<https://leetcode.com/problems/palindrome-number/description/>

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

**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:**

-2^31 <= x <= 2^31 - 1

**Follow up:**

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

**Attempt 1: 2024-06-29**

**Solution 1: Reversing the Entire Number (10 min)**

class Solution {

    public boolean isPalindrome(int x) {

        int result = 0;

        int y = x;

        while(y > 0) {

            result = result \* 10 + y % 10;

            y = y / 10;

        }

        return result == x;

    }

}

**Solution 2: Reversing the Half of the Number (10 min)**

class Solution {

    public boolean isPalindrome(int x) {

        // Test out by x = -121 or x = 10

        if(x < 0 || (x != 0 && x % 10 == 0)) {

            return false;

        }

        int y = 0;

        while(y < x) {

            y = y \* 10 + x % 10;

            x /= 10;

        }

        return y == x || x == y / 10;

    }

}

**Refer to**

<https://leetcode.com/problems/palindrome-number/solutions/3651712/2-method-s-c-java-python-beginner-friendly/>

**Approach 1: Reversing the Entire Number**

**Explanation:**

We begin by performing an initial check. If the input number x is negative, it cannot be a palindrome since palindromes are typically defined for positive numbers. In such cases, we immediately return false.

We initialize two variables:

reversed: This variable will store the reversed value of the number x.

temp: This variable is a temporary placeholder to manipulate the input number without modifying the original value.

We enter a loop that continues until temp becomes zero:

Inside the loop, we extract the last digit of temp using the modulo operator % and store it in the digit variable.

To reverse the number, we multiply the current value of reversed by 10 and add the extracted digit.

We then divide temp by 10 to remove the last digit and move on to the next iteration.

Once the loop is completed, we have reversed the entire number. Now, we compare the reversed value reversed with the original input value x.

If they are equal, it means the number is a palindrome, so we return true.

If they are not equal, it means the number is not a palindrome, so we return false.

The code uses a long long data type for the reversed variable to handle potential overflow in case of large input numbers.

class Solution {

public boolean isPalindrome(int x) {

if (x < 0) {

return false;

}

long reversed = 0;

long temp = x;

while (temp != 0) {

int digit = (int) (temp % 10);

reversed = reversed \* 10 + digit;

temp /= 10;

}

return (reversed == x);

}

}

**Approach 2: Reversing Half of the Number**

Instead of reversing the entire number, we can reverse only the last half of the number. This approach is tricky because when we reverse the last half of the number, we don't want the middle digit to be reversed back to its original value. This can happen if the number has an odd number of digits. To resolve this, we can compare the first half of the number with the reversed second half of the number.

**Explanation:**

We begin with an initial check to handle special cases:

If the input number x is negative, it cannot be a palindrome since palindromes are typically defined for positive numbers. In such cases, we immediately return false.

If x is non-zero and ends with a zero, it cannot be a palindrome because leading zeros are not allowed in palindromes. We return false for such cases.

We initialize two variables:

reversed: This variable will store the reversed second half of the digits of the number.

temp: This variable is a temporary placeholder to manipulate the input number without modifying the original value.

We enter a loop that continues until the first half of the digits (x) becomes less than or equal to the reversed second half (reversed):

Inside the loop, we extract the last digit of x using the modulo operator % and add it to the reversed variable after multiplying it by 10 (shifting the existing digits to the left).

We then divide x by 10 to remove the last digit and move towards the center of the number.

Once the loop is completed, we have reversed the second half of the digits. Now, we compare the first half of the digits (x) with the reversed second half (reversed) to determine if the number is a palindrome:

For an even number of digits, if x is equal to reversed, then the number is a palindrome. We return true.

For an odd number of digits, if x is equal to reversed / 10 (ignoring the middle digit), then the number is a palindrome. We return true.

If none of the above conditions are met, it means the number is not a palindrome, so we return false.

The code avoids the need for reversing the entire number by comparing only the necessary parts. This approach reduces both time complexity and memory usage, resulting in a more efficient solution.

class Solution {

public boolean isPalindrome(int x) {

if (x < 0 || (x != 0 && x % 10 == 0)) {

return false;

}

int reversed = 0;

while (x > reversed) {

reversed = reversed \* 10 + x % 10;

x /= 10;

}

return (x == reversed) || (x == reversed / 10);

}

}