

## # Modular Arithmetic

$$1. (a+b) \% M = ((a \% M) + (b \% M)) \% M$$

$$2. (a*b) \% M = ((a \% M) + (b \% M)) \% M$$

$$3. (a-b) \% M = ((a \% M) - (b \% M) + M) \% M$$

↓  
just to make sure that the no. remains +ve.

$$4. (a/b) \% M = ((a \% M) - (b^{-1} \% M)) \% M$$

→ Why always  $10^9+7$  is taken?

- Closer to int max value.
- Prime no. so its inverse is possible. (Multiplicative inverse)

→ Modular multiplicative Inverse

$$\begin{aligned}(a/b) \% M &= (A \times B^{-1}) \% M \\ &= (A \% M * \underbrace{B^{-1} \% M}_{\text{modular MI of B}}) \% M\end{aligned}$$

$$A * B = 1 \Rightarrow B \text{ is MI of } A.$$

$$(A * B) \% M = 1 \Rightarrow B \text{ is MMI of } A$$

↓

$$B \in [1, M-1]$$

→ It's not necessary that every no. will have MMI.

MMI is only possible if  $A$  &  $M$  are coprime.

→ To find MMI we can run a loop from 1 to  $M-1$ . So, complexity is  $O(M)$  ( $M=10^9+7$ ). Too much. Need to reduce.

↳ We will use Fermat little theorem

$$A^{M-1} \equiv 1 \pmod{M}, \text{ } M \text{ is prime \& } A \text{ is not a multiple of } M.$$

$$\text{Eg: } M=3 \quad A=2$$

$$2^2 = 4$$

$$4 \% 3 = 1$$

$$A^{M-1} \equiv 1 \pmod{M}$$

$$A^{M-2} \equiv A^{-1} \pmod{M}$$

$$A^{M-2} \% M = A^{-1} \quad \text{iff } M \text{ is prime \& } M \& A \text{ are coprime}$$

↓  
binExp(A, M-2, M)  
↓      ↙      ↘  
         a      b      M

Time complexity:  $O(\log N)$

→ What if M & A are not coprime?

We can use Extended Euclid Algorithm.

↳ There are N children & K toffees. ( $K < N$ ) Count the no. of ways to distribute toffees among N students. ( $K < N < 10^6$ ) Such that each student gets 1 toffee only. Return  $M = 10^9 + 7$

$${}^n C_k = \frac{n!}{(n-k)! k!}$$

```
#include <bits/stdc++.h>
using namespace std;
int M = 1e9 + 7;
int binExp(int a, int b, int m)
{
    a %= m;
    int ans = 1;
    while (b > 0)
    {
        if (b & 1)
        {
            ans = (ans * 1LL * a) % m;
        }
        a = (a * 1LL * a) % m;
        b >>= 1;
    }
    return ans;
}
```

```
const int N = 1e6 + 10;
int fact[N];
int main()
{
    fact[0] = 1;
    for (int i = 1; i < N; ++i)
    {
        fact[i] = (fact[i - 1] * 1LL * i) % M;
    }
    int q;
    cin >> q;
    while (q--) {
        int n, k;
        cin >> n >> k;
        int ans = fact[n];
        int deno = (fact[n - k] * 1LL * fact[k]) % M;
        ans = ans * binExp(deno, M - 2, M);
        cout << ans << endl;
    }
    return 0;
}
```

### Problem statement

[Send feedback](#)

You have been given an integer array/list (ARR) of size N. You have to return an array/list PRODUCT such that PRODUCT[i] is equal to the product of all the elements of ARR except ARR[i]

**Note :**

Each product can cross the integer limits, so we should take modulo of the operation.

Take MOD =  $10^9 + 7$  to always stay in the limits.

**Follow up :**

Can you try solving the problem in O(1) space?

