“First, solve the problem. Then, write the code.”

– John Johnson

Dear readers welcome to the article on how to ***Rotate a Number***. Trust me, it is a nice and easy problem that will help you analyze your thinking skills.

***Problem Statement***

Given two numbers n and k, we have to rotate the number n by k digits. If k is positive, right rotate the number i.e. remove k rightmost digits and make them leftmost. Else for negative values of k, left rotate the number, i.e. remove k digits from left and make them rightmost digits.

Eg) Rotating 562984 by +2 will give 846529, whereas rotating it by -2 will give 652984.

*Note*: k can be larger than the number of digits in number n. In that case, we will ignore rotations of the whole number and left/rotate only the rotations left after reduction.

Eg) If we are asked to rotate 562984 by +9, then we will ignore rotation of +6 (number of digits) and rotate it by +3, i.e. 984562.

Important Links: [Problem Link](https://www.pepcoding.com/resources/online-java-foundation/getting-started/rotate-a-number-official/ojquestion), [Question Video](https://www.youtube.com/watch?v=ysMHtDIlU0c), [Solution Video](https://www.youtube.com/watch?v=lt8oCGqYMGg)

***Deducing Algorithm***

Firstly, we should concentrate on how to solve this if k > 0 (right rotation) only and k is less than the number of digits of n.

Consider an example, if we are asked to rotate n = 25398 by k = +2 digits. As per the problem statement, the required result will be 98253.

We need to find out two entities to solve this problem, what is the ***divisor*** and what is the ***multiplier***.

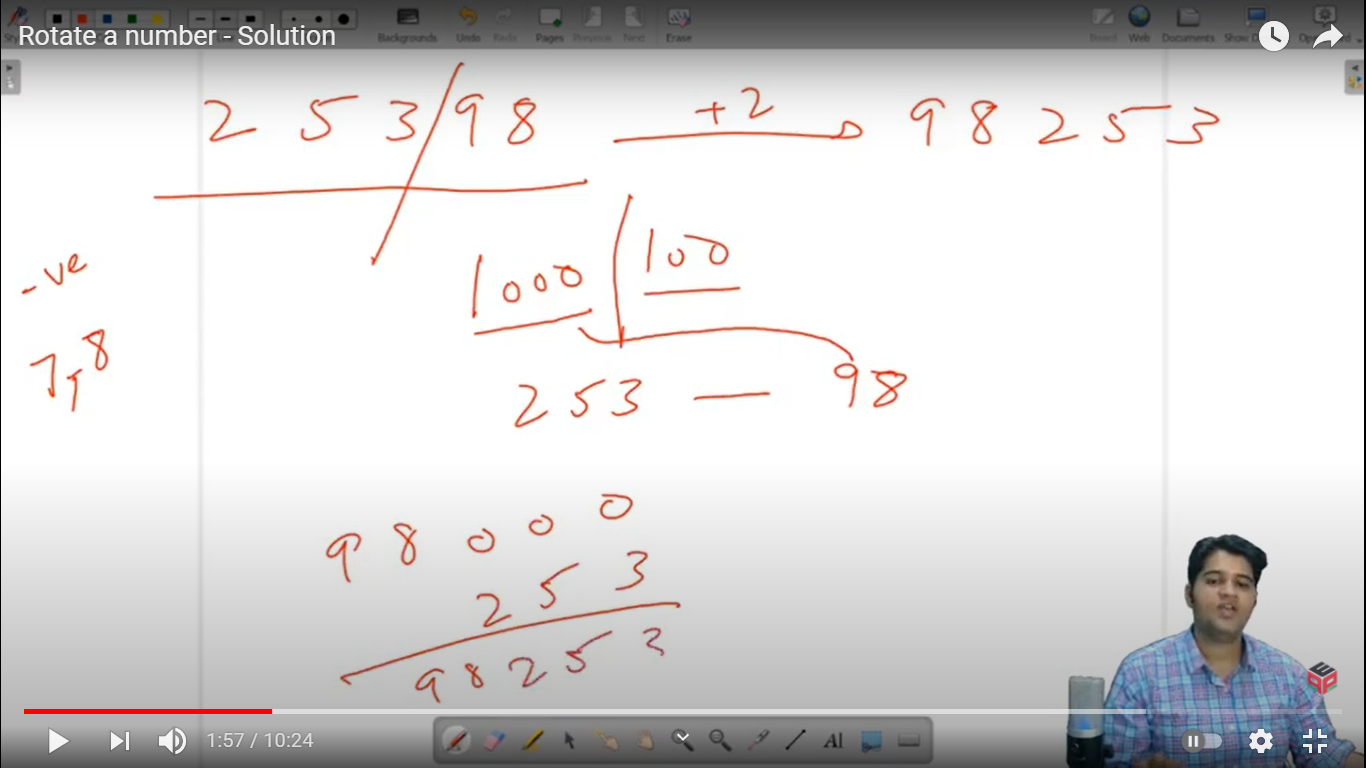
We want to break 25398 as *253* and *98,* and then join 98 and 253 as 98253. Can you guess, what can be the divisor and multiplier to break 98 and then add it to the front?

