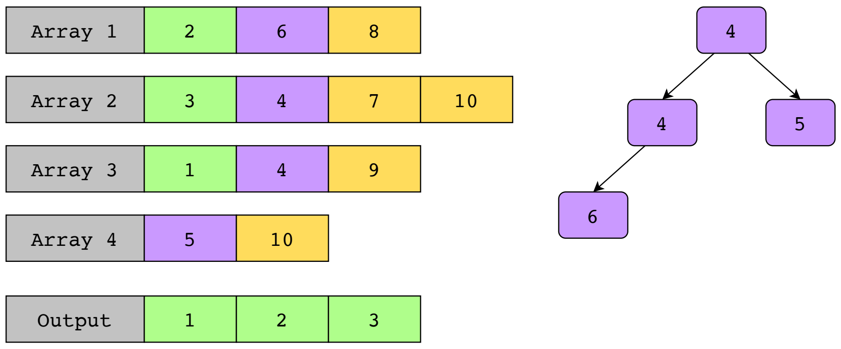
6 of 23

Pop the top of the min heap, that contains the smallest element, and add it to the **output** array.



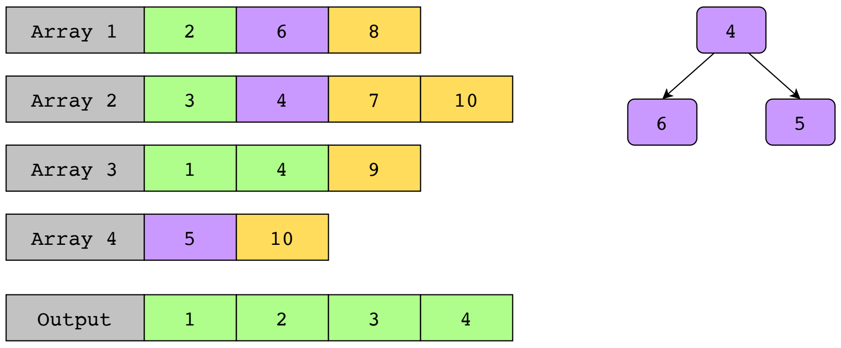
7 of 23

The pointer for the array that contains the previously popped element will be moved forward. The new element that this pointer points to will be pushed on to the heap.



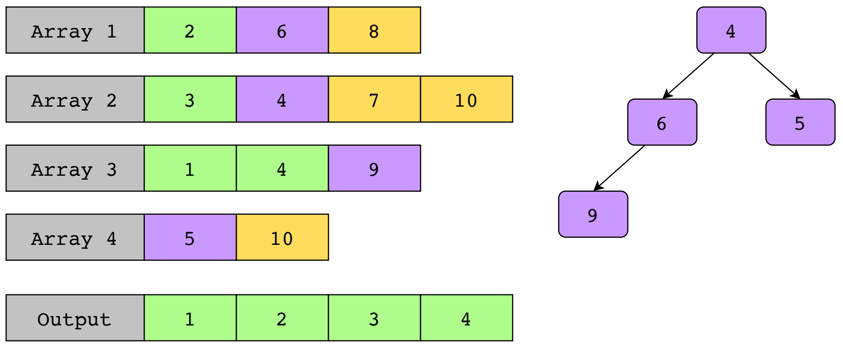
8 of 23

Pop the top of the min heap, that contains the smallest element, and add it to the **output** array.



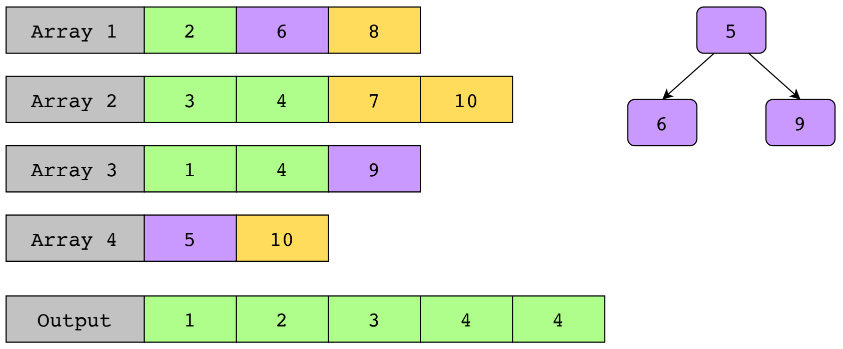
9 of 23

The pointer for the array that contains the previously popped element will be moved forward. The new element that this pointer points to will be pushed on to the heap.



