Palindrome Number

Problem definition

Given an integer x, return true if x is a palindrome, and false otherwise.

Example cases

Example 1:

Input: x = 121
Output: true
Explanation: 121 reads as 121 from left to right and from right to left.

Example 2:

Input: x = -121
Output: false
Explanation: From left to right, it reads -121. From right to left, it becomes 121-. Therefore it is not a palindrome.

Example 3:

Input: x = 10
Output: false
Explanation: Reads 01 from right to left. Therefore it is not a palindrome.

Constraints:

-231 <= x <= 231 - 1

Follow up: Could you solve it without converting the integer to a string?

Example test cases

In [1]:
```
def test_cases():
    assert isPalindrome(121) is True
    assert isPalindrome(-121) is False
    assert isPalindrome(10) is False
    print("All test cases passed!")
```

Solutions

$O(n)$

Solving by converting the integer to string.

In [2]:
```
def isPalindrome(x):
    x = str(x)
    pointer1 = 0
    pointer2 = len(x) - 1

    while not (pointer1 > pointer2):
        if x[pointer1] != x[pointer2]:
            return False
        pointer1 += 1
        pointer2 -= 1

    return True
```

In [3]:
```
isPalindrome(10)
```

Out[3]:
```
False
```

In [4]:
```
test_cases()
```

```
All test cases passed!
```

The problem is easy to solve when using two pointers method for strings. But since integers are not sequence objects, indexing doesn't work.

$O(n)$

Solving without converting the integer to string.

In [5]:
```
def isPalindrome(x):
    inverse_num = 0
    if (x < 0) or (x % 10 == 0 and x != 0):
        return False
    x_ = x
    while x_ != 0:
        inverse_num *= 10
        remainder = x_ % 10
        inverse_num += remainder
        x_ = x_ // 10
    return x == inverse_num
```

In [6]:
```
test_cases()
```

```
All test cases passed!
```

For integers, the task is a bit complicated. The most straightforward way was to convert the integer to its inverse and see if the numbers are equal. Additionally, negative numbers and numbers ending with 0 are always False.

`Modulus operator (%)` in python can be used for getting the remainder when divided by a certain number. To get the last digit of the number, we can do the following:

In [7]:
```
x = 13545
x % 10
```

Out[7]:
```
5
```

Next digit, can be extracted by removing the last digit and again applying the same concept:

In [8]:
```
x // 10
```

Out[8]:
```
1354
```

Floor division can be used to remove the last digit from the number. Then, reassign the number and do the same operation. The inverse number can be constructed during that process, each time summing up the last extracted digit and multiplying the total number by 10.

Even a better solution!

$O(n)$

In [9]:
```
def isPalindrome(x):
    if (x < 0) or (x % 10 == 0 and x != 0):
        return False

    inverse_num = 0
    while x > inverse_num:
        inverse_num = 10 * inverse_num + x % 10
        x //= 10

    return x == inverse_num or x == inverse_num // 10
```

In [10]:
```
test_cases()
```

```
All test cases passed!
```

The solution is still O(n), but in this case it just checks the half of the integer instead of traversing through the whole integer. Additionally, there is 1 less variable used.