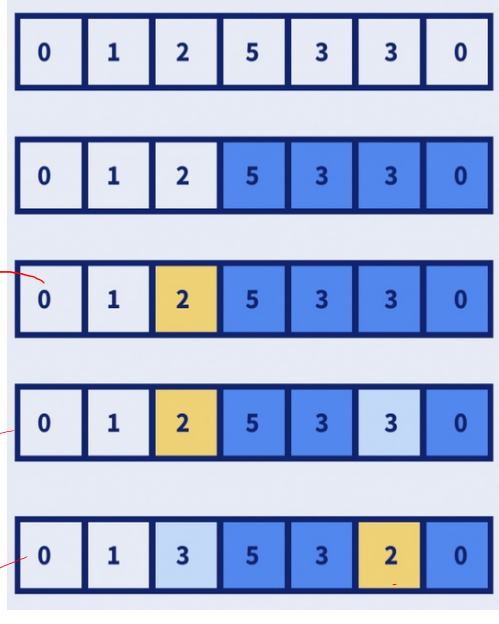
Lexicographic Order Permutation

Releated Questions:

1. Print all the permutation of a string in lexicographic order ?
2. Get Kth permutation of a string in lexicographic order ?
3. Get next greater number/next greater lexicographic permutation ?



PROPERTY_1: If in a number all the digits are sorted in ascending order from right side, then there cannot be any greater number by using those digits.

Example : 5332100 ; sorted in ascending order from right side.

PROPERTY_2 : If in a number all the digits are sorted in descending order from right side, then there cannot be any smaller number by using those digits.

Example : 0012335 ; sorted in descendig order from right side.

3. Get next greater number/next greater lexicographic permutation ?

Step1 : Identify Pivot

Traverse the array from right side and stop at the first element which is not in ascending order.

Example : 2 is pivot in 0125330

Question: Why we are traversing in ascending order from right side ? : Because of PROPERTY_1.

Step2: Find Successor of Pivot

Find the next greater digit than the Pivot among digits present in right-side of the pivot.

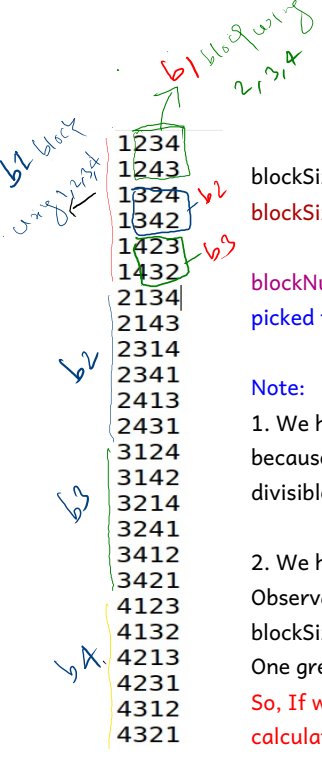
Note : Since right side of the pivot contains all the digits in sorted order, so we can apply binary serach.

Step 3: Swap the Pivot with Successor. Because we want just next greater element which is only possiple by replacing pivot with successor.

Step4: Now reverse sort(descending order from right side) all the elements lying in right-side of pivot position.

Note : Since right side of the pivot position is already in sorted order, so just need to reverse the element to get them in descending order.

Question: Why do we sort elements lying right-side of pivot position in descending order ? : Because of PROPERTY_1.



Kth Lexicographic Order Permutation

blockSize = n!/input_size = 4*3*2*1/4 = 3*2*1 = 3!
blockSize = n!/input_size = (n-1)!

blockNumber = k/blockSize; determines the character to be picked from input to fix the current position in output array.

Note:

1. We have picked blockNumber on 0-index based counting because integer divsion will give floor value for 'K' which is not divisible by blockSize.

2. We have picked 'K' on 0-index based counting. i.e. (K=K-1)
Observation : If 'K' is 1-index based and is divisible by blockSize (i.e. k%blockSize==0), caluclated blockNumber is One greater than actual blockNumber.
So, If we pick 'K' on 0-based index then both the calculated_blockNumber and actual_blockNumber are same.

This is a recursive algorithm, this is why to fix each position in output array we need to reset K and blockSize

```
private static char[] getKthLexicographicPermutation(List<Character> input, int k) {
    char[] output = new char[input.size()];

    int blockSize = factorial(input.size() - 1);

    k = k - 1;

    for (int i = 0; i < output.length; i++) {
        int blockNumber = k / blockSize;

        output[i] = input.get(blockNumber);
        input.remove(blockNumber);
        if (input.size() == 0)break;

        k = k % blockSize;
        blockSize = blockSize / input.size();
    }
    return output;
}
```

Time Complexity : O(n) + O(n^2)