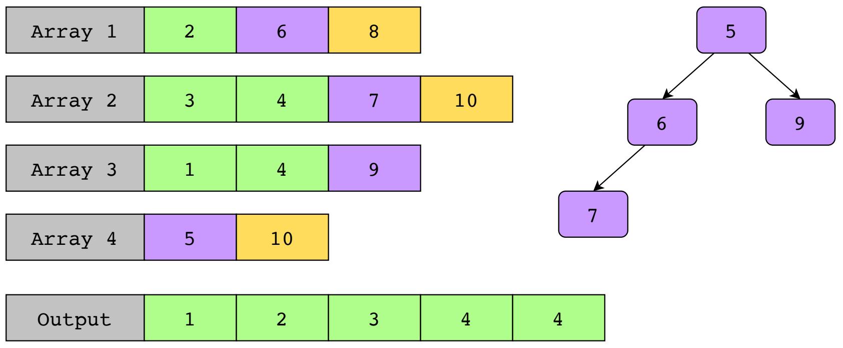
10 of 23

Pop the top of the min heap, that contains the smallest element, and add it to the **output** array.



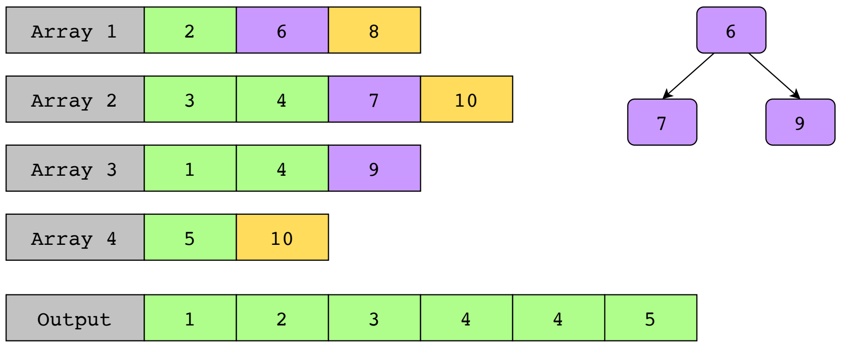
11 of 23

The pointer for the array that contains the previously popped element will be moved forward. The new element that this pointer points to will be pushed on to the heap.



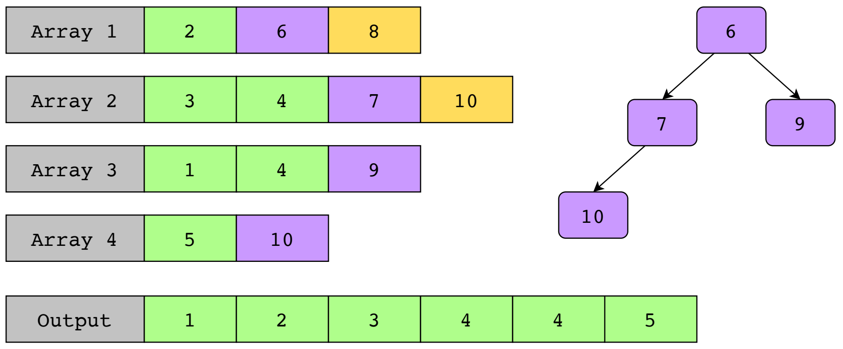
12 of 23

Pop the top of the min heap, that contains the smallest element, and add it to the **output** array.