**Detailed explanation** ( Input/output format, Notes, Images )

**Constraints :**

$1 \leq T \leq 100$

$0 \leq N \leq 10^5$

$0 \leq \text{ARR}[i] \leq 10^5$

Time Limit: 1 sec

**Sample Input 1 :**

```
2
3
1 2 3
3
5 2 2
```

**Sample Output 1 :**

```
6 3 2
1 2 3
```



```
int modInverse(int a, int b, int m) {
    a %= m;
    int ans = 1;
    while (b > 0)
    {
        if (b & 1)
        {
            ans = (ans * 1LL * a) % m;
        }
        a = (a * 1LL * a) % m;
        b >>= 1;
    }
    return ans;
}

int *getProductArrayExceptSelf(int *arr, int n) {
    int *prod = new int[n];
    long long product = 1;
    int M = 1000000007; // 10e9 + 7
    int cnt = 0;

    for (int i = 0; i < n; i++) {
        if (arr[i] != 0)
            product = (product * arr[i]) % M;
        else
            cnt++;
    }

    for (int i = 0; i < n; i++) {
        if (cnt > 1) {
            prod[i] = 0;
        } else if (cnt == 1) {
            if (arr[i] == 0) {
                prod[i] = product;
            } else {
                prod[i] = 0;
            }
        } else {
            if (arr[i] != 0) {
                prod[i] = (product * modInverse(arr[i], M - 2, M)) % M;
            } else {
                prod[i] = 0;
            }
        }
    }
    return prod;
}
```

```
long long modInverse(int a, int m) {
    long long m0 = m;
    long long y = 0, x = 1;

    if (m == 1)
        return 0;

    while (a > 1) {
        int q = a / m;
        int t = m;
        m = a % m, a = t;
        t = y;

        y = x - q * y;
        x = t;
    }

    if (x < 0)
        x += m0;

    return x;
}
```

← We can use Extended Euclid Algo to calculate the mod inverse.

## # Euclid's GCD Algorithm

$$\left. \begin{array}{l} \gcd(a, b) = \gcd(b, a \% b) \\ \gcd(a, 0) = a \end{array} \right\} \text{Euclid's GCD}$$

→ GCD of two numbers can never be greater than difference of them.

Now,

$$\begin{aligned} \gcd(a, b) &= \gcd(b, a) \\ \gcd(a, b) &= \gcd(b, a - b) \quad a > b. \end{aligned}$$

Mathematically,

$$a = gx \text{ \& } b = gy.$$

ie  $g$  divides both  $a$  &  $b$ .

$$\gcd(gx, gy) = \gcd(gy, g(x-y))$$

$$\gcd(100, 15)$$

$$\left. \begin{array}{l} 100, 15 \\ 15, 85 \\ 85, 70 \end{array} \right\} \text{repeated subtraction} \Rightarrow \text{division.}$$

$$\text{Hence, } \gcd(b, a \% b) \\ \downarrow \\ a - b$$

→ Eg:  $\gcd(15, 20)$

$$\downarrow \\ \gcd(20, 15)$$

$$\downarrow \\ \gcd(15, 5)$$

$$\downarrow \\ \gcd(5, 0) \Rightarrow \underline{\underline{5}}$$

Time complexity:  $O(\log(\min(a, b)))$

Note: --  $\gcd(a, b)$  is builtin fn for gcd in C++.

```
#include <bits/stdc++.h>
using namespace std;

int gcd(int a, int b) {
    return b == 0 ? a : gcd(b, a % b);
}

int main() {
    int a, b;
    cin >> a >> b;
    cout << gcd(a, b);
    return 0;
}
```

## # Extended Euclid's Algorithm

Applied to simplest diophantine equation,  $ax + by = c$ , where  $a, b, c$  are non zero integers, these methods show that the equation has either no sol<sup>n</sup> /  $\infty$  sol<sup>n</sup>, acc<sup>n</sup> to whether GCD of  $a$  &  $b$  divides  $c$   $\therefore$  not, there are no solutions.

