Summary

Time Complexity: O(k), where k is the number of lists, since we are generating k elements **Space Complexity: O(k)**, since we would have elements relative to k on the heap

Problem Pattern

1. First let's understand what the problem is asking for. It is asking for the Kth smallest number that is of the form 3^x * 5^y * 7^z
2. What is the first approach that comes to your mind? We know that the biggest this kth number could be is 3^k * 5^k * 7^k. So we can compute this for all values of x, y and z between 0 and k. Then from this list we can pick the kth smallest value. But this approach has a very bad runtime.
3. What optimizations can we make?
4. To get some perspective, make a list of some numbers that would be a part of your list above. Do you see a pattern for what the next value will be? It will be one of these:

3 * (some previous number in the list)
5 * (some previous number in the list)
7 * (some previous number in the list)

Problem Approach

1. We will have 1, 3, 5, 7, 9, 15, 21, 25, 27, 35 , 45 … The way we do it is we first get 3, 5 and 7, then we multiply 3 with 3, 5, 7 and then 5 with 3, 5, 7 and 7 with 3,5,7, we do not know the order of the result, but what we do know is that 3^(i+1) * 5^j * 7^k > 3^i *5^j * 7^k, same for 5 and 7.
2. So we can have 3 queues Q3, Q5 and Q7, to store the elements of factor 3, 5 and 7, and a result to count the number of elements. To get the min, we only need to look at the fronts of each queue: y = min(Q3.head(), Q5.head(), Q7.head())
3. The process we will follow is:

1. Initialize our queues Q3, Q5, and Q7

2. Insert 1*3,1* 5 and 1*7 into Q3, Q5, and Q7 respectively.

3. Let x be the minimum element in Q3, Q5, and Q7

4. If x was found in:

    Q3 -> append x*3, x*5 and X*7 to Q3, Q5, and Q7, Remove x from Q3.

    Q5 -> append x*5 and x*7 to Q5 and Q7, Remove x from Q5.

    Q7 -> only append x*7 to Q7. Remove x from Q7.

5. Repeat steps 3 - 4 until we've found k elements.

Problem Pseudocode

```
int getKthMagicNumber(int k) {

    if (k < 0)

        return 0

    int val = 0

    new Queue<Integer> queue3
    new Queue<lnteger> queue5
    new Queue<Integer> queue7


    queue3.add(1);


    for (i = 0 to k) {

        int v3 = queue3.size() > 0 ? queue3.peek() : Integer.MAX_VALUE
        int v5 = queue5.size() > 0 ? queue5,peek() : Integer.MAX_VALUE
        int v7 = queue7.size() > 0 ? queue7.peek() : Integer.MAX_VALUE


        val = Math.min(v3, Math.min(v5, v7))



        if (val == v3) {

            queue3.remove()

            queue3.add(3 * val)

            queue5.add(5 * val)

        } else if (val == v5) {

            queue5.remove()

            queue5.add(5 * val)

        } else if (val == v7) {

            queue7.remove()

        }

        queue7.add(7 * val)

    }

    return val

}
```

Reference: Cracking the Coding Interview by Gayle Laakmann McDowell

Alternate Approaches

1. We know that the biggest this kth number could be is 3^k * 5^k * 7^k. So, the naive way of doing this is to compute 3^a * 5^b * 7^c for all values of a, b, and c between 0 and k. We can throw them all into a list, sort the list, and then pick the kth smallest value.
2. We can have a min heap priority_queue for the given approach as well.

The idea is to push the first ugly number which is 1 into priority_queue and at every iteration take the top of priority_queue and push all the multiples of that top into priotiy_queue.
So in the first iteration, 1 is at the top so 1 is popped and 1 * 3, 1 * 5, 1 * 7 are pushed. In the second iteration, min is 3, so it is popped and 3 * 3, 3 * 5, 3 * 7 is pushed and so on.

3. Dynamic Programming: A time efficient solution with O(n) extra space (similar to shared approach without using queue). The ugly-number sequence is 1, 3, 5, 7, 9, 15, 21, 25, 27, 35 , 45, etc.

Because every number can only be divided by 3, 5, 7, we split the sequence to three groups as below

```
(1) 3×1, 3×3, 3×5, …

(2) 5×1, 5×3, 5×5, …

(3) 7×1, 7×3, 7x5 …
```

We find that every subsequence is the ugly-sequence itself (1, 3, 5, 7, 9…) multiplied by 3, 5, 7. Then we use a similar merge method as merge sort, to get every ugly number from the three subsequences. Every step we choose the smallest one, and move one step after.

1 Declare an array for numbers: ugly[n]

2 Initialize first number: ugly[0] = 1

3 Initialize three array index variables i3, i5, i7 to point to

1st element of the ugly array:

i3 = i5 = i7 = 0;

4 Initialize 3 choices for the next number:

next_mulitple_of_3 = ugly[i3]*3

next_mulitple_of_5 = ugly[i5]*5

next_mulitple_of_7 = ugly[i7]*7

5 Now go in a loop to fill all such numbers till k