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 $1e9+7$ is taken?

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

→ Modular multiplicative Inverse

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

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

$(A * B)\%M = 1 \Rightarrow B$ 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**

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

Eg: $M = 3 \quad A = 2$

$2^2 = 4 \qquad 4\%3 = 1$

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

$$A^{M-2} \equiv \bar{A}^1 \bmod M$$

$$A^{M-2} \,\%\, M = \bar{A}^1$$

iff M is prime & M & A 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 $M = 10^9 + 7$

$$^nC_k = \frac{n!}{(n-r)!\, r!}$$

```
#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 <= T <= 100
0 <= N <= 10^5
0 <= ARR[i] <= 10^5

Time Limit: 1 sec

Sample Input 1 :

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

Sample Output 1 :

```
6 3 2
```

```cpp
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;
}
```

```cpp
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.

Binary Exponentiation

→ What is the need of Binary Exponentiation? We do have pow(x,n) fn.

The reason is pow(a,n) returns the value in double format. Although double can store large values, but it can't store it precisely.

```cpp
#include <bits/stdc++.h>

using namespace std;

int main()
{
  double d= 1e20;
  cout<< fixed << setprecision(30) << d << endl;
  return 0;
}

// Output: 100000000000000000000.000000000000000000000000000000
```

This gives the expected output.

```cpp
#include <bits/stdc++.h>

using namespace std;

int main()
{
  double d= 1e24;
  cout<< fixed << setprecision(30) << d << endl;
  return 0;
}

// Output: 999999999999999983222784.000000000000000000000000000000
```

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 fn which return a^b.

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

→ Code Implementation :- Recursive Approach

```cpp
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 :-

$3^{13} \rightarrow 3^{(1101)_2} \rightarrow 3^{(8+4+0+1)}$ (1 1 0 1 over $2^3\ 2^2\ 2^1\ 2^0$) $\rightarrow 3^8 \times 3^4 \times 3^0 \times 3^1$

Max no. can be log (b)

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

1e9+7

Complexity : log(b)

→ Till now we have made certain assumptions ie. $a, b, M <= 10^9$

What if $a <= 10^{18}$ || $M <= 10^{18}$

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

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

$a <= 10^{18}$
$b <= 10^9$
$M <= 10^9$

↳ What will be the problem if $M <= 10^{18}$?

Even if we take a as long long.

First time

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

$\Rightarrow a = 10^{18}$

Now, on next iteration

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

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

long long, long long, 1e9+7

$a <= 10^9$
$b <= 10^9$
$M <= 10^{18}$

Overflow

$a * a = a + a + a + \dots + a$ (a times)

Overflow

long long $> 2 * 10^{18}$

Can do this

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

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

$\vdots$ a times

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

This enables us to take mod for each step.

→ for this we have Binary Multiplication

3×13

$3 \times (1\ 1\ 0\ 1)_2$

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

$3 \to 3$

+3 ↓ $6 \to \times$ (0 bit)

+6 ↓ $12 \xrightarrow{+3} 15$

+12 ↓ $24 \xrightarrow{+15} 39$

1 LSB

0

1

1

→ Code Implementation :-

```
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;
}
```

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

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 corrosponds 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
```

```cpp
#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 $b <= 10^{18}$ 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;   → a = (1LL*a*a)%M;
    b >>= 1;
  }
  return res;
}
```

1e9+7

→ 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 → long long whose max limit is around 10^{18}.

→ But,

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

↓

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

Mathematically incorrect

$GCD(a,b) = 1$

⇒ 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 | $1 \le K \le N$ & 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 → distinct prime factors of n.

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

$= 4$

→ Our motive $(a^b) \% M$

$a \equiv b \bmod n$ means when a is divided by n b is the remainder.

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

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

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

∴ Eq (1) becomes

$$a^b \% m = a^{b(m-1)} \% m \quad \Big\} \; m \text{ is prime}$$

→ $(50^{64^{32}}) \% (10^9+7)$

```cpp
#includ  <bits/stdc++. >
 e        h
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;
}
```

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

↓

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

→ if b is very large.

372. Super Pow

Medium 328 919 Add to List Share

Your task is to calculate a^b mod 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:

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

→ Approach 1 :-

- Use the binExp() function to achive the required result.
- Just need to make an adjustment where b>>=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 :-

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

$\therefore 1337 = 7 \times 1337$ (Not a prime no.)

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

$\therefore \left(a^{b \% 1140}\right) \% 1337 \rightarrow$ answer.

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

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

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

```cpp
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);
  }
};
```