$$ax + by = \gcd(a, b)$$

$\Downarrow$

$$\gcd(b, a \% b) = bx_2 + (a \% b)y_2$$

$$ax + by = bx_2 + (a \% b)y_2$$

$$\text{Dividend} = \text{quotient} \times \text{divisor} + \text{remainder}$$

$(a) \qquad \qquad \qquad \left\lfloor \frac{a}{b} \right\rfloor \qquad \qquad \qquad b \qquad \qquad \qquad (a \% b)$

$$a - \left\lfloor \frac{a}{b} \right\rfloor \cdot b = a \% b$$

$$bx_2 + \left(a - \left\lfloor \frac{a}{b} \right\rfloor \cdot b\right)y_2$$

$$bx_2 + ay_2 - \left\lfloor \frac{a}{b} \right\rfloor \cdot b \cdot y_2$$

$$ay_2 + b\left(x_2 - \left\lfloor \frac{a}{b} \right\rfloor y_2\right) = ax + by$$

$\Rightarrow$  on equating

$$x = y_2 \quad \& \quad y = x_2 - \left\lfloor \frac{a}{b} \right\rfloor \cdot y_2$$

Base case:  $(a, 0)$

$$ax + 0y = \gcd(a, 0) = a$$

$$\Rightarrow y = \text{anything} \quad x = 1$$

```
#include <bits/stdc++.h>
using namespace std;

pair<int, int> extended_gcd(int a, int b) {
    if(b == 0) {
        return {1, 1};
    }
    auto [x2, y2] = extended_gcd(b, a % b);
    int x = y2;
    int y = x2 - (a / b) * y2;
    return {x, y};
}

int main() {
    int a, b;
    cin >> a >> b;
    auto [x, y] = extended_gcd(a, b);
    cout << x << ' ' << y;
    return 0;
}
```

## # Multiplicative Modulo Inverse

$$(A \times B) \% m = 1$$

$$A \times B \equiv 1 \pmod{m}$$

$$\Rightarrow A \times B - 1 \equiv 0 \pmod{m}$$

$$\Rightarrow A \times B - 1 = mq \text{ (some multiple of } m)$$

$$\Rightarrow \begin{array}{ccccccc} A & \cdot & B & + & m & q & = & 1 \\ \downarrow & & \downarrow & & \downarrow & \downarrow & & \uparrow \\ a & & x & & b & & y & \text{gcd}(a, m) \end{array}$$

$$ax + by = \text{gcd}(a, b)$$

Extended Euclid's GCD:

∴ value of B is x in extended gcd algo.

```
long long modInverse(int a, int m) {
    long long m0 = m;
    long long y = 0, x = 1;

    if (m == 1)
        return 0;

    while (a > 1) {
        int q = a / m;
        int t = m;
        m = a % m, a = t;
        t = y;

        y = x - q * y;
        x = t;
    }

    if (x < 0)
        x += m0;

    return x;
}
```

## # Binary Exponentiation

→ What is the need of Binary Exponentiation? We do have  $\text{pow}(x, n)$  f<sup>n</sup>.

The reason is  $\text{pow}(x, n)$  returns the value in double format. Although double can store large values, but it can't store it precisely.

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    double d = 1e20;
    cout << fixed << setprecision(30) << d << endl;
    return 0;
}
// Output: 100000000000000000000.0000000000000000000000000000000000
```

This gives the expected output.

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    double d = 1e24;
    cout << fixed << setprecision(30) << d << endl;
    return 0;
}
// Output: 99999999999999999999983222784.0000000000000000000000000000000000
```

Precision Error

→ Steps to calculate :- We basically use divide and conquer.

$$2^{16} \rightarrow 2^8 \times 2^8$$

$$2^8 \rightarrow 2^4 \times 2^4$$

$$2^4 \rightarrow 2^2 \times 2^2$$

$$2^2 \rightarrow 2^1 \times 2^1$$

$$2^1 \rightarrow 2 \times 2^0$$

$$3^{13} \rightarrow 3 \times 3^{12}$$

$$3^{12} \rightarrow 3^6 \times 3^6$$

$$3^6 \rightarrow 3^3 \times 3^3$$

$$3^3 \rightarrow 3 \times 3^2$$

$$3^2 \rightarrow 3^1 \times 3^1$$

$$3^1 \rightarrow 3^1 \times 3^0$$

$$a^b$$

∴ Complexity will be  $\log b$ .

