

Kth Smallest Number in a Sorted Matrix (Hard)

We'll cover the following ^

- Problem Statement
- Try it yourself
- Solution
- Code
 - Time complexity
 - Space complexity
- An Alternate Solution
 - Time complexity
 - Space complexity

Problem Statement

Given an $N * N$ matrix where each row and column is sorted in ascending order, find the Kth smallest element in the matrix.

Example 1:

```
Input: Matrix=[
  [2, 6, 8],
  [3, 7, 10],
  [5, 8, 11]
],
K=5
Output: 7
Explanation: The 5th smallest number in the matrix is 7.
```

Try it yourself

Try solving this question here:

JavaPython3JS C++

```
1 def find_Kth_smallest(matrix, k):
2     number = -1
3     # TODO: Write your code here
4     return number
5
6
7 def main():
8     print("Kth smallest number is: " +
9         str(find_Kth_smallest([[2, 6, 8], [3, 7, 10], [5, 8, 11]], 5)))
10
11
12 main()
13
```

RunSaveReset

Solution

This problem follows the **K-way merge** pattern and can be easily converted to [Kth Smallest Number in M Sorted Lists](#). As each row (or column) of the given matrix can be seen as a sorted list, we essentially need to

find the Kth smallest number in 'N' sorted lists.

Code

Here is what our algorithm will look like:

```
1 from heapq import *
2
3
4 def find_Kth_smallest(matrix, k):
5     minHeap = []
6
7     # put the 1st element of each row in the min heap
8     # we don't need to push more than 'k' elements in the heap
9     for i in range(min(k, len(matrix))):
10         heappush(minHeap, (matrix[i][0], 0, matrix[i]))
11
12     # take the smallest(top) element form the min heap, if the running count is equal to k'
13     #return the number
14     # if the row of the top element has more elements, add the next element to the heap
15     numberCount, number = 0, 0
16     while minHeap:
17         number, i, row = heappop(minHeap)
18         numberCount += 1
19         if numberCount == k:
20             break
21         if len(row) > i+1:
22             heappush(minHeap, (row[i+1], i+1, row))
23     return number
24
25
26 def main():
27     print("Kth smallest number is: " +
28         str(find_Kth_smallest([[2, 6, 8], [3, 7, 10], [5, 8, 11]], 5)))
29
30
31 main()
32
```

Time complexity

First, we inserted at most 'K' or one element from each of the 'N' rows, which will take $O(\min(K, N))$. Then we went through at most 'K' elements in the matrix and remove/add one element in the heap in each step. As we can't have more than 'N' elements in the heap in any condition, therefore, the overall time complexity of the above algorithm will be $O(\min(K, N) + K * \log N)$.

Space complexity

The space complexity will be $O(N)$ because, in the worst case, our min-heap will be storing one number from each of the 'N' rows.

An Alternate Solution

Since each row and column of the matrix is sorted, is it possible to use **Binary Search** to find the Kth smallest number?

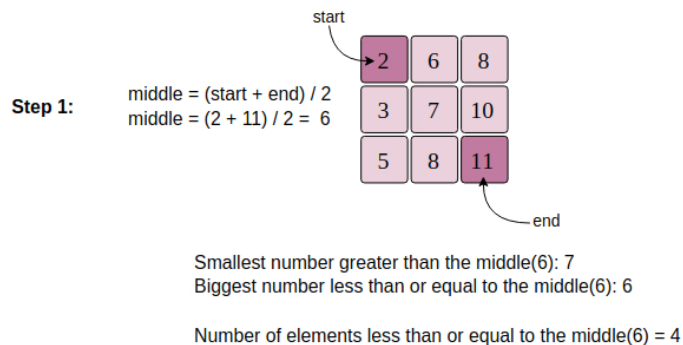
The biggest problem to use **Binary Search**, in this case, is that we don't have a straightforward sorted array, instead, we have a matrix. As we remember, in **Binary Search**, we calculate the middle index of the search space ('1' to 'N') and see if our required number is pointed out by the middle index; if not we either search in the lower half or the upper half. In a sorted matrix, we can't really find a middle. Even if we do consider some index as middle, it is not straightforward to find the search space containing numbers bigger or smaller than the number pointed out by the middle index.

