

1295. Find Numbers with Even Number of Digits

Easy  1270  99  Add to List  Share

Given an array `nums` of integers, return how many of them contain an **even number** of digits.

Example 1:

```
Input: nums = [12,345,2,6,7896]
Output: 2
Explanation:
12 contains 2 digits (even number of digits).
345 contains 3 digits (odd number of digits).
2 contains 1 digit (odd number of digits).
6 contains 1 digit (odd number of digits).
7896 contains 4 digits (even number of digits).
Therefore only 12 and 7896 contain an even number of digits.
```

Ans:

```
class Solution {
public int findNumbers(int[] nums) {
    int c=0;
    for(int num : nums)
    {
        if(countDigit(num)%2==0)
        {
            c++;
        }
    }
    return c;
}
public int countDigit(int num)
{
    int c=0;
    while(num>0)
    {
        c++;
        num /= 10;
    }
    return c;
}
}
```

1672. Richest Customer Wealth

Easy  1846  261  Add to List  Share

You are given an $m \times n$ integer grid `accounts` where `accounts[i][j]` is the amount of money the i^{th} customer has in the j^{th} bank. Return the **wealth** that the richest customer has.

A customer's **wealth** is the amount of money they have in all their bank accounts. The richest customer is the customer that has the maximum **wealth**.

Example 1:

```
Input: accounts = [[1,2,3],[3,2,1]]
Output: 6
Explanation:
1st customer has wealth = 1 + 2 + 3 = 6
2nd customer has wealth = 3 + 2 + 1 = 6
Both customers are considered the richest with a wealth of 6 each, so return 6.
```

class Solution

```
{
    public int maximumWealth(int[][] accounts)
    {

        int max = 0;
        int sum = 0;

        for(int r=0; r < accounts.length; r++)
        {

            for(int c=0; c < accounts[r].length; c++)
            {
                sum = sum + accounts[r][c];
            }

            if(sum > max)
            {
                max = sum;
            }

            sum=0;

        } // End outer loop

        return max;
    }

} // class end
```

Inverse Of A Number

Easy



1. You are given a number following certain constraints.
2. The key constraint is if the number is 5 digits long, it'll contain all the digits from 1 to 5 without missing any and without repeating any. e.g. 23415 is a 5 digit long number containing all digits from 1 to 5 without missing and repeating any digit from 1 to 5. Take a look at few other valid numbers - 624135, 81456273 etc. Here are a few invalid numbers - 139, 7421357 etc.
3. The inverse of a number is defined as the number created by interchanging the face value and index of digits of number. e.g. for 426135 (reading from right to left, 5 is in place 1, 3 is in place 2, 1 is in place 3, 6 is in place 4, 2 is in place 5 and 4 is in place 6), the inverse will be 416253 (reading from right to left, 3 is in place 1, 5 is in place 2, 2 is in place 3, 6 is in place 4, 1 is in place 5 and 4 is in place 6). More examples - inverse of 2134 is 1243 and inverse of 24153 is 24153.
4. Take as input number "n", assume that the number will follow constraints.
5. Print it's inverse.

Constraints

$1 \leq n < 10^8$, and following other constraints defined above.

Handwritten notes showing the process of finding the inverse of a number. The number 21453 is shown with its digits and positions (1 to 5 from right to left). The inverse is calculated by multiplying each digit by its position and summing the results.

Digit	Position	Product
2	5	10
1	4	4
4	3	12
5	2	10
3	1	3

Sum = 10 + 4 + 12 + 10 + 3 = 39

```
import java.util.*;
public class Main
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        int number = in.nextInt();

        int invertNumb = 0;
        int digitExtracted = 0;

        //Position of digit ,as we divide the number ,will increase till n =0
        int postnDigit = 1;
```

```

//extract each digit ,till last
while(number>0)
{
    // get each digit from back
    digitExtracted = number%10;

    //Reduce one for position in power
    digitExtracted--;

    invertNumb = invertNumb + (int)
(postnDigit * Math.pow(10 , digitExtracted));


    // increase the position of Digit
    postnDigit++;

    //Reduce The number
    number = number/10;
}

System.out.print(invertNumb);
}
}

```

The Curious Case Of Benjamin Bulbs

Easy 



1. You are given n number of bulbs. They are all switched off. A weird fluctuation in voltage hits the circuit n times. In the 1st fluctuation all bulbs are toggled, in the 2nd fluctuation every 2nd bulb is toggled, in the 3rd fluctuation every 3rd bulb is toggled and so on. You've to find which bulbs will be switched on after n fluctuations.
2. Take as input a number n, representing the number of bulbs.
3. Print all the bulbs that will be on after the nth fluctuation in voltage.