This is what we call as

Binary Exponentiation.

Let's say  $f(a, b)$  is a f<sup>n</sup> which return  $a^b$ .

$$f(a, b) = \begin{cases} f(a, b/2) \times f(a, b/2), & b \text{ is even} \\ a \times f(a, b/2) \times f(a, b/2), & b \text{ is odd.} \end{cases}$$

→ Code Implementation:- Recursive Approach

```
int M = 1e9 + 7;

int binExpRecur(int a, int b)
{
    if (b == 0)
        return 1;
    long res = binExpRecur(a, b / 2);
    if (b & 1)
    {
        return (a * (1LL * res * res) % M) % M;
    }
    else
    {
        return (1LL * res * res) % M;
    }
}
```

→ Iterative Approach:-

$3^{13} \rightarrow 3^{(1101)_2} \rightarrow 3^{(2^3 + 2^2 + 2^0 + 2^0)} \rightarrow 3^8 \times 3^4 \times 3^0 \times 3^1$   
 Max no. can be  $\log(b)$

```
int binExpIter(int a, int b)
{
    long res = 1;
    while (b > 0)
    {
        if (b & 1)
        {
            res = (res * a) % M;
        }
        a = (a * a) % M;
        b >>= 1;
    }
    return res;
}
```

1e9+7

$\rightarrow a = (1LL * a * a) \% M;$

Complexity:  $\log(b)$

→ Till now we have made certain assumptions i.e.  $a, b, M \leq 10^9$

What if  $a \leq 10^{18} \parallel M \leq 10^{18}$

$$a^b \% M = \{(a \% M)^b\} \% M$$

↑  
a agar M ke range se bada hai toh usko phle hi chota bana lo mod le ke.

$a \leq 10^{18}$   
 $b \leq 10^9$   
 $M \leq 10^9$

→ What will be the problem if  $M \leq 10^{18}$ ?

Even if we take a as long long.

First time

$$a = (10^9 * 10^9) \% M;$$

$$\Rightarrow a = 10^{18}$$

Now, on next iteration

$$a = (10^{18} * 10^{18}) \% M;$$

```
int binExpIter(long long a, long long b)
{
    long long res = 1;
    while (b > 0)
    {
        if (b & 1)
        {
            res = (res * a) % M;
        }
        a = (a * a) % M;
        b >>= 1;
    }
    return res;
}
```

1e9+7

$\rightarrow a = (1LL * a * a) \% M;$

$a \leq 10^9$   
 $b \leq 10^9$   
 $M \leq 10^{18}$



↓  
overflow

$$a * a = a + a + a + \dots + a$$

overflow

↓  
a times

Can do this

$$a + a < 2 \times 10^{18}$$

$$(a+a) \% M < 10^{18}$$

{  
a times

$$(a+a) \% M < 10^{18}$$

$$\text{long long} > 2 \times 10^{18}$$

This enables us to take mod for each step.

→ for this we have Binary Multiplication

$$3 \times 13$$

$$3 \times (1101)_2$$

$$3 \times (8 + 4 + 0 + 1)_2$$

3 → 3	↑	1	LSB
+3 ↓ 6 → x (0 bit)		0	
+6 ↓ 12 +3 → 15		1	
+12 ↓ 24 +15 → 39		1	

→ Code Implementation :-

Complexity :  $O(\log^2 n)$

```
int binExpIter(int a, int b)
{
    long res = 1;
    while (b > 0)
    {
        if (b & 1)
        {
            res = binMultiply(res, a);
        }
        a = binMultiply(a, a);
        b >>= 1;
    }
    return res;
}

int binMultiply(long long a, long long b)
{
    int res = 0;
    while (b > 0)
    {
        if (b & 1)
        {
            res = (res + a) % M;
        }
        a = (a + a) % M;
        b >>= 1;
    }
    return res;
}
```

Dr. Cooper is researching on the growth of bacteria in a colony of bacteria. He found out that a specific factor in the DNA of the bacteria corresponds with the growth factor of that specific type of bacteria. He named this factor The Growth Degree (TGD).

