Lexicographic Permutations

Define Lexicographic:

In mathematics, the lexicographic or lexicographical orders are a generalization of the way words are alphabetically ordered based on the alphabetical order of their component letters. This generalization consists primarily of defining a total order on the sequences (in this case: String) of elements of a finite totally ordered set, often called an alphabet.

How do we find lexicographic permutation?

There are many ways to find lexicographic permutation, the total possible combination can be found by

$$\frac{N!}{a!+b!...z!}$$ Where *N* is the total number of the element in the array, a...z is the number of repetition of a single element in the array.

For example: if an array was populated with: *[1,1,2,3]*, then the equation will be $\frac{4!}{2!} = \frac{24}{2} = 12$, this means the total possible permutations is 12.

How do we find all the elements of the permutation?

For example, we have an array: [0,1,2,3], we can construct a recursive tree, but first let's find the total number of elements which can be found by 4! = 24, which indicate we will have 24 permutations.

[Tree](Appendix A)

So as we can see from the tree, we have 24 permutations and all of them are ordered numerally. If we would like to find the 4th permutation, it would be 0231

How do we find the lexicographic permutations of '01234567890' at the 1 millionth place?

First, let's find the total possible permutations:

$$\frac{10!}{1!} = 3{,}628{,}800$$

Wow, that number is really big, if we want to find one of the permutations, that's a waste of computing resources. So Let's see if we can simplify this problem.

We know that we need to find the 1,000,000th permutations and every branch of the recursion is the total number of elements' factorial.***Heads up*** the place we need to find is 999,999 because the original permutation will carry on(01234567890). So Let's check which branch is 999,999 is on by using the place divided by a total amount of possible permutations under one branch and rounding down(**FLOOR**), luckily we don't have to do this and it's by default Java's typecasting method of choice.

$$\frac{999{,}999}{9!} \approx 2$$

Looks like our 999,999 is must be under the 2nd branch, we can take the current place then minus the current depth times the branch which can get the remaining possible permutations in between the branch.

$$999{,}999 - 725{,}760 = 274239$$

We got 274239 permutations in between, so let's cross out the index 2 in the string, and append into our new string.

S: 01~~2~~3456789 N:2

Let's do this until we ran to 0!:

274239/8! = 6 x 8! = 241920; 274239-241920 = 32319;
S: 013456~~7~~89 N:27
274239/8! = 6 x 8! = 241920; 274239-241920 = 32319;
S: 013456~~7~~89 N:27
32319/7! = 6 x 7! = 30240; 32319-30240 = 2079;
S: 01345~~6~~8~~9~~ N:278
2079/6! = 2 x 6! = 30240; 2079-1440 = 639;
S: 01~~3~~4569 N:2783
639/5! = 5 x 5! = 600; 639-600 = 39;
S: 01456~~9~~ N:27839
39/4! = 1 x 4! = 24; 39-24 = 15;
S: 0~~1~~456 N:278391
15/3! = 2 x 3! = 12; 15-12 = 3;
S: 04~~5~~6 N:2783915
3/2! = 1 x 2! = 2; 3-2 = 1;
S: 0~~4~~6 N:27839154
1/1! = 1 x 1! = 1; 1 -1 = 0;
S: 0~~6~~ N:278391546
0/0! = 0 x 0! = 0; 0 -0 = 0;
S: ~~0~~ N:2783915460

As you can see, we come up with an answer without a computer. Now let's move on to code, *(Code)[Appendix B]*

Appendix A

Tree

Appendix B

Code

```java
public class Main {

    // Factorial Function
    private static int factorial(int n)
    {
        if (n <= 1)
            return 1;
        return n*factorial(n-1);
    }

    public static void main(String[] args)
    {
        String str = "0123456789"; // Construct the string
        StringBuilder result = new StringBuilder(); // Because String is immutable, using StringBuilder to mutate the string
        int place = 1000000 - 1; // one millionth - 1 because the original permutation will carry on.

        // Given that string length as i -1 (because of index), irritate each element
        // from the beginning 'til i is equal to 0
        for(int i = str.length() - 1; i >= 0; i--)
        {
            int pos = place / factorial(i); // get the nearest position of the array
            place -= factorial(i) * pos; // current depth! * position multiplier to get the remaining possible permutations
            result.append(str.charAt(pos)); // As described, String is immutable so StringBuilder can be mutated so .append to the end of the string
            System.out.println(place);
            str = str.substring(0, pos) + str.substring(pos +1); // remove the element
        }
        System.out.println(result);
    }
}
```