Constraints

$2 \leq n < 10^9$

Format

Input

n, an integer

Explanation :

Suppose n= 20, bulbs were initially off.
 toggling is also happening n times i.e. 20 times bulbs will toggle.
 First time : all 20 bulbs will be toggled
 Second time : every second bulb will be toggled.
 and so on,
 we have to find after nth toggle ,which bulbs be ON.

1 ✓ ★	6 ✓✓✓✓	11 ✓✓	16 ✓✓✓✓✓ ★
2 ✓✓	7 ✓✓	12 ✓✓✓✓✓	17 ✓✓
3 ✓✓	8 ✓✓✓✓	13 ✓✓	18 ✓✓✓✓✓✓
4 ✓✓✓ ★	9 ✓✓✓ ★	14 ✓✓✓✓	19 ✓✓
5 ✓✓	10 ✓✓✓✓	15 ✓✓✓✓	20 ✓✓✓✓✓✓
1	4	9	16
1 ²	2 ²	3 ²	4 ²

Every Perfect Square, have odd number, of Factors.

Solution: Print perfect square of n , since only Perfect square number will be on after nth fluctuations.

```
import java.util.Scanner;
public class Main{
    public static void main(String[] args)
    {
        // Write your code here
        Scanner sc = new Scanner(System.in);
        int number = sc.nextInt();
        for(int i = 1 ; i*i <= number; i++)
        {
            // print perfect square numbers
            System.out.println(i*i);
        }
    }
}
```

Is A Number Prime

Easy 🔒



1. You've to check whether a given number is prime or not.
2. Take a number "t" as input representing count of input numbers to be tested.
3. Take a number "n" as input "t" number of times.
4. For each input value of n, print "prime" if the number is prime and "not prime" otherwise.

Constraints

1 $\leq t \leq 10000$
2 $\leq n < 10^9$

```
import java.util.*;

public class Prime
{

    public static void main(String[] args)
    {
        Scanner scn = new Scanner(System.in);

        int t = scn.nextInt();
        int i;
        int c = 0;

        while(t > 0)
        {
            int n = scn.nextInt();

            for( i = 2; i*i <= n; i++)
            {
                if(n % i == 0)
                {
                    c = 1;
                    break;
                }
            }
        }
    }
}
```

```


        if(c !=0)
        {
            System.out.println("not prime");
        }
        else
        {
            System.out.println("prime");
        }
        c=0;
        t--;
    }

}

}

```

Print All Primes Till N

Easy 

<
>

1. You've to print all prime numbers between a range.
2. Take as input "low", the lower limit of range.
3. Take as input "high", the higher limit of range.
4. For the range print all the primes numbers between low and high (both included).

Constraints

$2 \leq \text{low} < \text{high} < 10^6$

Format

Input

low
high

Output

n1
n2
.. all primes between low and high (both included)

Example

Sample Input

```
6
24
```

Sample Output

```
7
11
13
17
19
23
```

```
public class PrimeTillN {
```

```

public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int low = sc.nextInt();
    int high = sc.nextInt();
    int count = 0;
    // 2 <= low < high < 10 ^ 6
    while(low <= high)
    {
        // run loop from 2 to till square root of every number coming in low
        for(int i=2; i*i <= low; i++)
        {
            //check for divisibility
            if(low % i == 0)
            {
                //increment a count variable and break
                count ++;
                break;
            }
        }
        if(count == 0)
        {
            System.out.println(low);
        }
        // increase low
        low++;
        //reset the count
        count = 0;
    }
}
}

```


Print Fibonacci Numbers Till N

Easy 🔒

1. You've to print first n fibonacci numbers.
2. Take as input "n", the count of fibonacci numbers to print.
3. Print first n fibonacci numbers.