If a bacteria has a TGD of  $N$  and the bacteria colony has a initial alive population of  $A$ , then the number of bacteria in the colony after one day is  $X = A^N$ . But he noticed that some of the bacterias died due to availability of limited resources. So the number of alive bacteria in the colony is  $X = X \% M$  where  $M$  is the resource factor.

You have to calculate the number of alive bacteria in the colony after  $K$  days.

Input Format

The first line of input will contain a single integer  $T$ , denoting the number of test cases.

Each test case contains four space-separated integers  $A, N, M$ , and  $K$  — the number of alive bacteria at zero day, the TGD of the bacteria, the resource factor, and the number of days respectively.

Constraints

- $1 \leq T \leq 50$
- $1 \leq A, N \leq 10^6$
- $1 \leq M \leq 10^9 + 7$
- $1 \leq K \leq 100$

Output Format

For each test case, output on a new line, the number of alive bacteria at the  $k^{th}$  day.

Sample Input 0

```
1
3 2 100 4
```

Sample Output 0

```
21
```



```
#include <bits/stdc++.h>
using namespace std;

int apowb(int a, int b, int m) {
    int res = 1;
    while(b) {
        if(b&1) res = (res*1LL*a)%m;
        a = (a*1LL*a) % m;
        b>>=1;
    }
    return res;
}

int getBactCnt(int a, int n, int m, int k) {
    int res = a;
    while(k) {
        k--;
        res = apowb(res, n, m);
    }
    return res;
}

int main() {
    int t;
    cin>>t;

    while(t--) {
        int a, n, m, k;
        cin>>a>>n>>m>>k;
        int ans = getBactCnt(a, n, m, k);
        cout<<ans<<endl;
    }
    return 0;
}
```

→ What if the value of  $b \leq 10^{18}$  or larger?

for  $b = 10^{18}$ , the code given will run smoother.

Because, ultimately we are reducing the value of  $b$  by right shifting it.

So, this loop will run for at max  $\log b$ .

```

long long
int binExpIter(int a, int b)
{
    long res = 1;
    while (b > 0)
    {
        if (b & 1)
        {
            res = (res * a) % M;
        }
        a = (a * a) % M;
        b >>= 1;
    }
    return res;
}

```

1e9+7

→  $a = (1e9+7 * a) \% M;$

→ Even if we make the power of  $b > 10^{18}$  this code will work but we can't directly give such a large value to  $b$  since  $b \rightarrow$  long long whose max limit is around  $10^{18}$ .

→ But,

$$(a^b)^c \% M$$

↓

$$\{(a \% M)^{(b^c \% M)}\} \% M$$

Mathematically incorrect

$$\text{GCD}(a, b) = 1$$

⇒  $a$  &  $b$  are coprime.

Eg:  $(50^{64^{32}}) \% (10^9 + 7)$

↳ Euler Totient Function (ETF) value of  $N$  is

1 se lekar  $N$  tk jitne bhi no  $N$  se coprime hai wo uski ETF value hai.

ie count of  $K$  |  $1 \leq K \leq N$  &  $K, N$  are coprime.

