Problem

On the Code Jam team, we enjoy sending each other pangrams, which are phrases that use each letter of the English lphabet at least once. One common example of a pangram is "the quick brown fox jumps over the lazy dog". Someti es our pangrams contain confidential information — for example, CJ QUIZ: KNOW BEVY OF DP FLUX ALGOR THMS — so we need to keep them secure.

We looked through a cryptography textbook for a few minutes, and we learned that it is very hard to factor products f two large prime numbers, so we devised an encryption scheme based on that fact. First, we made some preparation :

We chose 26 different prime numbers, none of which is larger than some integer N.
We sorted those primes in increasing order. Then, we assigned the smallest prime to the letter A, the second smallest prime to the letter B, and so on.
Everyone on the team memorized this list.
Now, whenever we want to send a pangram as a message, we first remove all spacing to form a plaintext message. T en we write down the product of the prime for the first letter of the plaintext and the prime for the second letter of th plaintext. Then we write down the product of the primes for the second and third plaintext letters, and so on, ending with the product of the primes for the next-to-last and last plaintext letters. This new list of values is our ciphertext. he number of values is one smaller than the number of characters in the plaintext message.

For example, suppose that N = 103 and we chose to use the first 26 odd prime numbers, because we worry that it is t o easy to factor even numbers. Then A = 3, B = 5, C = 7, D = 11, and so on, up to Z = 103. Also suppose that we wa t to encrypt the CJ QUIZ... pangram above, so our plaintext is CJQUIZKNOWBEVYOFDPFLUXALGORITHMS. hen the first value in our ciphertext is 7 (the prime for C) times 31 (the prime for J) = 217; the next value is 1891, an so on, ending with 3053.

We will give you a ciphertext message and the value of N that we used. We will not tell you which primes we used, r how to decrypt the ciphertext. Do you think you can recover the plaintext anyway?

Input

The first line of the input gives the number of test cases, T. T test cases follow; each test case consists of two lines. T e first line contains two integers: N, as described above, and L, the length of the list of values in the ciphertext. The s cond line contains L integers: the list of values in the ciphertext.

Output

For each test case, output one line containing Case #x: y, where x is the test case number (starting from 1) and y is a tring of L + 1 uppercase English alphabet letters: the plaintext.

Limits

$1 \leq T \leq 100$.
Time limit: 20 seconds per test set.
Memory limit: 1 GB.
$25 \leq L \leq 100$.
The plaintext contains each English alphabet letter at least once.

Test set 1 (Visible)

$101 \leq N \leq 10000$.

Test set 2 (Hidden)

$101 \leq N \leq 10100$.

Sample

Input

2
103 31
217 1891 4819 2291 2987 3811 1739 2491 4717 445 65 1079 8383 5353 901 187 649 1003 697 3239 7663 291 123 79 1007 3551 1943 2117 1679 989 3053
10000 25
3292937 175597 18779 50429 375469 1651121 2102 3722 2376497 611683 489059 2328901 3150061 829981 42101 76409 38477 291931 730241 959821 1664197 3057407 4267589 4729181 5335543

Output

Case #1: CJQUIZKNOWBEVYOFDPFLUXALGORITHMS
Case #2: SUBDERMATOGLYPHICFJKNQVWXZ

Solution:

```
t = int(raw_input())

def gcd(a,b):
    if b == 0:
        return a
    return gcd(b, a%b)

def solve():
    n,l = map(int, raw_input().split())
    message = map(int, raw_input().split())
    primes = set()
    for i in xrange(l-1):
        if message[i] != message[i+1]:
            t = gcd(message[i], message[i+1])
            assert t <= n

            primes.add(t)
            primes.add(message[i]/t)
            primes.add(message[i+1]/t)

    vals = sorted(primes)
    d = {}
    idx = 0
    for v in vals:
        d[v] = idx
        idx += 1

    for i in xrange(26):
        test = [chr(d[vals[i]] + ord('A'))]
        pt = vals[i]
        ok = True
```

```
        for j in xrange(l):
            if message[j] % pt != 0:
                ok = False
                break
            pt = message[j] / pt
            test.append(chr(d[pt] + ord('A')))
        if ok:
            return ''.join(test)
    return "!!!"

for __ in xrange(t):
    print "Case #%d: %s" % (__+1, solve())
```