Palindrome Number

Easy

Given an integer `x`, return `true` if `x` is palindrome integer.
An integer is a **palindrome** when it reads the same backward as forward.

- For example, `121` is a palindrome while `123` is not.

Example 1:

```
Input: x = 121

Output: true

Explanation: 121 reads as 121 from left to right and from right to left.
```

```
class Solution {
    public boolean isPalindrome(int x) {
        int number=x;
        int sum=0,n;
        while(x>0){
            n=x%10;
            sum=sum*10+n;
            x=x/10;
        }
        if(sum==number){
                return true;
            }else
                return false;

    }
}
```

Submission Detail

11510 / 11510 test cases passed.

Runtime: **11 ms**

Memory Usage: **44.8 MB**

Roman to Integer

Easy

Roman numerals are represented by seven different symbols: `I`, `V`, `X`, `L`, `C`, `D` and `M`.

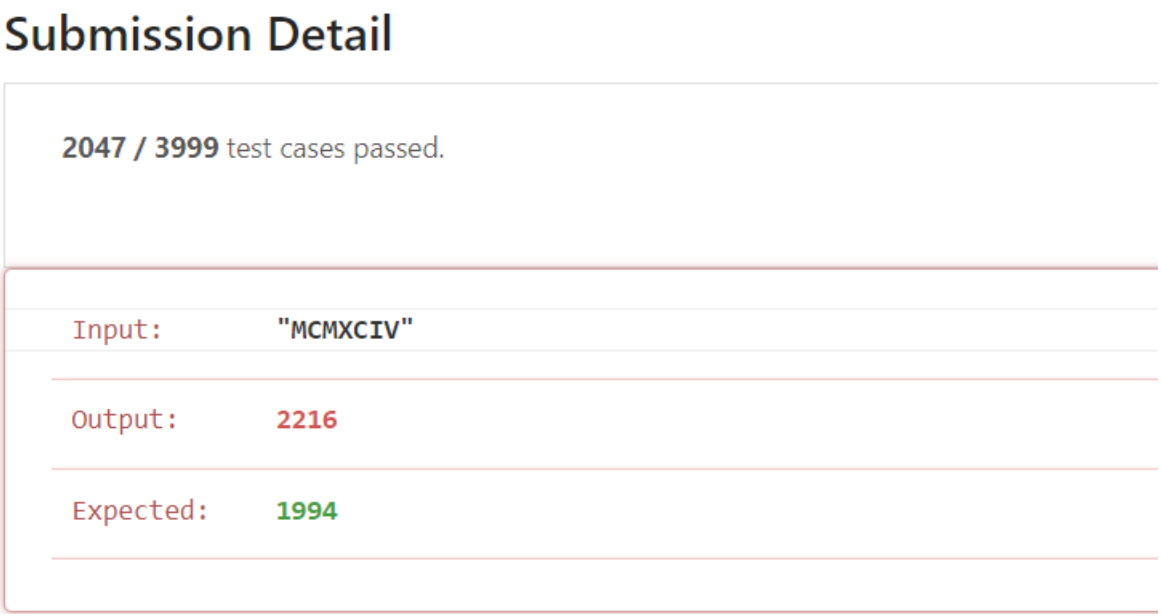
Symbol	Value
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

For example, `2` is written as `II` in Roman numeral, just two one's added together. `12` is written as `XII`, which is simply `X + II`. The number `27` is written as `XXVII`, which is `XX + V + II`.

```
class Solution {
    public int romanToInt(String s) {
        int sum=0;
        char c;
        for (int i = 0; i < s.length(); i++){
            c=s.charAt(i);
```

```
    if(c=='I'){
        sum=sum+1;
    }else{
        if(c=='V'){
            sum=sum+5;
        }else{
            if(c=='X'){
                sum=sum+10;
            }else{
                if(c=='L'){
                    sum=sum+50;
                }else{
                    if(c=='C'){
                        sum=sum+100;
                    }else{
                        if(c=='D'){
                            sum=sum+500;
                        }else{
                            if(c=='M'){
                                sum=sum+1000;
                            }
                        }
                    }
                }
            }
        }
    }
}
```

```
        return sum;
    }
}
```



Add Two Numbers

Medium

You are given two **non-empty** linked lists representing two non-negative integers. The digits are stored in **reverse order**, and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list.