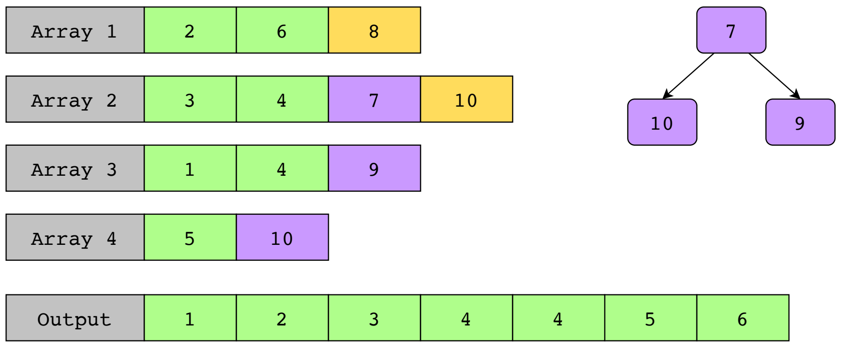
13 of 23

The pointer for the array that contains the previously popped element will be moved forward. The new element that this pointer points to will be pushed on to the heap.



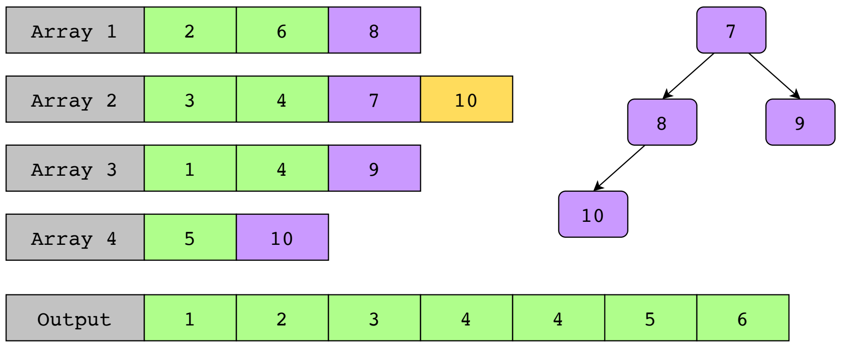
14 of 23

Pop the top of the min heap, that contains the smallest element, and add it to the **output** array.



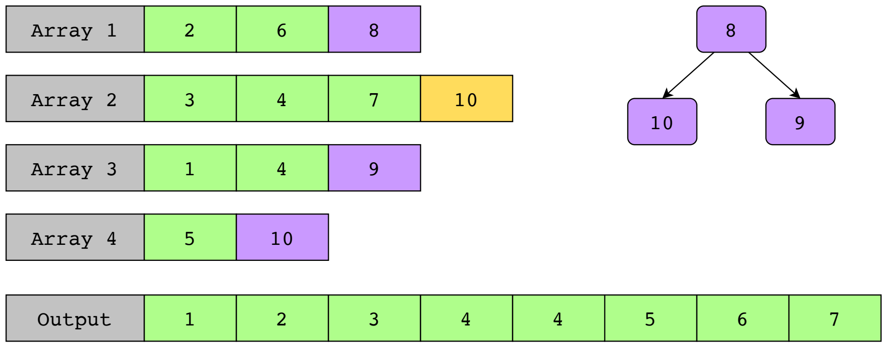
15 of 23

The pointer for the array that contains the previously popped element will be moved forward. The new element that this pointer points to will be pushed on to the heap.



