# K-Way Merge Pattern

The K-way Merge pattern is used when you are dealing with multiple sorted lists (or arrays, or streams, or linked lists) and need to merge or process them efficiently. Instead of merging two at a time repeatedly, we use a Min Heap (Priority Queue) to always pick the smallest (or largest) element among the K lists.

## Core Idea:

1. Initialize a Min Heap with the first element of each list. 2. Extract the minimum element from the heap. 3. Insert the next element from the same list into the heap. 4. Repeat until the heap is empty (or until k elements are processed).

## Dry Run Example:

```
Example: Merge K Sorted Lists
lists = [[1,4,7], [2,5,8], [3,6,9]]

- Initialize heap: (1,0,0), (2,1,0), (3,2,0)
- Pop 1 → result = [1], push (4,0,1)
- Pop 2 → result = [1,2], push (5,1,1)
- Pop 3 → result = [1,2,3], push (6,2,1)
Continue until heap is empty → result = [1,2,3,4,5,6,7,8,9]
```

## Generic Boilerplate (Python):

```
import heapq
from typing import List

def kWayMerge(lists: List[List[int]]) -> List[int]:
    minHeap = []
    result = []

    # Step 1: Initialize heap with first element of each list
    for i in range(len(lists)):
        if lists[i]:
            heapq.heappush(minHeap, (lists[i][0], i, 0))

    # Step 2: Extract min, and insert the next element from that list
    while minHeap:
        val, list_idx, ele_idx = heapq.heappop(minHeap)
        result.append(val)

        if ele_idx + 1 < len(lists[list_idx]):
            next_val = lists[list_idx][ele_idx + 1]
            heapq.heappush(minHeap, (next_val, list_idx, ele_idx + 1))

    return result
```

## Applications on LeetCode:

### *1. LeetCode 23: Merge k Sorted Lists*

```
import heapq

class Solution:
    def mergeKLists(self, lists):
        minHeap = []
        for i, node in enumerate(lists):
            if node:
                heapq.heappush(minHeap, (node.val, i, node))
        dummy = ListNode(0)
```

```
        curr = dummy

        while minHeap:
            val, i, node = heapq.heappop(minHeap)
            curr.next = node
            curr = curr.next

            if node.next:
                heapq.heappush(minHeap, (node.next.val, i, node.next))

        return dummy.next
```

## *2. LeetCode 373: Find K Pairs with Smallest Sums*

```
import heapq
from typing import List

class Solution:
    def kSmallestPairs(self, nums1: List[int], nums2: List[int], k: int) -> List[List[int]]:
        if not nums1 or not nums2:
            return []

        minHeap = []
        result = []

        for i in range(min(len(nums1), k)):
            heapq.heappush(minHeap, (nums1[i] + nums2[0], i, 0))

        while minHeap and len(result) < k:
            val, i, j = heapq.heappop(minHeap)
            result.append([nums1[i], nums2[j]])

            if j+1 < len(nums2):
                heapq.heappush(minHeap, (nums1[i] + nums2[j+1], i, j+1))

        return result
```

## *3. LeetCode 378: Kth Smallest Element in a Sorted Matrix*

```
import heapq

class Solution:
    def kthSmallest(self, matrix, k):
        n = len(matrix)
        minHeap = []

        for i in range(min(n, k)):
            heapq.heappush(minHeap, (matrix[i][0], i, 0))

        count, number = 0, 0
        while minHeap:
            number, r, c = heapq.heappop(minHeap)
            count += 1
            if count == k:
                break

            if c + 1 < n:
                heapq.heappush(minHeap, (matrix[r][c+1], r, c+1))

        return number
```