# Solution: Kth Smallest Number in M Sorted Lists

Let's solve the Kth Smallest Number in M Sorted Lists problem using the K-Way Merge pattern.

**We'll cover the following**

## Statement

Given an $m$ number of sorted lists in ascending order and an integer, `k`, find the $k^{th}$ smallest number among all the given lists.

Although there can be repeating values in the lists, each element is considered unique and, therefore, contributes to calculating the $k^{th}$ smallest element.

If `k` is greater than the total number of elements in the input lists, return the greatest element from all the lists and if there are no elements in the input lists, return 0.

**Constraints**:

- $1 \leq$ `m` $\leq 300$
- $0 \leq$ `list[i].length` $\leq 300$
- $-10^9 \leq$ `list[i][j]` $\leq 10^9$
- $1 \leq$ `k` $\leq 10^9$

## Solution

So far, you've probably brainstormed some approaches and have an idea of how to solve this problem. Let's explore some of these approaches and figure out which one to follow based on considerations such as time complexity and any implementation constraints.

### Naive approach

A naive approach to this problem can be to merge all the lists into a single list and then sort it to find the $k^{th}$ smallest number from it.

The time complexity of the naive approach is $O(n \log n)$, where $n$ is the total number of elements present in all the lists. Since sorting takes $O(n \log n)$, while searching up to $k^{th}$ number on the sorted list costs us $O(k)$.

### Optimized approach using k-way merge

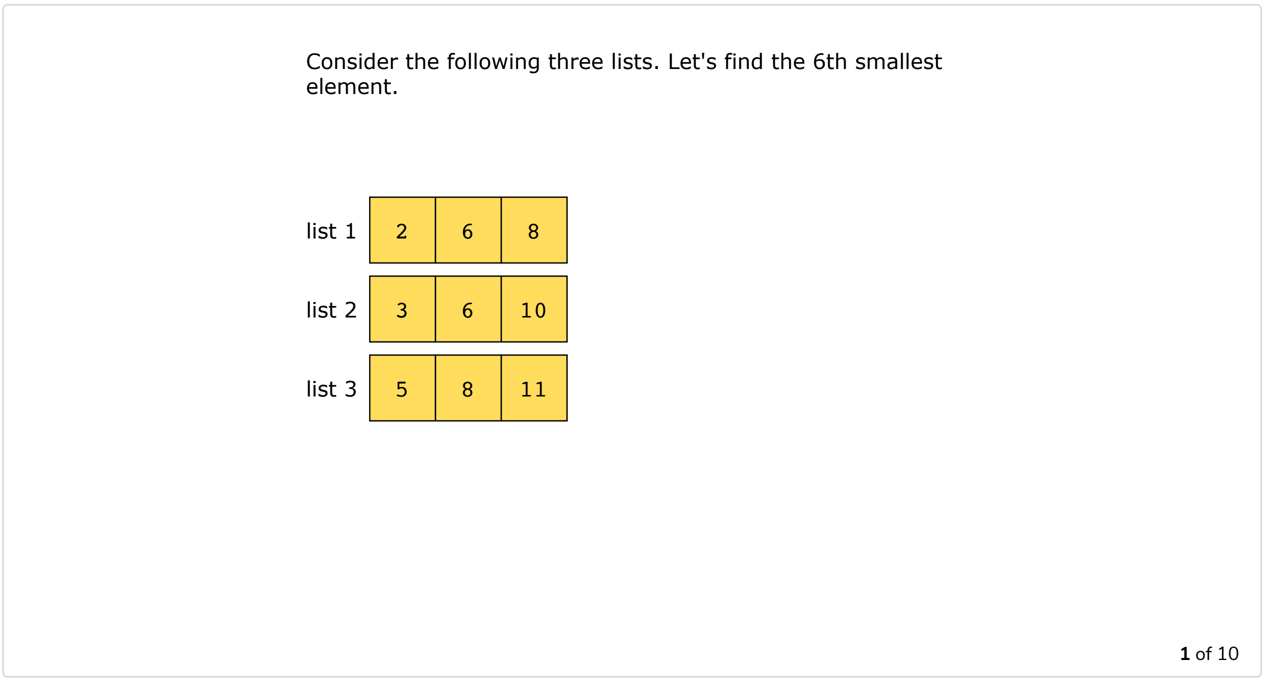
In a k-way merge pattern, we have k-sorted lists and imagine them as a single merged sorted list. We need to find the $k^{th}$ smallest element that will be the $k^{th}$ element in the merged and sorted list of all the lists.