An alternative could be to apply the **Binary Search** on the "number range" instead of the "index range". As we know that the smallest number of our matrix is at the top left corner and the biggest number is at the bottom right corner. These two numbers can represent the "range" i.e., the **start** and the **end** for the **Binary**

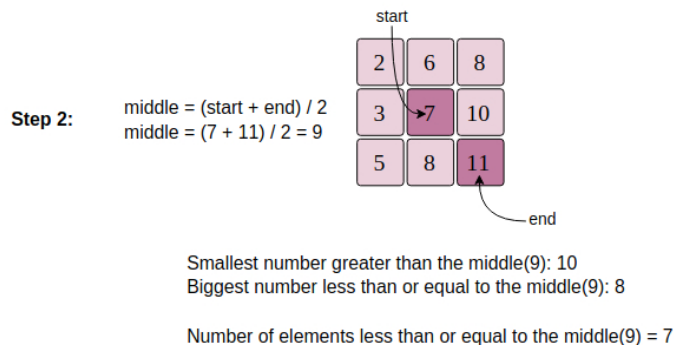
Search. Here is how our algorithm will work:

1. Start the **Binary Search** with `start = matrix[0][0]` and `end = matrix[n-1][n-1]`.
2. Find `middle` of the `start` and the `end`. This `middle` number is NOT necessarily an element in the matrix.
3. Count all the numbers smaller than or equal to `middle` in the matrix. As the matrix is sorted, we can do this in $O(N)$.
4. While counting, we can keep track of the “smallest number greater than the `middle`” (let’s call it `n1`) and at the same time the “biggest number less than or equal to the `middle`” (let’s call it `n2`). These two numbers will be used to adjust the “number range” for the **Binary Search** in the next iteration.
5. If the count is equal to ‘K’, `n1` will be our required number as it is the “biggest number less than or equal to the `middle`”, and is definitely present in the matrix.
6. If the count is less than ‘K’, we can update `start = n2` to search in the higher part of the matrix and if the count is greater than ‘K’, we can update `end = n1` to search in the lower part of the matrix in the next iteration.

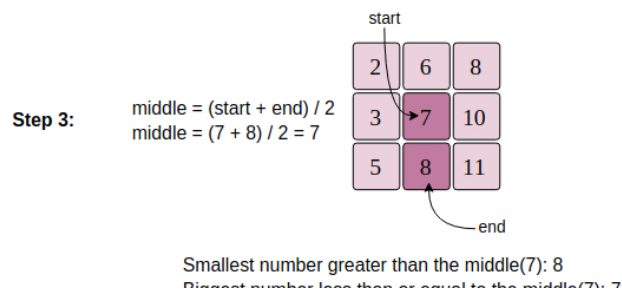
Here is the visual representation of our algorithm:



As there are only 4 elements less than or equal to the middle, and we are looking for the 5th smallest number, so let's search higher and **update our 'start' to the smallest number greater than the middle**.



As there are 7 elements less than or equal to the middle, and we are looking for the 5th smallest number, so let's search lower and **update our 'end' to the biggest number less than or equal to the middle**.



Biggest number less than or equal to the middle(7). /

Number of elements less than or equal to the middle(7) = 5

As there are 5 elements less than or equal to the middle therefore '7', which is the biggest number less than or equal to the middle, is our required number

Here is what our algorithm will look like:

Java Python3 C++ JS

```
1 def find_Kth_smallest(matrix, k):
2     n = len(matrix)
3     start, end = matrix[0][0], matrix[n - 1][n - 1]
4     while start < end:
5         mid = start + (end - start) / 2
6         smaller, larger = (matrix[0][0], matrix[n - 1][n - 1])
7
8         count, smaller, larger = count_less_equal(matrix, mid, smaller, larger)
9
10        if count == k:
11            return smaller
12        if count < k:
13            start = larger # search higher
14        else:
15            end = smaller # search lower
16
17    return start
18
19
20 def count_less_equal(matrix, mid, smaller, larger):
21     count, n = 0, len(matrix)
22     row, col = n - 1, 0
23     while row >= 0 and col < n:
24         if matrix[row][col] > mid:
25             # as matrix[row][col] is bigger than the mid, let's keep track of the
26             # smallest number greater than the mid
27             larger = min(larger, matrix[row][col])
28             row -= 1
29         else:
30             # as matrix[row][col] is less than or equal to the mid, let's keep track of the
31             # biggest number less than or equal to the mid
32             smaller = max(smaller, matrix[row][col])
33             count += row + 1
34             col += 1
35
36     return count, smaller, larger
37
38
39 def main():
40     print("Kth smallest number is: " +
41         str(find_Kth_smallest([[1, 4], [2, 5]], 2)))
42
43     print("Kth smallest number is: " +
44         str(find_Kth_smallest([[-5]], 1)))
45
46     print("Kth smallest number is: " +
47         str(find_Kth_smallest([[2, 6, 8], [3, 7, 10], [5, 8, 11]], 5)))
48
49     print("Kth smallest number is: " +
50         str(find_Kth_smallest([[1, 5, 9], [10, 11, 13], [12, 13, 15]], 8)))
51
52
53 main()
```

Run Save Reset

Time complexity

The **Binary Search** could take $O(\log(\max - \min))$ iterations where 'max' is the largest and 'min' is the smallest element in the matrix and in each iteration we take $O(N)$ for counting, therefore, the overall time complexity of the algorithm will be $O(N * \log(\max - \min))$.

Space complexity

The algorithm runs in constant space $O(1)$.


[← Back](#)

[Next →](#)

Kth Smallest Number in M Sorted List...

Smallest Number Range (Hard)

 Completed

 Report an Issue  Ask a Question