K-way Merge Pattern

The **K-way Merge** pattern is used to **merge K sorted lists (or arrays) into a single sorted list**.
Instead of merging two lists at a time, it uses a **heap** to always pick the smallest (or largest) among the current heads of all lists—allowing efficient merging.

Classic Use Cases:

- Merge K sorted arrays into one
- Find the Kth smallest number in a matrix of sorted rows/columns
- External sorting (when data is too big for memory)

K-way Merge Pattern:
Merge K Sorted Arrays

Input Arrays

arr1 = [1, 4, 7] arr2 = [2, 5, 8] arr3 = [3, 9]

Task: Merge these K sorted arrays into one sorted array.

Step 1 Start by pushing the first element of each array into the min-heap.

Step 2 Start to push:

Step 3 Repeat until the min-heap is empty.

Merged!

Tip: This pattern is used for merging multiple sorted lists, external sorting, and finding Kth smallest elements in sorted matrices!

Classic Problem: Merge K Sorted Arrays

Problem Statement:
Given K sorted arrays, merge them into a single sorted array.

Example:
Input:

arr1 = {1, 4, 7}
arr2 = {2, 5, 8}
arr3 = {3, 6, 9}

Output:
`{1, 2, 3, 4, 5, 6, 7, 8, 9}`

Approach

1. **Log Session a min-heap** (priority queue).
2. **Push the first element of each array** into the heap, along with info about which array and index it came from.
3. **Repeat until heap is empty:**
   - Pop the smallest item from the heap, add to the result.
   - If there's another element in the same array, push that next element to the heap.
4. At the end, your result array is sorted.

C Code Example: Merge K Sorted Arrays

Here's a simple implementation for 3 arrays (for clarity).

```
#include <stdio.h>
#include <stdlib.h>

#define K 3 // number of arrays
#define N 3 // size of each array

// Heap node stores value and source array info
typedef struct {
    int value;
    int arrIdx;
    int elemIdx;
} HeapNode;

void swap(HeapNode *a, HeapNode *b) {
    HeapNode temp = *a;
    *a = *b;
    *b = temp;
}

// Heapify up (for min-heap)
void heapifyUp(HeapNode heap[], int idx) {
    while (idx > 0) {
        int parent = (idx - 1) / 2;
```

```
        if (heap[parent].value > heap[idx].value) {
            swap(&heap[parent], &heap[idx]);
            idx = parent;
        } else break;
    }
}

// Heapify down (for min-heap)
void heapifyDown(HeapNode heap[], int size, int idx) {
    int left = 2 * idx + 1;
    int right = 2 * idx + 2;
    int smallest = idx;

    if (left < size && heap[left].value < heap[smallest].value)
        smallest = left;
    if (right < size && heap[right].value < heap[smallest].value)
        smallest = right;
    if (smallest != idx) {
        swap(&heap[smallest], &heap[idx]);
        heapifyDown(heap, size, smallest);
    }
}

void mergeKSortedArrays(int arr[K][N], int result[], int totalSize) {
    HeapNode heap[K];
    int heapSize = 0;

    // Initialize: push first element of each array
    for (int i = 0; i < K; i++) {
        heap[heapSize].value = arr[i][0];
        heap[heapSize].arrIdx = i;
        heap[heapSize].elemIdx = 0;
        heapifyUp(heap, heapSize);
        heapSize++;
    }

    int resIdx = 0;
    while (heapSize > 0) {
        // Pop min element
        HeapNode minNode = heap[0];
        result[resIdx++] = minNode.value;

        // If more elements in same array, insert next
        if (minNode.elemIdx + 1 < N) {
            heap[0].value = arr[minNode.arrIdx][minNode.elemIdx + 1];
            heap[0].arrIdx = minNode.arrIdx;
            heap[0].elemIdx = minNode.elemIdx + 1;
        } else {
            // Remove this heap node
            heap[0] = heap[heapSize - 1];
            heapSize--;
        }
        heapifyDown(heap, heapSize, 0);
    }
}
```

```
int main() {
    int arr[K][N] = {
        {1, 4, 7},
        {2, 5, 8},
        {3, 6, 9}
    };
    int result[K * N];

    mergeKSortedArrays(arr, result, K * N);

    printf("Merged array: ");
    for (int i = 0; i < K * N; i++) {
        printf("%d ", result[i]);
    }
    printf("\n");
    return 0;
}
```

How does this work?

- **Start:** Heap contains `{1, 2, 3}` from 3 arrays.
- **Pop:** 1 → add 4 from same array → heap now `{2, 3, 4}`
- **Pop:** 2 → add 5 from same array → heap now `{3, 4, 5}`
- ...and so on, always keeping the smallest of the next available elements.

Where is this useful?

- Merging many sorted lists/arrays
- Finding the Kth smallest/largest across sorted data
- External sorting (large data files)
- Merging K sorted linked lists (the logic is the same)

Practice Challenge

- Try merging K **linked lists** (instead of arrays).
- Try finding the **Kth smallest element** in a sorted matrix.