Here's how we implement our algorithm using a min-heap to find the $k^{th}$ smallest element in $m$ sorted lists:

> **Note**: We'll implement the heap using the `PriorityQueue` class.

1. Push the first element of each list in the min-heap along with the index of the list and the index of the element in the list.
2. Pop the top element of the min-heap.
3. If the popped element has the next element in its list, push the next element of this list in the min-heap.
4. Repeat steps $2$ and $3$ until there are elements in the min-heap or we've removed $k$ elements from the heap.
5. The last element popped from the min-heap during this process is the required $k^{th}$ smallest element in the $m$ sorted lists.

The slides below illustrate how we would like the algorithm to run.



Consider the following three lists. Let's find the 6th smallest element.

Let's look at the code for this solution below:

Java

```java
import java.util.*;

class KSmallestNumber {
    public static int kSmallestNumber(List<List<Integer>> lists, int k) {
        // storing the length of lists to use it in a loop later
        int listLength = lists.size();
        // declaring a min-heap to keep track of smallest elements
        PriorityQueue<int[]> kthSmallest = new PriorityQueue<>((a, b) -> a[0] - b[0]);

        for (int index = 0; index < listLength; index++) {
            // if there are no elements in the input lists, continue
            if (lists.get(index).size() == 0) {
                continue;
            } else {
                // placing the first element of each list in the min-heap
                kthSmallest.offer(new int[] {lists.get(index).get(0), index, 0});
            }
        }

        // set a counter to match if our kth element
        // equals to that counter, return that number
        int numbersChecked = 0, smallestNumber = 0;
        while (!kthSmallest.isEmpty()) {  // iterating over the elements pushed in our min-heap
            // get the smallest number from top of heap and its corresponding list and index
            int[] smallest = kthSmallest.poll();
            smallestNumber = smallest[0];
            int listIndex = smallest[1];
            int numIndex = smallest[2];
```

Kth Smallest Number in M Sorted Lists

### Solution summary

We can summarize our solution in the following steps.

1. Push the first element of each list in the min-heap.
2. Pop the top element of the min-heap, and keep track of the list index and the element index in the list. Now, push the next element of the popped element from the list if the element index is not the last index in the list.
3. Repeat step 2 until we have popped $k$ elements from the heap. If we have popped $k$ elements, then the $k^{th}$ element popped in this process is the $k^{th}$ smallest element.

### Time complexity

The time complexity of the first step is

- The cost of pushing $m$ elements in the heap is as follows:

$$\log 1 + \log 2 + \log 3 + \cdots + \log m = \log(1 * 2 * 3 * \cdots * m) = \log(m!)$$

As per Stirling's approximation, $O(\log m!) \approx O(m \log m)$.

So, the time complexity of the first loop is $O(m \log m)$.

- In the `while` loop, we pop and push on to the heap $k$ number of times until we find the $k^{th}$ smallest number. So, the time complexity of this step is $O(k \log m)$.

So, the total time complexity of this solution is

$$O(m \log m + k \log m) = O((m + k) \log m)$$

### Space complexity

The space complexity is $O(m)$, where $m$ is the total number of elements in the heap.

← Back
Kth Smallest Number i...

Next →
Find K Pairs with Smal...

Mark as Completed