# Palindrome Number

Easy

Given an integer , return if is palindrome integer. x true x

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){

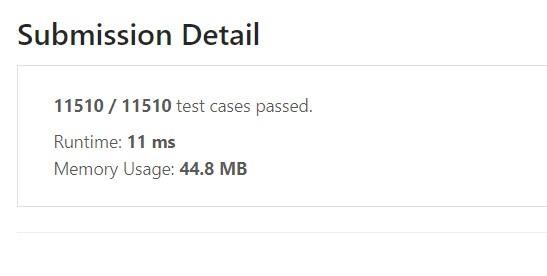
return true;

}else

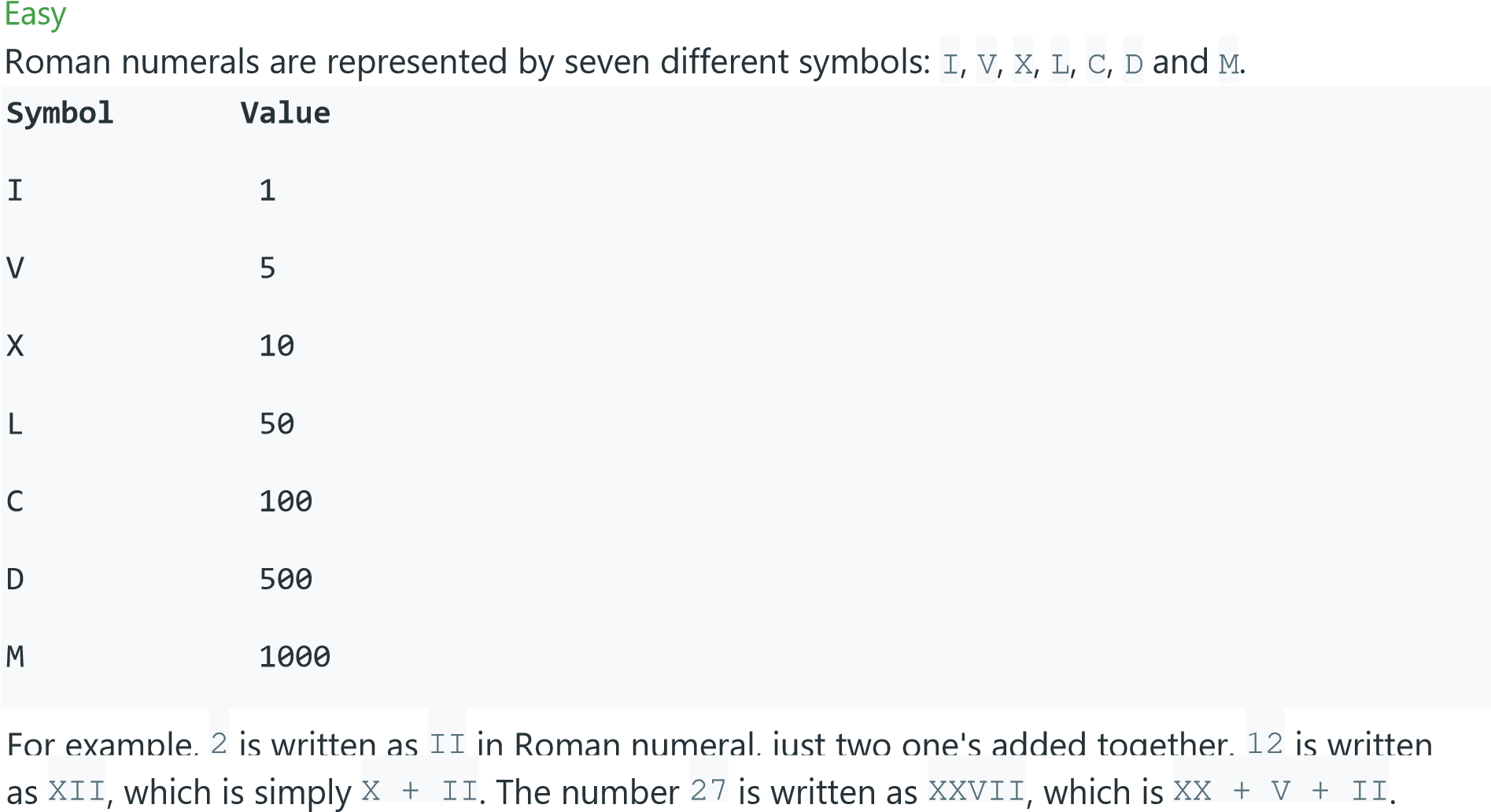
return false;

