# Merge k Sorted Lists

\*\*Description:\*\* Merge k sorted linked lists into one using divide and conquer, not heaps.

\*\*Problem Statement:\*\*  
In data processing or batch job scheduling, it's common to have multiple sorted logs (linked lists) from different sources. To prepare these logs for unified analysis, they must be merged into a single sorted list.  
  
You're given k sorted linked lists, each representing sorted data from independent processes. Your task is to merge all k lists into a single sorted linked list.  
  
Rather than using a heap or priority queue, implement a \*\*divide and conquer\*\* strategy (similar to merge sort) to merge them. Recursively merge pairs of lists until one final sorted list remains.

\*\*Input Format:\*\*  
A list of k linked lists, where each linked list is represented as a list of integers.

\*\*Constraints:\*\*  
0 <= k <= 10^4  
-10^4 <= ListNode.val <= 10^4

\*\*Output Format:\*\*  
Return a single sorted list containing all elements from the k linked lists.

\*\*Tags:\*\* linked list, divide and conquer, sorting, merge sort, recursion, intermediate

\*\*Sample Input:\*\*  
[[1,4,5], [1,3,4], [2,6]]

\*\*Sample Output:\*\*  
[1,1,2,3,4,4,5,6]

\*\*Test Cases:\*\*

* Test Case 1 Input:  
  [[1,4,5], [1,3,4], [2,6]]
* Expected Output:  
  [1,1,2,3,4,4,5,6]
* Test Case 2 Input:  
  [[1,2,3], [], [4,5]]
* Expected Output:  
  [1,2,3,4,5]
* Test Case 3 Input:  
  [[], [], []]
* Expected Output:  
  []
* Test Case 4 Input:  
  [[5], [1,2,3,4]]
* Expected Output:  
  [1,2,3,4,5]
* Test Case 5 Input:  
  [[10,20,30], [5,15,25], [1,2,3]]
* Expected Output:  
  [1,2,3,5,10,15,20,25,30]

# Majority Element Finder

\*\*Description:\*\* Find the element that appears more than n/2 times using divide and conquer.

\*\*Problem Statement:\*\*  
In distributed systems, it's important to reach a consensus on a common value, like a server ID or config version. If one value appears more than half the time, it's considered the majority.  
  
You are given an array of size n. Your task is to find the element that appears more than ⌊n / 2⌋ times.  
  
Use a \*\*divide and conquer\*\* approach, similar to merge sort. The key insight: if a majority element exists in the entire array, it must be the majority in at least one of the two halves.  
  
Solve this problem recursively by dividing the array and merging results to find the global majority.

\*\*Input Format:\*\*  
A list of integers representing the elements.

\*\*Constraints:\*\*  
1 <= nums.length <= 5 \* 10^4  
-10^9 <= nums[i] <= 10^9

\*\*Output Format:\*\*  
Return the majority element that appears more than ⌊n / 2⌋ times.

\*\*Tags:\*\* divide and conquer, arrays, majority element, recursion, intermediate

\*\*Sample Input:\*\*  
[2,2,1,1,1,2,2]

\*\*Sample Output:\*\*  
2

\*\*Test Cases:\*\*

* Test Case 1 Input:  
  [2,2,1,1,1,2,2]
* Expected Output:  
  2
* Test Case 2 Input:  
  [3,3,4]
* Expected Output:  
  3
* Test Case 3 Input:  
  [1,1,1,2,3,4,1]
* Expected Output:  
  1
* Test Case 4 Input:  
  [5,5,5,5,2,2,2]
* Expected Output:  
  5
* Test Case 5 Input:  
  [9,9,9,9,1,2,3,9]
* Expected Output:  
  9

# Josephus Survivor Position

\*\*Description:\*\* Find the last remaining person's position in a circle after eliminating every k-th.

\*\*Problem Statement:\*\*  
In theoretical computer science and game theory, the Josephus problem is a famous recursive problem. Given a group of people standing in a circle, eliminate every k-th person in sequence until only one remains.  
  
The goal is to determine the \*\*position\*\* of the survivor.  
  
This has applications in circular scheduling, memory allocation, and queue simulation.  
  
Implement a recursive solution to find the position of the last surviving person, starting with position 0. For k=2, a known closed-form exists, but your solution must work for any k.

\*\*Input Format:\*\*  
Two integers n and k — the number of people and the step size for elimination.

\*\*Constraints:\*\*  
1 <= n <= 10^5  
1 <= k <= 10^5

\*\*Output Format:\*\*  
Return an integer indicating the position (0-indexed) of the last survivor.

\*\*Tags:\*\* math, recursion, josephus, circular array, algorithm, intermediate

\*\*Sample Input:\*\*  
7 3

\*\*Sample Output:\*\*  
4

\*\*Test Cases:\*\*

* Test Case 1 Input:  
  7 3
* Expected Output:  
  4
* Test Case 2 Input:  
  5 2
* Expected Output:  
  2
* Test Case 3 Input:  
  6 1
* Expected Output:  
  5
* Test Case 4 Input:  
  10 2
* Expected Output:  
  4
* Test Case 5 Input:  
  1 1
* Expected Output:  
  0