Solution: Kth Smallest Element in a Sorted Matrix

Let's solve the Kth Smallest Element in a Sorted Matrix problem using the K-Way Merge pattern.

We'll cover the following

Statement

Find the k^{th} smallest element in an $(n \times n)$ matrix, where each row and column of the matrix is sorted in ascending order.

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

Constraints:

- `n` $==$ `matrix.length`
- `n` $==$ `matrix[i].length`
- $1 \leq$ `n` ≤ 300
- $-10^9 \leq$ `matrix[i][j]` $\leq 10^9$
- $1 \leq$ `k` $\leq n^2$

Solution

A key observation when tackling this problem is that the matrix is sorted along rows and columns. This means that whether we look at the matrix as a collection of rows or as a collection of columns, we see a collection of sorted lists.

In a k-way merge pattern, we have k-sorted arrays and imagine them as a single, merged sorted array. We need to find the k^{th} smallest element, which is the k^{th} element in the merged and sorted array of all the elements in the matrix.

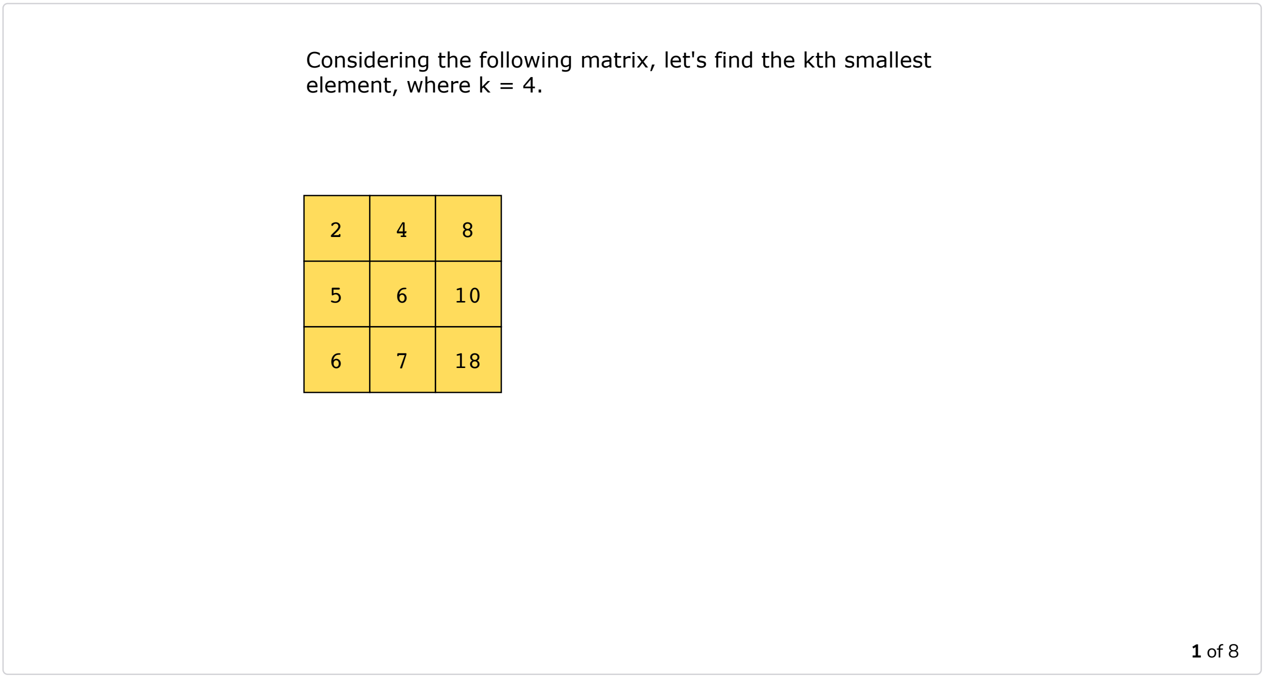
> **Note**: We'll implement the heap using the `PriorityQueue` class.

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

1. We push the first element of each row of the matrix in the min-heap, storing each element along with its row and column index.
2. Remove the top (root) of the min-heap.
3. If the popped element has the next element in its row, push the next element in the heap.
4. Repeat steps 2 and 3 as long as there are elements in the min-heap, and stop as soon as we've popped k elements from it.

5. The last popped element in this process is the k^{th} smallest element in the matrix.

The slides below demonstrate the working of the above-mentioned algorithm:



Considering the following matrix, let's find the kth smallest element, where k = 4.

2	4	8
5	6	10
6	7	18

1 of 8

We can see the code of this solution below:

Java

```java
import java.util.*;

class KthSmallest {

    public static int kthSmallestElement(int[][] matrix, int k) {
        // storing the number of rows in the matrix to use it in later
        int rowCount = matrix.length;
        // declaring a min-heap to keep track of smallest elements
        PriorityQueue<int[]> minHeap = new PriorityQueue<>((a, b) -> a[0] - b[0]);

        for (int i = 0; i < rowCount; i++) {
            // pushing the first element of each row in the min-heap
            // The offer() method pushes an element into an existing heap
            // in such a way that the heap property is maintained.
            minHeap.offer(new int[]{matrix[i][0], i, 0});
        }

        int numbersChecked = 0;
        int smallestElement = 0;
        // iterating over the elements pushed in our minHeap
        while (!minHeap.isEmpty()) {
            // get the smallest number from top of heap and its corresponding row and column
            int[] curr = minHeap.poll();
            smallestElement = curr[0];
            int rowIndex = curr[1];
            int colIndex = curr[2];
            numbersChecked++;
            // when numbersChecked equals k, we'll return smallestElement
```

Kth Smallest Element in a Sorted Matrix

The time complexity of the first step is:

- $O(n)$ for iterating the first element of each row, where n is the number of rows of the matrix.
- The cost of pushing n elements in the heap is as follows:

$$\log 1 + \log 2 + \log 3 + \ldots + \log n = \log(1 * 2 * 3 * \ldots * n) = \log(n!)$$

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

Therefore, the time complexity of the first loop is $O(n \log n)$.

- In the while-loop, we pop and push k elements in the heap until we find the k^{th} smallest element. Therefore, the time complexity of this step is $O(k \log n)$.

Overall, the total time complexity of this solution is $O(n \log n + (k \log n)) = O((n + k) \log n)$

Space complexity

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

Back
Kth Smallest Element ...

Next
Median of Two Sorted...

✓ Mark as Completed