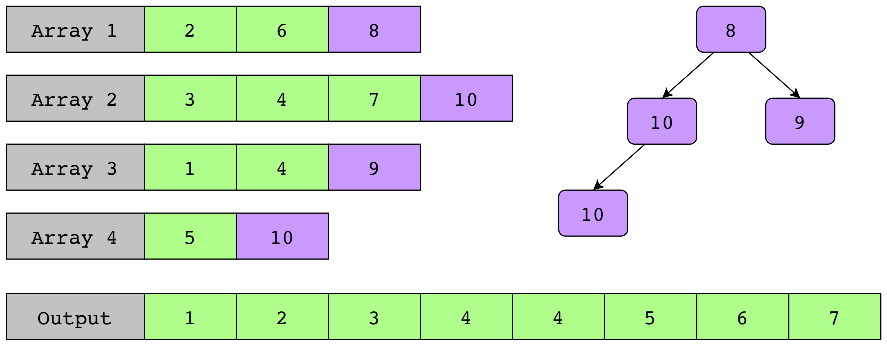
16 of 23

Pop the top of the min heap, that contains the smallest element, and add it to the **output** array.



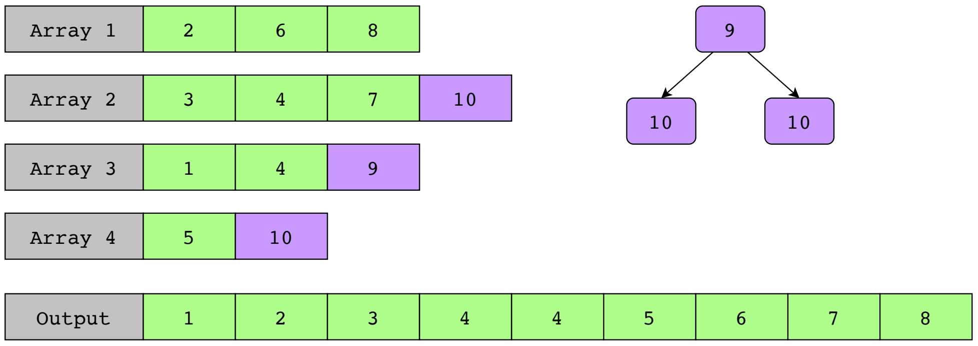
17 of 23

The pointer for the array that contains the previously popped element will be moved forward. The new element that this pointer points to will be pushed on to the heap.



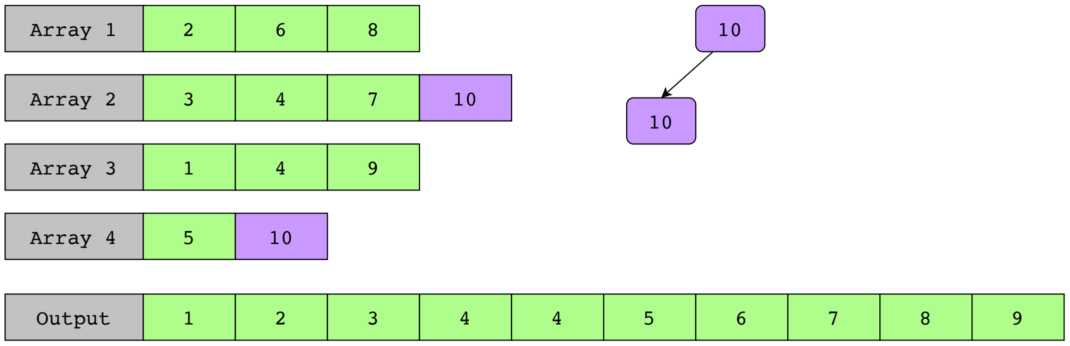
18 of 23

Pop the top of the min heap, that contains the smallest element, and add it to the **output** array.



