Mod of Power

# Description

Design an **efficient algorithm** to calculate

for large values of *B*, *P*, and *M*.

#### NOTE:

Direct calculation of , for large values of B & P, of will exceed the range of any data type.

#### Theorem:

(A × B × C) mod N = **[**(A mod N) × (B mod N) × (c mod N)**]** mod N

Since we want **B** to the power **P** and take modulus **M**, so it is going to be:

(**(B** mod **M)** × **(B** mod **M)** × **(B** mod **M)** × ...*(****P*** *times)*) mod **M**

Input: **Already Implemented**

The first line of input is an integer T (T < 100,000), that indicates the number of test cases. Each case consists of three integer values (in the order B, P, M) will be read one number per line. B and P are integers in the range 0 to 2147483647 inclusive. M is an integer in the range 1 to 46340 inclusive.

Output: **Already Implemented**

The result of the computation, a single integer.

Function: **Implement it!**

long ModOfPower(long B, long P, long M)

It takes the three long integers (B, P, M) and should return the mod value according to the above equation.

ModOfPower.cs includes this method.

# Test Cases

|  |  |  |
| --- | --- | --- |
| **#** | **Input** | **Output** |
| **1** | 3 18132 17 | 13 |
| **2** | 17 1765 3 | 2 |
| **3** | 10 0 40 | 1 |
| **4** | 2374859 3029382 36123 | 13195 |

# C# Help

## Getting the size of 1D array

int size = array1D.GetLength(0);

## Getting the size of 2D array

int size1 = array2D.GetLength(0);

int size2 = array2D.GetLength(1);

## Creating 1D array

int [] array1D = new int [size]

## Creating 2D array

int [,] array2D = new int [size1, size2]

## Sorting single array

Sort the given array "items" in ascending order

Array.Sort(items);

## Sorting parallel arrays

Sort the first array "master" and re-order the 2nd array "slave" according to this sorting

Array.Sort(master, slave);