Leetcode 1903 - Largest Odd Number in String....

Ex 1 num = "52" "2" is even ⇒ Continue checking
Output = "5" "5" is Odd ⇒ Return 5,

Ex 2 num = "4206" "6" is even
Output = "" Return "";

Ex 3 num = "35427" ⇒ "7" is odd
Output = "35427" Return, 35427

App Will start from the last character to check the digit is Odd or not.

If odd is found, we will return the substring from the beginning up to that index.

If no odd digit is found, will return an empty String.

```
class Solution {
    public String largestOddNumber(String num) {
        int n = num.length();


        for(int i = n -1 ; i>=0 ; i--){ // start from end
            if((num.charAt(i) - '0') % 2 != 0){ // line 1
                return num.substring(0 , i+1); // line 2
            }
        }
        return "";
    }
}
```

line 1 - It checks if the current character in the string is an odd number.

num.charAt(i) gives the character index i, but it's a char, not an integer.

That's why, subtracting '0' converts into an integer.

eg. char digit = '5';

int value = digit - '0'; = '5' - '0' = 5

line 2 - This will returns the largest odd-numbered substring from the given string.

return num.substring(0, i+1);

↑ startIndex

↓ endIndex (include the current odd digit found).

Ex - "42039857462O"

index → 0 1 2 3 4 5 6 7 8 9 10 11

O E E E E

✓ X X X X

, index is 7

But if you will count from 0, then it comes at 8.

That's why we are adding i+1.

∴ return num.substring(0, 7+1);

num.substring(0,8);

↳ "14203985 7"

So, Time Compl^n = O(N)

Space " = O(N) {In Java strings are immutable, meaning any operation like substring creates a new string in memory instead of modifying the original one. That's why extra space is required.

In the worst case, if the entire string is odd, then it creates a copy of the entire string, which takes O(N) space.