ETF of 5 → 1, 2, 3, 4

$$\phi(5) = 4$$

$$\phi(n) = n * \prod \left(1 - \frac{1}{p}\right)$$

$p \rightarrow$  distinct prime factors of  $n$ .

$$\phi(5) = 5 \left(1 - \frac{1}{5}\right) = 4$$

$$p = 5$$

→ Our motive  $(a^b) \% M$

According to Euler's theorem:-

$$a^b \equiv a^{b \bmod \phi(m)} \pmod{m}$$

$$\Downarrow$$
$$a^b \% m = a^{b \bmod \phi(m)} \quad \text{--- ①}$$

if  $m$  is prime

$$\Rightarrow \phi(m) = m \left(1 - \frac{1}{m}\right) = m-1$$

ie. ETP value of coprime no. is one less than the no. itself.

∴ Eq ① becomes

$$a^b \% m = a^{b(m-1)} \% m \quad \left. \vphantom{a^b \% m} \right\} m \text{ is prime}$$

$$\rightarrow (50^{64^{32}}) \% (10^9+7)$$

```
#include <bits/stdc++. >
using namespace std;

int M = 1e9 + 7;

int binExpIter(int a, long long b, int m)
{
    int res = 1;
    while (b > 0)
    {
        if (b & 1)
        {
            res = (res * 1LL * a) % m;
        }
        a = (a * 1LL * a) % m;
        b >>= 1;
    }
    return res;
}

int main()
{
    cout << binExpIter(50, binExpIter(64, 32, M-1), M);
    return 0;
}
```

$$(50^{64^{32}}) \% M$$
$$\downarrow$$
$$50^{(64^{32} \% M-1)} \% M$$

→ if  $b$  is very large.

## 372. Super Pow

Medium 328 919 Add to List Share

Your task is to calculate  $a^b \bmod 1337$ , where  $a$  is a positive integer and  $b$  is an extremely large positive integer given in the form of an array.

## Example 1:

Input:  $a = 2, b = [3]$   
Output: 8

## Example 2:

Input:  $a = 2, b = [1,0]$   
Output: 1024

## Example 3:

Input:  $a = 1, b = [4,3,3,8,5,2]$   
Output: 1

## Example 4:

Input:  $a = 2147483647, b = [2,0,0]$   
Output: 1198

## Constraints:

- $1 \leq a \leq 2^{31} - 1$
- $1 \leq b.length \leq 2000$
- $0 \leq b[i] \leq 9$
- $b$  doesn't contain leading zeros.

→ Approach 1:-

- Use the `binExp1b2()` function to achieve the required result.
- Just need to make an adjustment where  $b \gg= 1$
- Need to implement that manually since  $b$  is no longer an integer.
- So write a function which divides a number (stored in form of array) by 2.
- But this will be very hectic.

→ Approach 2:-

$$a^b \% M = a^{b \% \phi(M)}$$

$$1337 = 7 \times 191 \text{ (Not a prime no.)}$$

$$\Rightarrow \phi(1337) = 1337 \times \left(1 - \frac{1}{7}\right) \left(1 - \frac{1}{191}\right) = 6 \times 190 = 1140$$

$$(a^{b \% 1140}) \% 1337 \rightarrow \text{answer.}$$

→ How to calculate  $b \% 1140$  since  $b$  is in form of an array?

$$b = \{4, 3, 3, 8, 5, 2\}$$

$$b \% 1140 = (4 \times 10^5) \% 1140 + (3 \times 10^4) \% 1140 + (3 \times 10^3) \% 1140 + (8 \times 10^2) \% 1140 + (5 \times 10^1) \% 1140 + 2 \% 1140$$

```
class Solution
{
public:
    int binExp(int a, int b, int m)
    {
        a %= m;
        int ans = 1;
        while (b > 0)
        {
            if (b & 1)
            {
                ans = (ans * 1LL * a) % m;
            }
            a = (a * 1LL * a) % m;
            b >>= 1;
        }
        return ans;
    }
    int superPow(int a, vector<int> &b)
    {
        int m = 1337;
        if (a % m == 0)
            return 0;
        int p = 0;
        for (int i = 0; i < b.size(); i++)
        {
            p = (p * 10 + b[i]) % 1140;
        }
        if (p == 0)
            p += 1140;
        return binExp(a, p, m);
    }
};
```