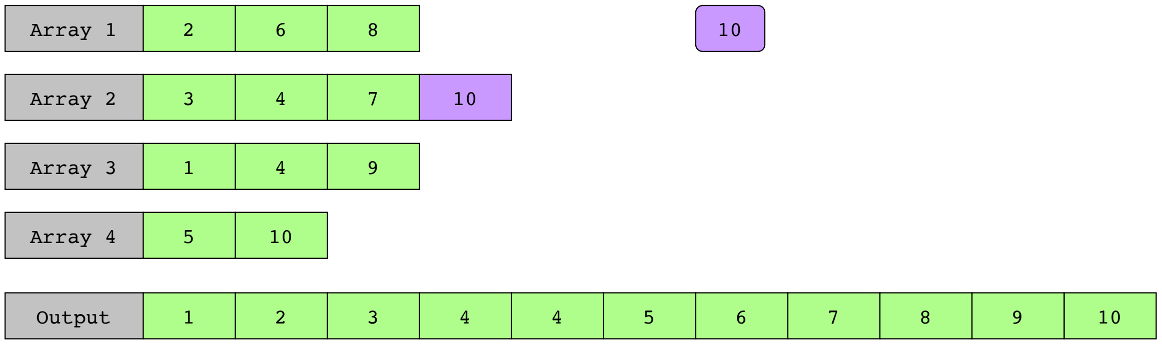
19 of 23

Because there are no more elements left to add, we will continue popping from the heap and add popped elements to the **output** array until the heap is empty.



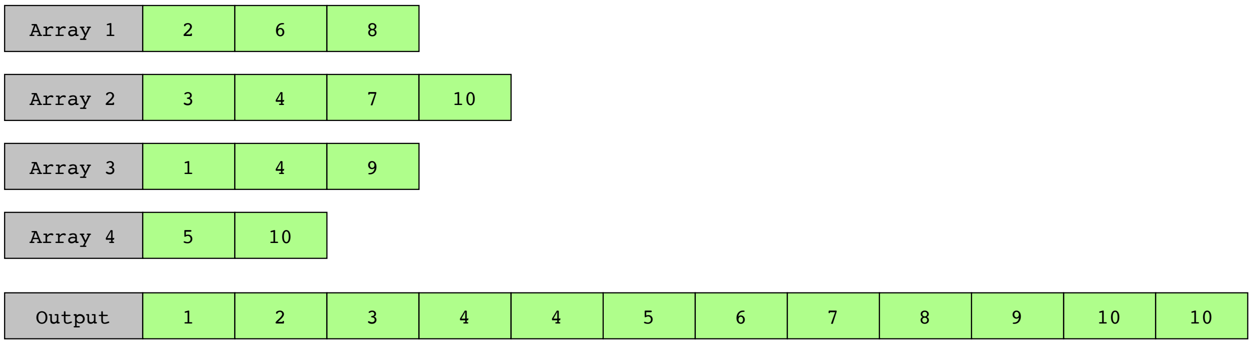
20 of 23

Because there are no more elements left to add, we will continue popping from the heap and add popped elements to the **output** array until the heap is empty.



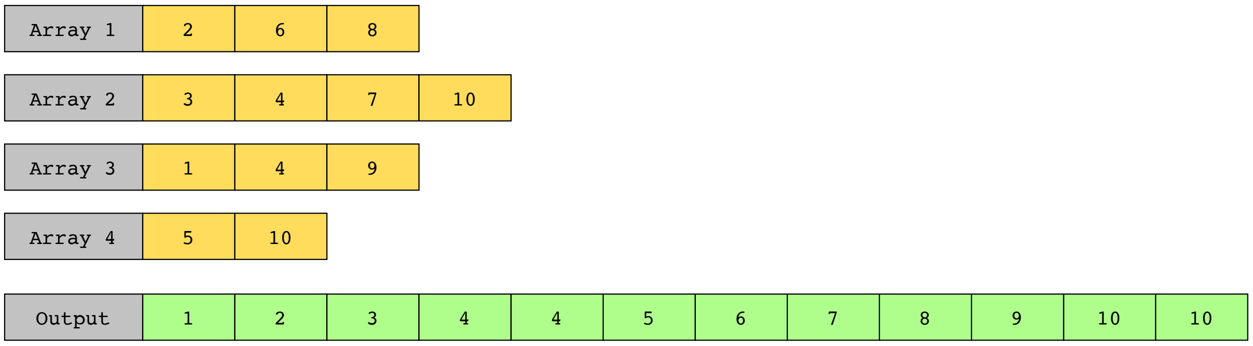
21 of 23

Because there are no more elements left to add, we will continue popping from the heap and add popped elements to the **output** array until the heap is empty.