Constraints

$1 \leq n < 40$

Format

Input

n

Output

0

1

1

2

3

5

8

.. first n fibonaccis

Example

Sample Input

10

```
import java.util.Scanner;

public class Fibo {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int first = 0;
        int last = 1;

        //all three variables act as pointer , move one step ahead in each step
        for(int i=0; i < n ; i++)
        {
            //print Fib no
            System.out.println(first);

            // third pointer is found by adding prev two term
            int current = first + last;

            //assign last ,to first i.e. move 1 step ahead
            first = last;

            // assign current to last i.e. move 1 step ahead
```

```

        last = current;
    }

}
}

```

Time Complexity : $O(n)$, since loop will run n times

Gcd And Lcm

Easy 🔒



1. You are required to print the Greatest Common Divisor (GCD) of two numbers.
2. You are also required to print the Lowest Common Multiple (LCM) of the same numbers.
3. Take input "num1" and "num2" as the two numbers.
4. Print their GCD and LCM.

Constraints

$2 \leq n \leq 10^9$

Format

Input

num1
num2
.. the numbers whose GCD and LCM we have to find.

```

import java.util.Scanner;

public class LCMGCD {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n1 = sc.nextInt();
        int n2 = sc.nextInt();
        int original_n1 = n1;
        int original_n2 = n2;
        int temp;
        while(n2 % n1 != 0)
        {

            // get the reminder, make it new divisor
            temp = n2 % n1;

            //assign new dividend
            n2 = n1;

```

```

        //assign remainder for divisor
        n1 =temp;

    }
    System.out.println(n1);
    // n1 * n2 = lcm * gcd
    System.out.println( original_n1*original_n2/n1);
}
}

```

Time Complexity:

$O(\min(a, b))$ Since we have just one loop from $\min(a, b)$ to 1.

Prime Factorisation Of A Number

Easy 🔒

<

>

1. You are required to display the prime factorisation of a number.
2. Take as input a number n.
3. Print all its prime factors from smallest to largest.

For example:
for n = 1440, the output should be: 2 2 2 2 3 3 5

Constraints

$2 \leq n < 10^9$

Format

Input

n, an integer

```

import java.util.Scanner;

public class PrimeFactor {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n1 = sc.nextInt(); // .i.e 10
        //run loop till square root of number
        for(int i=2; i*i <= n1; i++)
        {
            // keep dividing num, till further it's not divisible
            while(n1 % i == 0) // 10/1 ..2..
            {
                n1 = n1 / i;
                System.out.print(i+" ");
            }
        }
    }
}

```

```

    }
    // Factor may lie beyond square root of n ,e.g 26
    if(n1 !=1)
    {
        System.out.println(n1);
    }
}
}

```

Time Complexity : $O(\sqrt{n})$;

Digit Frequency

Easy

1. Each digit (0 to 9) denotes the student of the Optica Student community.
2. You are given a number n where ith digit denotes that ith task that is assigned to the corresponding digit student.
2. You are given a digit d denotes a student.
3. You are required to calculate the frequency of digit d in number n or how many tasks are assigned to student d.

Constraints
 $0 \leq n \leq 10^9$ $0 \leq d \leq 9$

Format
Input
A number n A digit d
Output
A number representing frequency of digit d in number n

Example
Sample Input

```
994543234
4
```

Sample Output

```
3
```

```

import java.util.*;
public class Main {
    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        int d = scn.nextInt();
        int f = getDigitFrequency(n, d);
        System.out.println(f);
    }
    public static int getDigitFrequency(int n, int d)
    {

        int counter = 0;
        int rem =0;

```

```

while(n>0)
{
    // extract the digit
    rem = n%10;
    //compare with target digit, if found increment the counter
    if(rem==d)
    {
        counter++;
    }
    //reduce the digit
    n =n/10;
}
return counter;
}
}

```

Decimal To Any Base

Easy 

1. You are given a decimal number n.
2. You are given a base b.
3. You are required to convert the number n into its corresponding value in base b.

