**Reverse Integer**

Given a signed 32-bit integer x, return x*with its digits reversed*. If reversing x causes the value to go outside the signed 32-bit integer range [-231, 231 - 1], then return 0.

**Assume the environment does not allow you to store 64-bit integers (signed or unsigned).**

**Example 1:**

**Input:** x = 123

**Output:** 321

**Example 2:**

**Input:** x = -123

**Output:** -321

**Example 3:**

**Input:** x = 120

**Output:** 21

**Constraints:**

* -231 <= x <= 231 - 1

/\*\*

\* @param {number} x

\* @return {number}

\*/

var reverse = function(x) {

};

Solution

Approach 1: Pop and Push Digits & Check before Overflow

**Intuition**

We can build up the reverse integer one digit at a time. While doing so, we can check beforehand whether or not appending another digit would cause overflow.

**Algorithm**

Reversing an integer can be done similarly to reversing a string.

We want to repeatedly "pop" the last digit off of x*x* and "push" it to the back of the \text{rev}rev. In the end, \text{rev}rev will be the reverse of the x*x*.

To "pop" and "push" digits without the help of some auxiliary stack/array, we can use math.

//pop operation:

pop = x % 10;

x /= 10;

//push operation:

temp = rev \* 10 + pop;

rev = temp;

However, this approach is dangerous, because the statement \text{temp} = \text{rev} \cdot 10 + \text{pop}temp=rev⋅10+pop can cause overflow.

Luckily, it is easy to check beforehand whether or this statement would cause an overflow.

To explain, lets assume that \text{rev}rev is positive.

1. If temp = \text{rev} \cdot 10 + \text{pop}*temp*=rev⋅10+pop causes overflow, then it must be that \text{rev} \geq \frac{INTMAX}{10}rev≥10*INTMAX*​
2. If \text{rev} > \frac{INTMAX}{10}rev>10*INTMAX*​, then temp = \text{rev} \cdot 10 + \text{pop}*temp*=rev⋅10+pop is guaranteed to overflow.
3. If \text{rev} == \frac{INTMAX}{10}rev==10*INTMAX*​, then temp = \text{rev} \cdot 10 + \text{pop}*temp*=rev⋅10+pop will overflow if and only if \text{pop} > 7pop>7

Similar logic can be applied when \text{rev}rev is negative.

class Solution {

public int reverse(int x) {

int rev = 0;

while (x != 0) {

int pop = x % 10;

x /= 10;

if (rev > Integer.MAX\_VALUE/10 || (rev == Integer.MAX\_VALUE / 10 && pop > 7)) return 0;

if (rev < Integer.MIN\_VALUE/10 || (rev == Integer.MIN\_VALUE / 10 && pop < -8)) return 0;

rev = rev \* 10 + pop;

}

return rev;

}

}

**Complexity Analysis**

* Time Complexity: O(\log(x))*O*(log(*x*)). There are roughly \log\_{10}(x)log10​(*x*) digits in x*x*.
* Space Complexity: O(1)*O*(1).