// Working program with FastReader

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.Scanner;

import java.util.StringTokenizer;

public class Main {

static class FastReader {

BufferedReader br;

StringTokenizer st;

public FastReader() {

br = new BufferedReader(new InputStreamReader(System.in));

}

String next() {

while (st == null || !st.hasMoreElements()) {

try{

st = new StringTokenizer(br.readLine());

}

catch (IOException e) {

e.printStackTrace();

}

}

return st.nextToken();

}

int nextInt() {

return Integer.parseInt(next());

}

long nextLong() {

return Long.parseLong(next());

}

double nextDouble() {

return Double.parseDouble(next());

}

String nextLine() {

String str = "";

try{

str = br.readLine();

}

catch (IOException e) {

e.printStackTrace();

}

return str;

}

}

public static void main(String[] args) {

FastReader s=new FastReader();

int n = s.nextInt();

int k = s.nextInt();

int count = 0;

while (n-- > 0) {

int x = s.nextInt();

if (x%k == 0)

count++;

}

System.out.println(count);

}

}

**1) Checking if the number is even or odd without using the % operator:**

Although this trick is not much better than using % operator but is sometimes efficient (with large numbers). Use & operator:

System.out.println((a & 1) == 0 ? "EVEN" : "ODD" );

Example:

num = 5

Binary: “101 & 1” will be 001, so true

num = 4

Binary: “100 & 1” will be 000, so false.

2**) Fast Multiplication or Division by 2**

Multiplying by 2 means shifting all the bits to left and dividing by 2 means shifting to the right.

Example : 2 (Binary 10): shifting left 4 (Binary 100) and right 1 (Binary 1)

n = n << 1; // Multiply n with 2

n = n >> 1; // Divide n by 2

**3) Swapping of 2 numbers using XOR:**

This method is fast and doesn’t require the use of 3rd variable.

// A quick way to swap a and b

a ^= b;

b ^= a;

a ^= b;

**4) Calculating the number of digits directly: To calculate number of digits in a number, instead of looping we can efficiently use log :**

No. of digits in N = Math.floor(Math.log10(N)) + 1;

**5) Efficient trick to know if a number is a power of 2 The normal technique of division the complexity comes out to be O(logN), but it can be solved using O(v) where v are the number of digits of number in binary form.**

/\* Method to check if x is power of 2\*/

static boolean isPowerOfTwo (int x)

{

/\* First x in the below expression is

for the case when x is 0 \*/

return x!=0 && ((x&(x-1)) == 0);

}

6) Sorting Algorithm:

**Arrays.sort() used to sort elements of a array.**

import java.util.Arrays;

Arrays.sort(arr);

**Collections.sort() used to sort elements of a collection.**

// Java program to demonstrate working of Collections.sort()

import java.util.\*;

public class Collectionsorting

{

public static void main(String[] args)

{

ArrayList<String> al = new ArrayList<String>();

al.add("Geeks For Geeks");

al.add("Friends");

al.add("Dear");

al.add("Is");

al.add("Superb");

/\* Collections.sort method is sorting the

elements of ArrayList in ascending order. \*/

Collections.sort(al);

System.out.println("List after the use of" + " Collection.sort() :\n" + al);

}

}

7)binary search that returns index, otherwise -1

Arrays.binarySearch(intArr,intKey)

8) // conversion from String object to StringBuffer

StringBuffer sbr = new StringBuffer(str);

sbr.reverse();

System.out.println(sbr);

// conversion from String object to StringBuilder

StringBuilder sbl = new StringBuilder(str);

sbl.append("ForGeeks");

System.out.println(sbl);

9)

import java.math.BigInteger;

import java.util.Scanner;

public class Example

{

// Returns Factorial of N

static BigInteger factorial(int N)

{

// Initialize result

BigInteger f = new BigInteger("1"); // Or BigInteger.ONE

// Multiply f with 2, 3, ...N

for (int i = 2; i <= N; i++)

f = f.multiply(BigInteger.valueOf(i));

return f;

}

// Driver method

public static void main(String args[]) throws Exception

{

int N = 20;

System.out.println(factorial(N));

}

}

**10)**

import java.math.BigInteger;

class Test

{

public static int gcd(int a, int b)

{

BigInteger b1 = BigInteger.valueOf(a);

BigInteger b2 = BigInteger.valueOf(b);

BigInteger gcd = b1.gcd(b2);

return gcd.intValue();

}

public static long gcd(long a, long b)

{

BigInteger b1 = BigInteger.valueOf(a);

BigInteger b2 = BigInteger.valueOf(b);

BigInteger gcd = b1.gcd(b2);

return gcd.longValue();

}

// Driver method

public static void main(String[] args)

{

System.out.println(gcd(3, 5));

System.out.println(gcd(10000000000L, 600000000L));

}

}

**11)** Efficient trick to know if a number is a power of 2 The normal technique of division the complexity comes out to be O(logN), but it can be solved using O(v) where v are the number of digits of number in binary form.

filter\_none

brightness\_4

/\* Method to check if x is power of 2\*/

static boolean isPowerOfTwo (int x)

{

/\* First x in the below expression is

for the case when x is 0 \*/

return x!=0 && ((x&(x-1)) == 0);

}

12) checking for a prime number: Java has inbuilt isProbablePrime() method in BigInteger class. It returns true if this BigInteger is probably prime(with some certainty), false if it’s definitely composite.

filter\_none

brightness\_4

BigInteger.valueOf(1235).isProbablePrime(1)