**Inverse a Number**

**Question:**

1. You are given a number following certain constraints.
2. The key constraint is if the number is 5 digits long, it'll contain all the digits from 1 to 5 without missing any and without repeating any. e.g. 23415 is a 5 digit long number containing all digits from 1 to 5 without missing and repeating any digit from 1 to 5.Take a look at few other valid numbers - 624135, 81456273, etc. Here are a few invalid numbers - 139, 7421357, etc.
3. The inverse of a number is defined as the number created by interchanging the face value and index of digits of the number. e.g. for 426135 (reading from right to left, 5 is in place 1, 3 is in place 2, 1 is in place 3, 6 is in place 4, 2 is in place 5 and 4 is in place 6), the inverse will be 416253 (reading from right to left, 3 is in place 1, 5 is in place 2,2 is in place 3, 6 is in place 4, 1 is in place 5 and 4 is in place 6) More examples - the inverse of 2134 is 1243 and inverse of 24153 is 24153
4. Take as input number "n", assume that the number will follow constraints.
5. Print it's inverse.

**Input format:**

"n" where n is an integer, following constraints defined above.

**Output format:**

"i", the inverted number

**Constraints:**

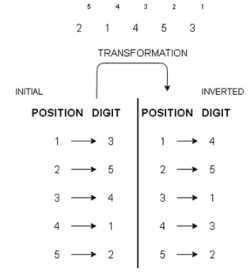
1 <= n < 10^8, and following other constraints defined above. Let us first understand the question statement properly:

If we are given a number **613254** satisfying all the constraints given, we need to invert it as such it gives us the resulting output of **621435.**

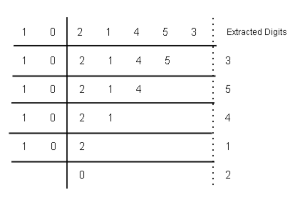
****

**Solution Approach:**

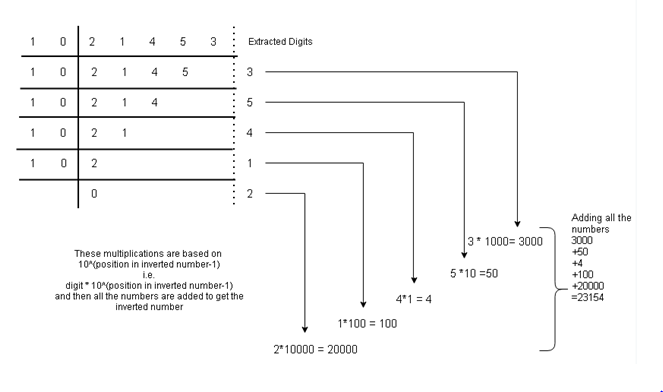
For simplicity, we take a smaller number for instance. Let it be **21453**. We construct a table that correlates the digits in the number with their position index.

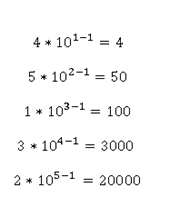


So we start off by dividing the given number by 10 and extracting the digits from the 1st position of the given number. We observe that by continuous division by 10, we strip off the last digit one by one, and to fit in the required



conditions for forming an inverted number, we can see, that we will have to form the new number by multiplying the extracted digits by a power of 10 to place them at the desired position and get our required inverted number



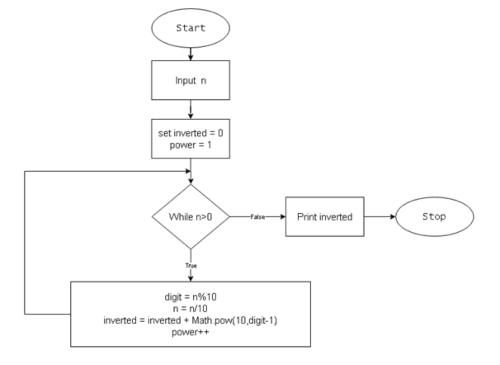


We add all these digits after multiplying them with their respective required positional powers subtracted by 1 to get the inverted number.

The inverted number we get is **23154**.

We create a variable for storing inverted number and initialize it to 0, along with a variable which increments its count after every iteration of the loop and used for calculating the power of 10(for place value) and multiplying the extracted digit with it.

The algorithm for the problem can be illustrated as:



**Programming Implementation (in Java)**

import java.util.\*;

public class Main {

    public static void main(String[] args) {

        Scanner scn = new Scanner(System.in);

        int n = scn.nextInt();

        int inv = 0; //inverted number at start

        int op = 1; // original place

        while (n > 0) {

            int od = n % 10; //last digit  of original number

            int id = op; // id =  inverted digit

            int ip = od; // ip =  inverted place

            //Make change to inv  using ip and id

            inv = inv + (id \* (int) Math.pow(10, ip - 1));

            n = n / 10;

            op++;

        }

        System.out.println(inv);

    }

}

We can reduce our code by not using *id* and *ip* variables in the following manner:

import java.util.\*;

public class Main {

    public static void main(String[] args) {

        Scanner scn = new Scanner(System.in);

        int n = scn.nextInt();

        int inverted = 0;

        int power = 1;

        while (n > 0) {

            int digit = n % 10;

            n = n / 10;

            inverted += power \* Math.pow(10, digit - 1);

            power++;

        }

        System.out.println(inverted);

    }

}

**Space & Time Complexity Analysis**

* What is the **time complexity** of the above solution ?

Since, we are running a loop which reduces the number digit by digit, i.e. extracts one digit in each iteration, time complexity will be O(number of digits).

Now, for a number N, number of digits will be Hence time complexity will turn out to be O(log10 N).

*Note*: We add *power \* Math.pow(10, digit - 1)* to our *inverted* variable at each step of iteration. Calculating 10(digit-1) will take log(digit – 1) time, which is negligible as digits are less than 8, thus log(8-1) can be considered constant.

* What is the **space complexity** of above solution ?

We are not using any data structure, hence it is taking constant space. Thus space complexity will be O(1).

Note: You may think that we have used two integers *inverted* and *power*, which will take 4 or 8 bytes each (depending on the system architecture), hence we should write our space complexity as O(8) or O(16). But we generally ignore such constant space in Big – O complexity analysis.