To get the last k digits of n, we will take divisor as ***10k***, that is divide n by 10k to get 98 in this case.

Now, we will have the last k digits in a variable say *q.* Also*,* store the reduced number in a variable, say *r*, i.e. r = n % divisor. In this case, the value of r will be 253.

So the only thing remaining is to multiply 98 by a multiplier and add the product to r (253), which will bring 98 in front of r = 253 i.e. make n as 98253. Try to think in a similar manner and find what can be the multiplier.

Let the number of digits in number n be *nod*. So, the multiplier will be ***10(nod - k)***. We will multiply 98 by 10(5-2) = 103 = 1000. Hence q will become 98 \* 1000 = 98000. q will now be added to r (253), thus making it 98000 + 253 = 98253.



I hope, you get the logic used. If you are not able to understand it, then I recommend you watch [this](https://www.youtube.com/watch?v=lt8oCGqYMGg) video from [0:20, 2:00].

The first remaining special case is when ***k > number of digits***. As you can see that rotating the number of nod digits by nod places will bring the same number again. Hence, we will try to subtract nod as many times as possible from the value of k.

Can you think to do this using a formula? Try to use modulo operation.

So, we can make k as k % nod before we start with our algorithm. Taking modulo of k by nod will reduce the k to less than nod, neglecting the full cycles of nod.

Now, the last special case remaining is when ***k is negative*** (left rotation).

I want you to derive a relation between left rotation and right rotation, i.e. find an equivalent positive value of k, for each k < 0.

Were you able to think that rotating k digits from first to last is the same as rotating (nod - k) digits from last to first?

Yey! We deduced a relation that whenever we get a negative value of k, we will add the number of digits *nod* to k, to make it a positive equivalent of it.

***Pseudo Code/ Algorithm***

* Get the total number of digits in n (by repeated division of n by 10) in a variable, say, *nod*.

*Note*: Apply division on a temporary variable, otherwise your n will become equal to the number of test cases that are going to be passed, i.e. 0.

* Update k as k = k % nod (to handle the first special case).
* If k is negative, add nod to the value of k (to handle second special case).
* Now calculate multiplier and divisor, which are 10k and 10(nod-k) respectively.
* Get the last k digits of the number n, in a variable q, by using the formula q = n / divisor, where the divisor is 10k.
* Get the first (nod - k) digits in a variable r, by using the formula r = n % divisor.
* Now Multiply q with the multiplier, i.e. q = q \* multiplier where multiplier = 10(nod - k).
* The resultant value will be r + q.

Seems simple, right? Do give it a try before reading the code.

***Implementation (Java)***

**import java.util.\*;**

**import java.lang.\*;**

**import java.io.\*;**

**public class Main{**

**public static void main(String[] args)**

**{**

**Scanner scn = *new* Scanner(System.in);**

**int n = scn.nextInt();**

**int k = scn.nextInt();**

**int nod = 0; *//number of digits***

**int temp = n;**

***//temporarily storing number to perform division***

***while*(temp != 0)**

**{**

**temp /= 10;**

**nod++;**

**}**

***//25398 will give nod = 5***

**int div = 1; *//initialize divisor = 1***

**int mult = 1; *//initialize multiplier = 1***

***for*(int i = 1; i<= nod;i++)**

**{**

***if*(i <= k)**

***//while the iterator is less than k***

**{**

***//we increase the value of divisor***

**div \*= 10;**

**}**

***else***

***//or else we increase the value of multiplier***

**{**

**mult \*= 10;**

**}**

**}**

**int quo = n / div; *// extracting the quotient***

**int rem = n % div; *// extracting the remainder***

**int r = rem \* mult + quo;**

***//forming the rotated number***

**System.out.println(r);**

**}**

**}**

This code is written and explained by Mr. Sumeet Malik in [this video](https://www.youtube.com/watch?v=lt8oCGqYMGg) from *[3:00, 9:55]*. Please refer to it if you are stuck somewhere.

***Time & Space Complexity Analysis***

Give it a try, first find the time complexity of getting the number of digits in n, then finding values of multiplier and divisor, and finally the values of q and r.

***Time Complexity*** -

* Getting the number of digits in n will take O(log10 n) time.
* Finding 10k and 10(nod - k) will take O(k) and O(nod - k) time respectively since k is of the order of the number of digits (after taking mod with nod), its time complexity will also be O(log10 n).
* Finding q and r are constant time-taking operations.

Hence, the overall time complexity will be O(number of digits) = O(log10 n).

***Space Complexity*** - We are just using integer variables nod, multiplier, divisor, r, q, which will take constant {O(1)} space.

I hope, you enjoyed solving the problem with me. Let’s meet now to deep dive into how the number system works from the next article.

*Article Written By: Archit Aggarwal*