Constraints

$0 \leq d \leq 512$
 $2 \leq b \leq 10$

Format

Input

A number n
 A base b

Output

A number representing corresponding value of n in number system of base b

Example

Sample Input

57
 2

Sample Output

111001

Required Number: $(1172)_8$

Destination Base	8	634
8	79	②
8	9	①
8	1	①
8	0	①

$2 \times 10^0 = 2$
 $7 \times 10^1 = 70$
 $1 \times 10^2 = 100$
 $1 \times 10^3 = 1000$

$2 + 70 + 100 + 1000$
 $= 1172$

$$\begin{aligned}
 (634)_{10} &= 6 \times 10^2 + 3 \times 10^1 + 4 \times 10^0 \\
 &= 600 + 30 + 4 \\
 &= \underline{634} \\
 (1172)_8 &= 1 \times 8^3 + 1 \times 8^2 + 7 \times 8^1 + 2 \times 8^0 \\
 &= 512 + 64 + 56 + 2 \\
 &= \underline{634}
 \end{aligned}$$

```

import java.util.*;

public class Main
{
    public static void main(String[] args)
    {
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();// number
        int b = scn.nextInt();// base

        int dn = getValueInBase(n, b);

        System.out.println(dn);
    }

    public static int getValueInBase(int n, int b)
    {
        int co =0;
        int c =1;
        int rem=0;

        while(n>0)
        {
            //get the digit one by one
            rem = n % b;

            // place the digit to correct place
            co = rem * c + co;

            //reduce the digit
            n = n / b;

            //increase the c
            c = c * 10;
        }
        return co;
    }
}

```

TIME COMPLEXITY :

We are extracting digits of number n and performing some minute calculations, which will take $O(\log_{10} n)$ time as there can be maximum $\text{floor}(\log_{10} n)$ digits in a number n .

Any Base To Decimal

Easy

1. You are given a number n .
2. You are given a base b . n is a number on base b .
3. You are required to convert the number n into its corresponding value in decimal number system.

Constraints

$0 \leq n \leq 1000000000$
 $2 \leq b \leq 10$

Format

Input

A number n
A base b

Output

A decimal number representing corresponding value of n in base b .

Example

Sample Input

```
111001
2
```

Sample Output

```
57
```

Soln:

We'll be converting 1172 (base 8) into a decimal number.

Step 1 : EXTRACT THE LAST DIGIT

Divide the number by 10(destination base) and calculate all the remainders.

10	1172
10	117 2
10	11 7
10	1 1
10	0 1

Step 2 : MULTIPLY DIGIT WITH POWER OF SOURCE BASE

Now, we have to multiply these remainders with respective powers of the source base(8) and add them to our answer.

10	1172
10	117 2
10	11 7
10	1 1
	0 1

$2 \times 8^0 = 2$
 $7 \times 8^1 = 56$
 $1 \times 8^2 = 64$
 $1 \times 8^3 = 512$

$2 + 56 + 64 + 512 = \underline{634}_{10}$

```
import java.util.*;

public class Main
{
    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        int b = scn.nextInt();
        int d = getValueIndecimal(n, b);
        System.out.println(d);
    }

    public static int getValueIndecimal(int n, int b)
    {
        int rem =0;
        int decimalValue =0;
        int powerIncrementor =1;
        while(n>0)
        {
            // extract the digit one by one
            rem = n%10;
            // form the decimal value
            decimalValue = rem*powerIncrementor + decimalValue;
```



```

        // increment the power
        powerIncrementor = powerIncrementor*b;
        //reduce the digit
        n = n/10;
    }
    return decimalValue;
}
}

```

Print all Divisors of a given Number

👍 114

Problem Statement: Given an integer N, return all divisors of N.

A divisor of an integer N is a positive integer that divides N without leaving a remainder. In other words, if N is divisible by another integer without any remainder, then that integer is considered a divisor of N.