22 of 23

Because the heap is now empty, the **output** array contains all the elements of the arrays in sorted order.



23 of 23

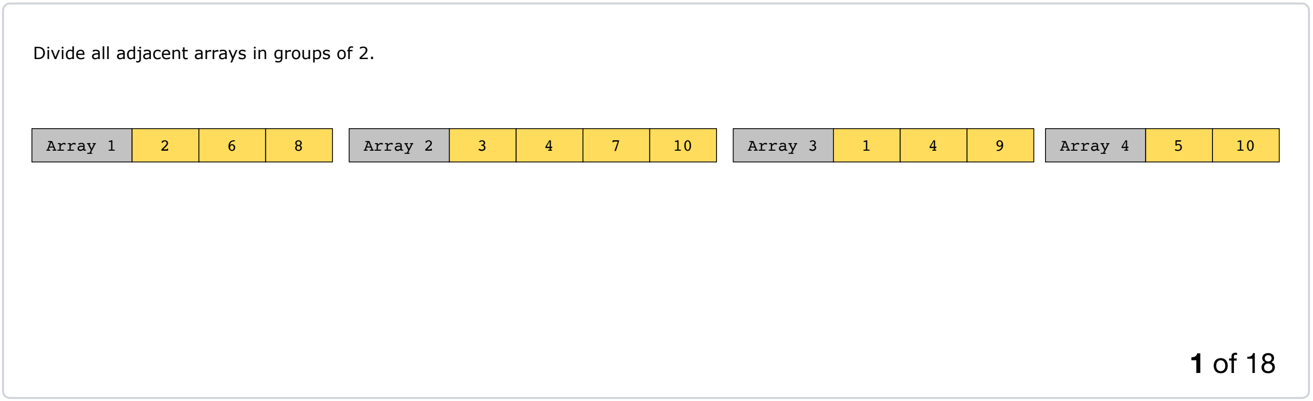
Besides using a min heap, another effective approach for performing a K-way merge involves grouping and merging pairs of lists. This technique simplifies the merging process by reducing it to a series of two-way merges.

Making groups of two and repeatedly merging them

Here are the steps of this method:

1. Start by dividing the K sorted lists into pairs, making groups of two. This organizes our lists into manageable units for merging.
2. For each pair of lists, perform a standard two-way merge operation. This is similar to the merge step in merge sort, where two sorted lists are combined into a single sorted list. This step results in $k/2$ merged lists.
3. If there are an odd number of lists in a group at any point, simply leave one list unmerged in that round. This ensures that no list is left out of the merging process.
4. Repeat the process of pairing up the resulting lists from the previous merge and merging them again until only one sorted list remains, which is the final result.

The slides below illustrate an example of using this approach with arrays:

