Kth Smallest Number in a Sorted Matrix (Hard)

We'll cover the following

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:

Python3

```
def find_Kth_smallest(matrix, k):
  number = -1
  # TODO: Write your code here
  return number


def main():
  print("Kth smallest number is: " +
        str(find_Kth_smallest([[2, 6, 8], [3, 7, 10], [5, 8, 11]], 5)))


main()
```

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:

Python3

```
from heapq import *


def find_Kth_smallest(matrix, k):
    minHeap = []

    # put the 1st element of each row in the min heap
    # we don't need to push more than 'k' elements in the heap
    for i in range(min(k, len(matrix))):
        heappush(minHeap, (matrix[i][0], 0, matrix[i]))

    # take the smallest(top) element form the min heap, if the running count
    # if the row of the top element has more elements, add the next element
    numberCount, number = 0, 0
    while minHeap:
        number, i, row = heappop(minHeap)
        numberCount += 1
        if numberCount == k:
            break
        if len(row) > i+1:
            heappush(minHeap, (row[i+1], i+1, row))
    return number


def main():
    print("Kth smallest number is: " +
          str(find_Kth_smallest([[2, 6, 8], [3, 7, 10], [5, 8, 11]], 5)))
```

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, th re overall time complexity of the above algorithm will be $O(min(K, N) + K * logN)$.

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?

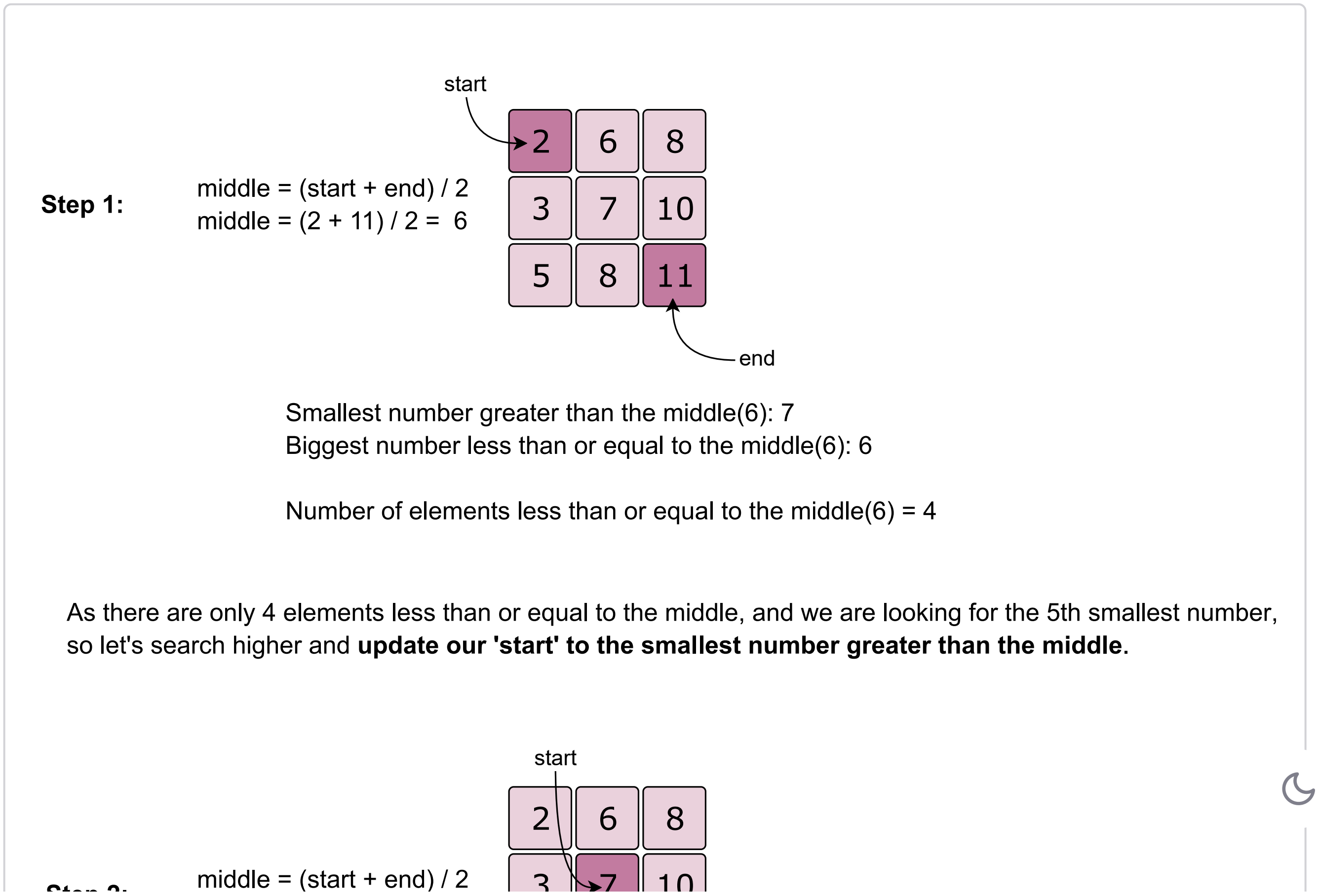
The biggest problem to use **Binary Search**, in this case, is that we don’t have a straightforward sorted array, instead, we have a matrix. As we remember, in **Binary Search**, we calculate the middle index of the search space (‘1’ to ‘N’) and see if our required number is pointed out by the middle 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` i m A 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’, `n2` 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:



Step 1:

middle = (start + end) / 2
middle = (2 + 11) / 2 = 6

Smallest number greater than the middle(6): 7
Biggest number less than or equal to the middle(6): 6

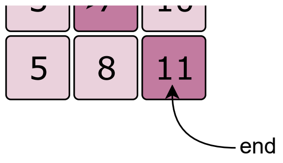
Number of elements less than or equal to the middle(6) = 4

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.**

middle = (start + end) / 2

Step 2:

middle = (7 + 11) / 2 = 9



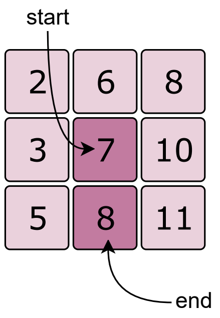
Smallest number greater than the middle(9): 10
Biggest number less than or equal to the middle(9): 8

Number of elements less than or equal to the middle(9) = 7

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.**

Step 3:

middle = (start + end) / 2
middle = (7 + 8) / 2 = 7



Smallest number greater than the middle(7): 8
Biggest number less than or equal to the middle(7): 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:

Python3

```
def find_Kth_smallest(matrix, k):
  n = len(matrix)
  start, end = matrix[0][0], matrix[n - 1][n - 1]
  while start < end:
    mid = start + (end - start) / 2
    smaller, larger = (matrix[0][0], matrix[n - 1][n - 1])

    count, smaller, larger = count_less_equal(matrix, mid, smaller, larger)

    if count == k:
      return smaller
    if count < k:
      start = larger  # search higher
    else:
      end = smaller  # search lower

  return start


def count_less_equal(matrix, mid, smaller, larger):
  count, n = 0, len(matrix)
  row, col = n - 1, 0
  while row >= 0 and col < n:
    if matrix[row][col] > mid:
      # as matrix[row][col] is bigger than the mid, let's keep track of the
      # smallest number greater than the mid
```

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

Kth Smallest Number in M Sorted List...

Next →

Smallest Number Range (Hard)

✓ Mark as Comple

Report an Issue