Summary

Time Complexity: O(N), where N is the length of the input sequence since our operations depend on that length directly.
Space Complexity: O(N), since the recursion call stack also is proportional to the length.

Problem Pattern

1. You basically need to find the Kth permutation from a list of all permutations that is sorted. Can you do this without finding all possible permutations?
2. We know that n! is the number of permutations for a sequence of size n. Then, if we fix the first element, total permutations of remaining elements would be (n-1)!
3. Consider the example string [ABCD]. Then you can divide your permutations in blocks, each starting with one of the letters in the set. There will be 6 permutations in each group.

```
A → BCD

        BDC

        CBD

        CDB

        DBC

        DCB ... similarly for B, C and D
```

4. Can blocks be further divided? Can you determine which blocks the Kth permutation will lie in? Think about a recursive approach.

Problem Approach

1. If we are given K and we somehow guess which block it's going to lie in, that will help us find at least the first element.

Similarly, within that block, if we can identify a sub-block where k resides, it will help us find the second element.
We can do this recursively until we run out of options.

2. Say your sequence is [ABCD] and K = 8,

First group will have 6 permutations each, since K=8, our required permutation will lie in the 2nd group.

```
A...
```

```
B...
```

```
C...
```

```
D...
```

Now we see the sub-groups for B, each sub-group will have 2 permutations. Since K=8, our required permutation will lie in the first group.

```
B A...
```

```
B A...
```

```
B C...
```

```
B C...
```

```
B D...
```

```
B D...
```

Now we see the sub-groups for B A, each sub-group will have 1 permutation. Since K=8, our required permutation will be the 2nd one one this sub-group.

```
B A C..
```

```
B A D..
```

There is only one possibility now, which is B A D C. This is our final answer.

3. This can be solved recursively as well as iteratively. Recursive solution will need more memory.

Problem Pseudocode

```
void findKthPermutation(List v, k, result) {
    if (v is Empty)
        return

    n = v.size()
    // count is number of permutations starting with first character
    count = factorial(n - 1)
    selected = (k - 1) / count
    append(v(selected)) to result
    v.remove(selected)

    k = k - (count * selected);
    findKthPermutation(v, k, result)
}

String getPermutation(n, k) {
    List<Character> v
    char c = 'A'
```

```
    for (i = 1 to n)
        v.add(c)
        c++

    result = new String
    findKthPermutation(v, k, result)
    return result
}
```

Alternate Approaches

1. You could find all possible permutations, store them in an array and then return the Kth permutation from there. This is a naive approach and has high time and space complexity.
2. Or you could maintain a continuous count of permutations as you generate them and stop at the Kth permutation. Even here, in the worst case you would be generating all the permutations, so the time complexity is high.