Examples

Example 1:

Input: N = 36

Output: [1, 2, 3, 4, 6, 9, 12, 18, 36]

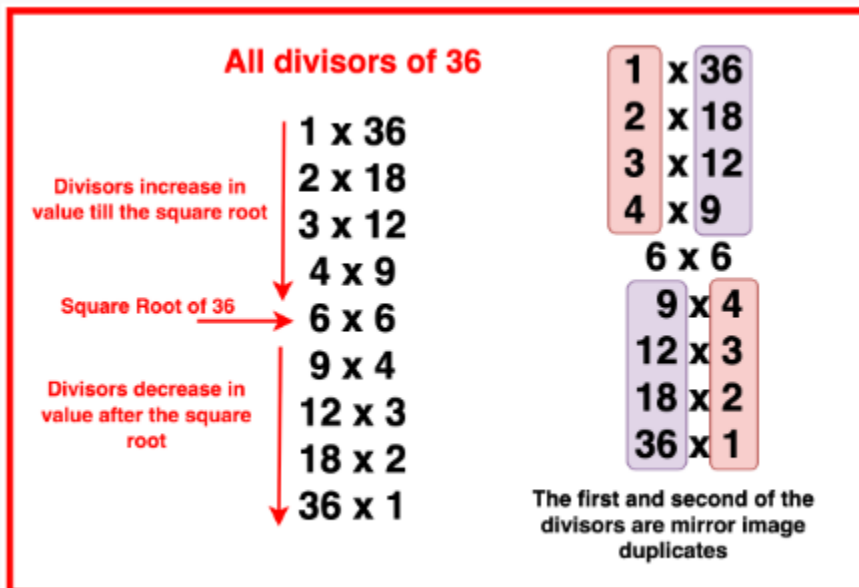
Explanation: The divisors of 36 are 1, 2, 3, 4, 6, 9, 12, 18, 36.

Example 2:

Input: N = 12

Output: [1, 2, 3, 4, 6, 12]

Explanation: The divisors of 12 are 1, 2, 3, 4, 6, 12.



```
package com.test;
```

```

import java.util.ArrayList;

class Main {
    public static ArrayList<Integer> findDivisors(int n) {
        // Initialize an empty
        // ArrayList to store the divisors
        ArrayList<Integer> divisors = new ArrayList<>();

        // Iterate up to the square
        // root of n to find divisors
        // Calculate the square root of n
        int sqrtN = (int) Math.sqrt(n);

        // Loop from 1 to the
        // square root of n
        for (int i = 1; i <= sqrtN; ++i) {
            // Check if i divides n
            // without leaving a remainder
            if (n % i == 0) {
                // Add divisor i to the list
                divisors.add(i);

                // Add the counterpart divisor
                // if it's different from i
                if (i != n / i) {
                    // Add the counterpart
                    // divisor to the list
                    divisors.add(n / i);
                }
            }
        }

        // Return the list of divisors
        return divisors;
    }

    public static void main(String[] args) {
        int number = 12;
        ArrayList<Integer> divisors = findDivisors(number);

        System.out.print("Divisors of " + number + " are: ");
        for (int divisor : divisors) {
            System.out.print(divisor + " ");
        }
        System.out.println();
    }
}

```

Check if a number is Armstrong Number or not

Problem Statement: Given an integer N, return true it is an Armstrong number otherwise return false.

An Armstrong number is a number that is equal to the sum of its own digits each raised to the power of the number of digits.

Examples

Example 1:

Input: N = 153

Output: True

Explanation: $1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153$

Example 2:

Input: N = 371

Output: True

Explanation: $3^3 + 7^3 + 1^3 = 27 + 343 + 1 = 371$

```
import java.lang.Math;

public class ArmstrongNumber {
    // Function to check if a
    // number is an Armstrong number
    public static boolean isArmstrong(int num) {
        // Calculate the number of
        // digits in the given number
        int k = String.valueOf(num).length();
        // Initialize the sum of digits
        // raised to the power of k to 0
        int sum = 0;
        // Copy the value of the input
        // number to a temporary variable n
        int n = num;
        // Iterate through each
        // digit of the number
        while(n > 0){
            // Extract the last
            // digit of the number
            int ld = n % 10;
            // Add the digit raised to
            // the power of k to the sum
            sum += Math.pow(ld, k);
            // Remove the last digit
        }
    }
}
```

```

        // from the number
        n = n / 10;
    }
    // Check if the sum of digits raised to
    // the power of k equals the original number
    return sum == num ? true : false;
}

public static void main(String[] args) {
    int number = 153;
    if (isArmstrong(number)) {
        System.out.println(number + " is an Armstrong
number.");
    } else {
        System.out.println(number + " is not an Armstrong
number.");
    }
}
}

```

Check if a number is Palindrome or Not

Problem Statement: Given an integer N, return true if it is a palindrome else return false.

A palindrome is a number that reads the same backward as forward. For example, 121, 1331, and 4554 are palindromes because they remain the same when their digits are reversed.

Examples

Example 1:

Input: N = 4554

Output: Palindrome Number

Explanation: The reverse of 4554 is 4554 and therefore it is palindrome number

Example 2:

Input: N = 7789

Output: Not Palindrome

Explanation: The reverse of number 7789 is 9877 and therefore it is not palindrome

```

public class Main {

    // Function to check if a
    // given integer is a palindrome
    static boolean palindrome(int n) {
        // Initialize a variable to
        // store the reverse of the number
        int revNum = 0;
        // Create a duplicate variable to

```

```

// store the original number
int dup = n;
// Iterate through each digit of
// the number until it becomes 0
while (n > 0) {
    // Extract the last
    // digit of the number
    int ld = n % 10;
    // Build the reverse number
    // by appending the last digit
    revNum = (revNum * 10) + ld;
    // Remove the last digit
    // from the original number
    n = n / 10;
}
// Check if the original number
// is equal to its reverse
if (dup == revNum) {
    // If equal, return true
    // indicating it's a palindrome
    return true;
} else {
    // If not equal, return false
    // indicating it's not a palindrome
    return false;
}
}

public static void main(String[] args) {
    int number = 4554;

    if (palindrome(number)) {
        System.out.println(number + " is a palindrome.");
    } else {
        System.out.println(number + " is not a
palindrome.");
    }
}
}

```

Reverse Digits of A Number

Problem Statement: Given an integer N return the reverse of the given number.

Note: If a number has trailing zeros, then its reverse will not include them. For e.g., reverse of 10400 will be 401 instead of 00401.

Examples

Example 1:

Input: N = 12345

Output: 54321

Explanation: The reverse of 12345 is 54321.

Example 2:

Input: N = 7789

Output: 9877

Explanation: The reverse of number 7789 is 9877.

Input Number:

$$7789 \% 10 = 9$$

$$\begin{array}{r} 7789 \\ 10 \end{array} \rightarrow 778 \% 10 = 8$$

$$\begin{array}{r} 778 \\ 10 \end{array} \rightarrow 77 \% 10 = 7$$

$$\begin{array}{r} 77 \\ 10 \end{array} \rightarrow 7 \% 10 = 7$$

$$\begin{array}{r} 7 \\ 10 \end{array} \rightarrow 0$$

Reversed Number:

$$0 \times 10 + 9 = 9$$

$$9 \times 10 + 8 = 98$$

$$98 \times 10 + 7 = 987$$

$$987 \times 10 + 7 = 9877$$

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        // Declare a variable 'n' to
        // store the input integer.
        int n;
        // Prompt the user to enter an
```

```
// integer and store it in 'n'.
Scanner scanner = new Scanner(System.in);
n = scanner.nextInt();
// Initialize a variable 'revNum' to
// store the reverse of the input integer.
int revNum = 0;
// Start a while loop to reverse the
// digits of the input integer.
while(n > 0){
    // Extract the last digit of
    // 'n' and store it in 'ld'.
    int ld = n % 10;
    // Multiply the current reverse number
    // by 10 and add the last digit.
    revNum = (revNum * 10) + ld;
    // Remove the last digit from 'n'.
    n = n / 10;
}
// Print the reversed number.
System.out.println(revNum);
}
}
```