Problem A. Maximum Element In A Stack

Time limit: 10 seconds

As an ACM-ICPC newbie, Aishah is learning data structures in computer science. She has already known that a stack, as a data structure, can serve as a collection of elements with two operations:

- push, which inserts an element to the collection, and
- pop, which deletes the most recently inserted element that has not yet deleted.

Now, Aishah hopes a more intelligent stack which can display the maximum element in the stack dynamically. Please write a program to help her accomplish this goal and go through a test with several operations.

Aishah assumes that the stack is empty at first. Your program will output the maximum element in the stack after each operation. If at some point the stack is empty, the output should be zero.

Input

The input contains several test cases, and the first line is a positive integer T indicating the number of test cases which is up to 50.

To avoid unconcerned time consuming in reading data, each test case is described by seven integers n ($1 \le n \le 5 \times 10^6$), p, q, m ($1 \le p, q, m \le 10^9$), SA, SB and SC ($10^4 \le SA, SB, SC \le 10^6$). The integer n is the number of operations, and your program should generate all operations using the following code in C++.

```
int n, p, q, m;
unsigned int SA, SB, SC;
unsigned int rng61(){
    SA ^= SA << 16;
    SA ^= SA >> 5;
    SA ^= SA << 1;
    unsigned int t = SA;
    SA = SB;
    SB = SC;
    SC ^= t ^ SA;
    return SC;
}
void gen(){
    scanf("%d%d%d%d%u%u%u", &n, &p, &q, &m, &SA, &SB, &SC);
    for(int i = 1; i <= n; i++){
        if(rng61() % (p + q) < p)
            PUSH(rng61() % m + 1);
        else
            POP();
    }
}
```

The procedure `PUSH(v)` used in the code inserts a new element with value v into the stack and the procedure `POP()` pops the topmost element in the stack or does nothing if the stack is empty.

Output

For each test case, output a line containing `Case #x: y`, where `x` is the test case number starting from 1, and `y` is equal to $\bigoplus_{i=1}^{n} (i \cdot a_i)$ where a_i is the answer after the i-th operation and $\oplus$ means bitwise xor.

Sample

standard input	standard output
2 4 1 1 4 23333 66666 233333 4 2 1 4 23333 66666 233333	Case #1: 19 Case #2: 1

Hint

The first test case in the sample input has 4 operations:

- `POP()`;
- `POP()`;
- `PUSH(1)`;
- `PUSH(4)`.

The second test case also has 4 operations:

- `PUSH(2)`;
- `POP()`;
- `PUSH(1)`;
- `POP()`.