Assignment 3a — Part 1: Merging k Sorted Linked Lists

Problem Statement: Implement the C function:

```
struct ListNode* mergeKLists(struct ListNode** lists, int listsSize);
```

that merges an array of k sorted singly-linked lists into a single sorted list.

Approach:

- Handle edge case: if `listsSize == 0`, return `NULL` immediately.
- Initialize a dummy head node (`dummy`) and set a tail pointer (`tail = &dummy`).
- Maintain an array `lists[0..k-1]` pointing to current heads of each list.
- *K-way merge:* Loop until all lists are empty:
  - Scan indices `0..listsSize-1` to find `minIdx` with the smallest head value.
  - If no non-empty list remains (`minIdx < 0`), break the loop.
  - Detach the node `minNode = lists[minIdx]` and advance that list (`lists[minIdx] = minNode->next`).
  - Append `minNode` to the result list (`tail->next = minNode`; `tail = minNode`).
- Return the merged list head: `dummy.next`.

Design Decisions & Trade-offs:

- Chose a simple linear scan across k heads for clarity and minimal code complexity, meeting the assignment requirement for an $O(N \times k)$ solution.
- Although a min-heap could achieve $O(N \log k)$ time, the overhead of additional data structures was unnecessary for expected input sizes and beyond the assignment's scope.
- No new nodes are allocated—existing nodes are relinked in-place, satisfying memory-safety and in-place merging expectations.

Complexity Analysis:

- *Time Complexity:* $O(N \times k)$, where N is the total number of nodes and $k =$ `listsSize`. Each of the N steps scans k list heads.
- *Space Complexity:* $O(1)$ auxiliary space (in-place) plus $O(k)$ for the head-pointer array; no additional node allocations.

Testing & Robustness:

- Verified correct merging for various cases: all lists empty, single non-empty list, lists of different lengths.
- Tested with increasing k and random values; output matches expected sorted order.
- Edge cases (e.g., listsSize = 0) handled as per assignment specification.