}

}



# Roman to Integer



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;

}

}

}

}

}

}

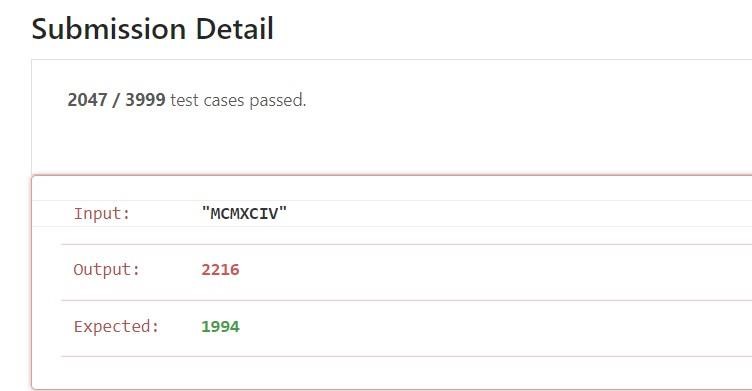
}

}

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) {

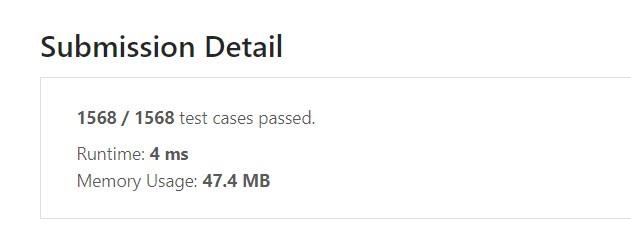
curr.next = new ListNode(carry);

}

return newList.next;

}

}



# Reverse Integer

Medium

|  |  |  |  |
| --- | --- | --- | --- |
| Given a signed 32-bit integer x | , return x | *with its digits reversed*. If reversing x | causes the value to |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | [-2 | 31 | , 2 | 31 | - 1] |   go outside the signed 32-bit integer range , 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;

}

}



# Median of Two Sorted Arrays

Hard

Given two sorted arrays nums1 and nums2 of size m and n respectively, return **the median** of the two sorted arrays.

The overall run time complexity should be O(log (m+n)).

**Example 1:**

**Input:** nums1 = [1,3], nums2 = [2]

**Output:** 2.00000

**Explanation:** merged array = [1,2,3] and median is 2.

class Solution {

public double findMedianSortedArrays(int[] nums1, int[] nums2) { int l=nums1.length+nums2.length;

int newArr[]=new int[l]; int a1=nums1.length; int b1=nums2.length;

System.arraycopy(nums1, 0, newArr, 0, a1);

System.arraycopy(nums2, 0, newArr, a1, b1); for (int i = 0; i < newArr.length; i++) { System.out.println(newArr[i]);

}

int temp=0;

for (int i = 0; i < newArr.length; i++) { for(int j=i+1;j<newArr.length;j++) { if(newArr[i]>newArr[j]) { temp=newArr[i]; newArr[i]=newArr[j];

newArr[j]=temp;

}

}

}

System.out.println(Arrays.toString(newArr));

int m;

double median; if(l%2==0) {

m=(0+(l-1))/2;

median=(double)(newArr[m]+newArr[m+1])/2;

return median; }else { m=(0+(l-1))/2; median=newArr[m];

return median;

}

}

}