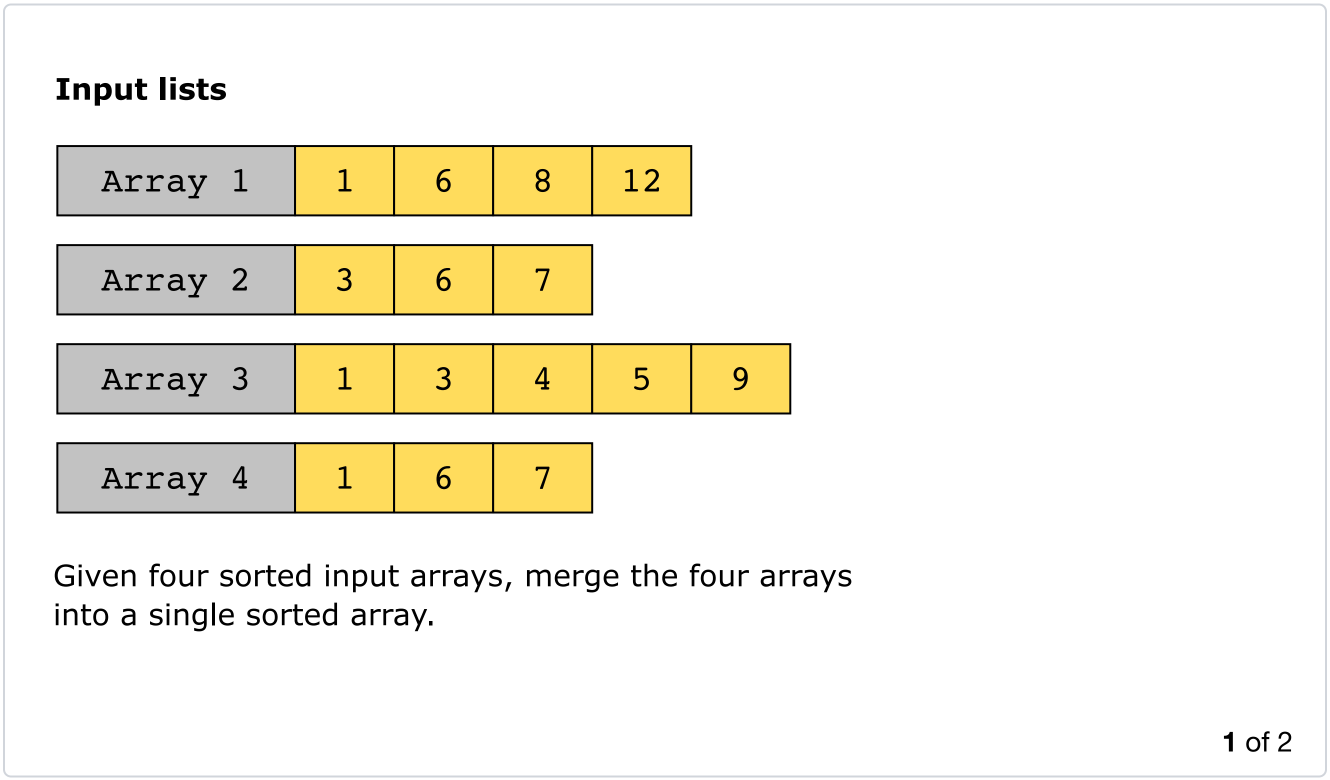

Divide all adjacent arrays in groups of 2.

1 of 18

Examples

The following examples illustrate some problems that can be solved wit
this approach:

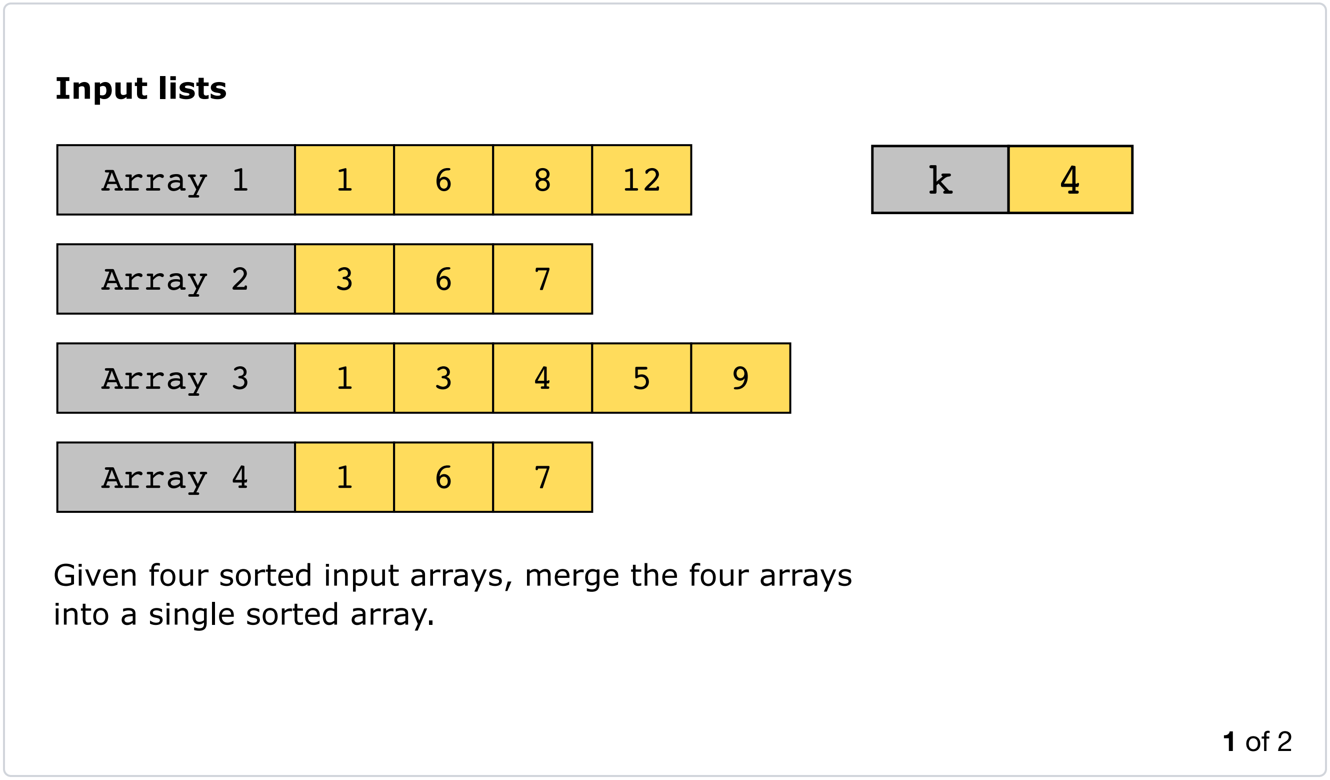
1. **Median of k sorted arrays:** Find the median of the k sorted arrays.



Given four sorted input arrays, merge the four arrays into a single sorted array.

1 of 2

2. **The kth smallest element in multiple sorted arrays:** Find the k^{th} smallest element of multiple sorted arrays.



Given four sorted input arrays, merge the four arrays into a single sorted array.

Does your problem match this pattern?

Yes, if one or both of the following conditions are fulfilled:

- **Involves merging sorted arrays or a matrix:** The problem involves a collection of sorted arrays or a matrix with rows or columns sorted in a specific order that needs to be merged. This could be the core of the problem or a step toward the solution.
- **Seeking the k^{th} smallest/largest across sorted collections:** The problem involves identifying the k^{th} smallest or largest element across multiple sorted arrays or linked lists.

Real-world problems

Many problems in the real world use the K-way merge pattern. Let's look at some examples.

- **Patient records aggregation:** In healthcare informatics, patient data often come from multiple sources, such as lab results, physician notes, and imaging reports, each sorted by date or priority. Integrating these data streams into a single, chronologically ordered patient record is essential for providing comprehensive care. The K-way merge pattern can efficiently combine these sorted data streams, ensuring doctors and nurses have timely access to a unified view of patient history, which is crucial for diagnosis and treatment planning.
- **Merging financial transaction streams:** Financial institutions process transactions from multiple sources, including trades, payments, and account transfers. Analysts need these transactions merged into a single stream to perform real-time market analysis or fraud detection. By applying the K-way merge, transactions from different sources can be integrated into a coherent order based on time or transaction ID, enabling more effective monitoring and analysis of financial activities.
- **Log file analysis:** Large-scale web services generate log files from multiple servers, each chronologically ordered. Analyzing these logs for insights into user behavior, system performance, or error diagnosis requires merging them into a single, time-ordered stream. The K-way merge pattern facilitates this by efficiently combining multiple sorted log files, enabling analysts to perform comprehensive log analysis without significant preprocessing.

Strategy time!

Match the problems that can be solved using the K-way merge pattern.

> **Note:** Select a problem in the left-hand column by clicking it, and then click one of the two options in the right-hand column.

Match The Answer

Select an option from the left-hand side