code jam
printf("hello, world!\n");

Time Remaining: 31min 28sec Rank: 3318 Score: 0

Contest scoreboard | **freybupt@gmail.com** | Sign out

Round 1A 2012

A. Password Problem

B. Kingdom Rush

C. Cruise Control

Ask a question

View my submissions

- Submissions

Password Problem

10pt	3 incorrect attempts **3140/3401 users** correct (92%)
10pt	Not attempted **2986 users** attempted

Kingdom Rush

15pt	In progress... **1399/2727 users** correct (51%)
18pt	Not attempted **1310 users** attempted

Cruise Control

17pt	Not attempted **29/114 users** correct (25%)
30pt	Not attempted **14 users** attempted

- Top Scores

SnapDragon	100
pieguy	100
dzhulgakov	100
squark	100
msg555	100
dzetkulict	100
wata	100
Plagapong	100
omeometo	100
falcon112358	100

Problem A. Password Problem

Confused? Read the quick-start guide.

Small input 10 points	Solve A-small You may try multiple times, with penalties for wrong submissions.
Large input 10 points	You must solve the small input first. You will have 8 minutes to solve 1 input file. (Judged after contest.)

Problem

I have a really long password, and sometimes I make a mistake when I type it. Right now I've typed part of my password, but I might have made some mistakes. In particular, I might have pressed the wrong key while typing one or more of the previous characters. Given how likely I was to get each character right, what should I do?

I have three options:

1. Finish typing the password, then press "enter". I know I'll type the rest of the characters perfectly. If it turns out that one of the earlier characters was wrong, I'll have to retype the whole thing and hit "enter" again -- but I know I'll get it right the second time.
2. Hit "backspace" some number of times, deleting the last character(s) I typed, and then complete the password and press "enter" as in option 1. If one of the characters I didn't delete was wrong, I'll have to retype the whole thing and press "enter", knowing I'll get it right the second time.
3. Give up by pressing "enter", retyping the password from the start, and pressing "enter" again. I know I'll get it right this time.

I want to minimize the *expected* number of keystrokes needed. Each character in the password costs 1 keystroke; each "backspace" costs 1 keystroke; pressing "enter" to complete an attempt or to give up costs 1 keystroke.

Note: The "expected" number of keystrokes is the average number of keystrokes that would be needed if the same situation occurred a very large number of times. See the example below.

Example

Suppose my password is "guest" and I have already typed the first two characters, but I had a 40% chance of *making a mistake* when typing each of them. Then there are four cases:

- I typed "gu" without error. This occurs with probability 0.6 * 0.6 = 0.36.
- I typed the 'g' correctly but I made a mistake typing the 'u'. Then I have two letters typed still, but the second one is wrong: "gX". (Here, the 'X' character represents a mistyped letter.) This occurs with probability 0.6 * 0.4 = 0.24.
- I typed the 'u' correctly but I made a mistake typing the 'g': "Xu". This occurs with probability 0.4 * 0.6 = 0.24.
- I made a mistake typing both letters, so I have two incorrect letters: "XX". This

occurs with probability 0.4 * 0.4 = 0.16.

I don't know how many mistakes I actually made, but for any strategy, I can calculate the *expected* number of keys required to use it. This is shown in the table below:

	"gu"	"gx"	"xu"	"xx"	Expected
Probability	0.36	0.24	0.24	0.16	-
Keystrokes if I keep typing	4	10	10	10	7.84
Keystrokes if I press backspace once	6	6	12	12	8.4
Keystrokes if I press backspace twice	8	8	8	8	8
Keystrokes if I press enter right away	7	7	7	7	7

If I keep typing, then there is an 0.36 probability that I will need 4 keystrokes, and an 0.64 probability that I will need 10 keystrokes. If I repeated the trial many times, then I would use 4 keystrokes 36% of the time, and 10 keystrokes the remaining 64% of the time, so the average number of keystrokes needed would be 0.36 * 4 + 0.64 * 10 = 7.84. In this case however, it is better to just press enter right away, which requires 7 keystrokes.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case begins with a line containing two integers, **A** and **B**. **A** is the number of characters that I have already typed, and **B** is the total number of characters in my password.

This is followed by a line containing **A** real numbers: p_1, p_2, ..., $p_{\mathbf{A}}$. p_i represents the probability that I *correctly* typed the i^{th} letter in my password. These real numbers will consist of decimal digits and at most one decimal point. The decimal point will never be the first or the last character in a number.

Output

For each test case, output one line containing "Case #x: y", where x is the case number (starting from 1) and y is the expected number of additional keystrokes I need, not counting the letters I have typed so far, and assuming I choose the optimal strategy. y must be correct to within an absolute or relative error of 10^{-6}.

Limits

$1 \le \mathbf{T} \le 20$.
$0 \le p_i \le 1$ for all i.

Small dataset

$1 \le \mathbf{A} \le 3$.
$\mathbf{A} < \mathbf{B} \le 100$.

Large dataset

$1 \le \mathbf{A} \le 99999$.
$\mathbf{A} < \mathbf{B} \le 100000$.

Sample

Input	Output
3 2 5 0.6 0.6 1 20	Case #1: 7.000000 Case #2: 20.000000 Case #3: 4.500000

```
1
3 4
1 0.9 0.1
```