You may assume the two numbers do not contain any leading zero, except the number 0 itself.

```
Input: l1 = [2,4,3], l2 = [5,6,4]

Output: [7,0,8]

Explanation: 342 + 465 = 807.
```

```
class Solution {
    public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
        ListNode newList = new ListNode();
    ListNode curr = newList;
    int sum=0,carry = 0,x,y;
```

```
    while (l1 != null || l2 != null) {
        if(l1 != null) {
            x= l1.val;
        } else {
            x=0;
        }
        if(l2 != null) {
            y=l2.val;
        } else {
            y=0;
        }
        sum = carry + x + y;
        carry = sum / 10;
        curr.next = new ListNode(sum % 10);
        curr = curr.next;
        if (l1 != null) l1 = l1.next;
        if (l2 != null) l2 = l2.next;
    }
    if (carry > 0) {
        curr.next = new ListNode(carry);
    }
     return newList.next;
     }
}
```

Submission Detail

1568 / 1568 test cases passed.

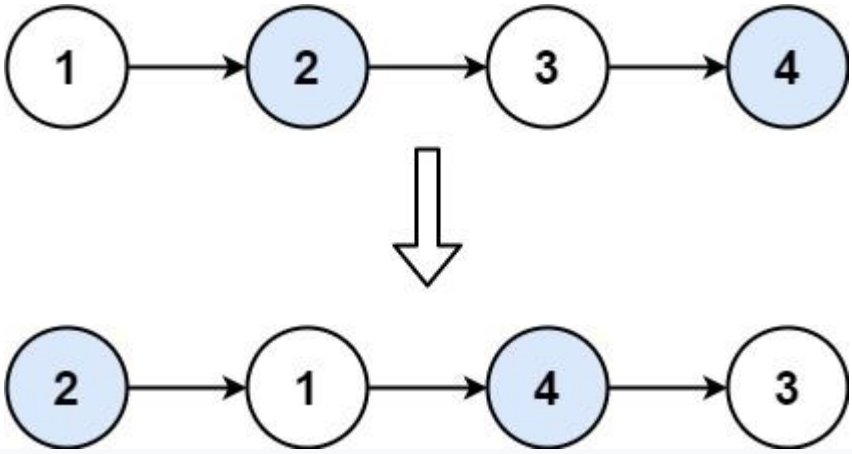
Runtime: **4 ms**
Memory Usage: **47.4 MB**

Swap Nodes in Pairs

Medium

Given a linked list, swap every two adjacent nodes and return its head. You must solve the problem without modifying the values in the list's nodes (i.e., only nodes themselves may be changed.)

Example 1:

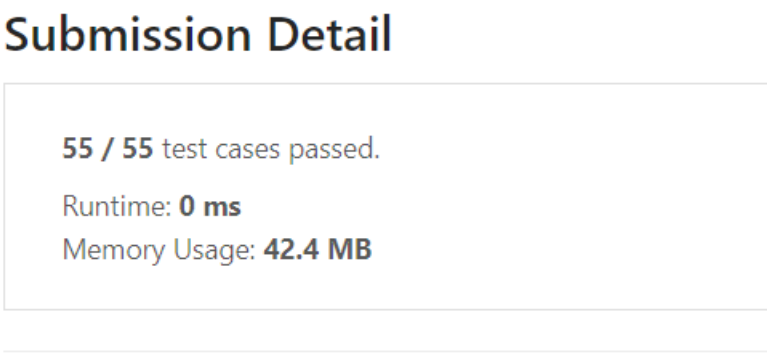



```
Input: head = [1,2,3,4]

Output: [2,1,4,3]
```

```
class Solution {
    public ListNode swapPairs(ListNode head) {
      if(head == null || head.next == null) {
          return head;
       }
       ListNode fNode = head;
       ListNode sNode = head.next;
```

```
        ListNode newList = swapPairs(sNode.next);
        sNode.next = fNode;
        fNode.next = newList;
        return sNode;
    }
}
```



Reverse Integer

Medium

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 `[-2^31, 2^31 - 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
```

```
class Solution {
    public int reverse(int x) {
        int rev=0,rem;
        if(x<0) {
                                while(x<0)
```

```
                {
                        rem=x%10;
                        rev=(rev*10)+rem;
                        x=x/10;
                }
            }else{
        while (x > 0) {
                        rem = x % 10;
                        rev = (rev * 10) + rem;
                        x = x / 10;
                }
    }
    return rev;
  }
}
```

Submission Detail

1027 / 1032 test cases passed.

Input:	**1534236469**
Output:	**1056389759**
Expected:	**0**