**Code Jam 2017 - Round 1C**

# Core Training

## Problem

Writing Code Jam problems is hard, so we have built an AI to come up with new ideas. To make the AI as creative as possible, we have given it **N** different "cores", each of which has its own "personality". However, just like people, these cores may become distracted or corrupt or may refuse to work; the i-th core has a *success probability* $\mathbf{P_i}$ of functioning properly. As long as at least **K** of the cores function properly, the AI will function properly. Otherwise, it will probably become evil and trap us in a maze of fiendish puzzles of its own design. And who knows what it might do to Code Jam — it might just write a bunch of tough probability problems!

To prevent this from happening, we plan to train one or more of the cores to become more reliable. We have a total of **U** "training units" that we can use to improve the cores. Spending X units on the i-th core will add X to its success probability. We can divide up the units among the cores however we like, and it is possible that one or more cores may not receive any units. Of course, a core's success probability cannot be increased above 1.

If we assign the training units to maximize the probability that the AI will function properly, what is that probability?

## Solving this problem

This problem has 2 Small datasets and no Large dataset. You must solve the first Small dataset before you can attempt the second Small dataset. You will be able to retry either of the datasets (with a time penalty).

## Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow; each consists of three lines. The first line contains two integers **N** and **K**: the total number of cores, and the minimum number of cores that must succeed for the AI to function properly. The second line contains one rational **U**: the number of training units. The third line contains **N** rational numbers $\mathbf{P_i}$; the i-th of these gives the probability that the i-th core will function properly. All of these probabilities are specified to exactly four decimal places of precision.

## 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 the probability that the AI will function properly if the training units are assigned optimally. `y` will be considered correct if it is within an absolute or relative error of $10^{-6}$ of the correct answer. See the FAQ for an explanation of what that means, and what formats of real numbers we accept.

## Limits

Memory limit: 1 GB.
$1 \leq \mathbf{T} \leq 100$.
$1 \leq \mathbf{N} \leq 50$.

For all i, 0.0000 ≤ $\mathbf{P_i}$ ≤ 1.0000.
0.0000 ≤ **U** ≤ **N** - the sum of all $\mathbf{P_i}$. (There will not be more training units than can be used.)

**Small dataset 1 (Test Set 1 - Visible)**

Time limit: 20 seconds.
**K** = **N**.
(All of the cores must function properly for the AI to function properly.)

**Small dataset 2 (Test Set 2 - Visible)**

Time limit: 40 seconds.
1 ≤ **K** ≤ **N**.

## Sample

### Sample Input

```
4
4 4
1.4000
0.5000 0.7000 0.8000 0.6000
2 2
1.0000
0.0000 0.0000
2 1
0.0000
0.9000 0.8000
2 1
0.1000
0.4000 0.5000
```

### Sample Output

```
Case #1: 1.000000
Case #2: 0.250000
Case #3: 0.980000
Case #4: 0.760000
```

Note that the last two sample cases would not appear in Small dataset 1.

In Sample Case #1, we have enough training units to spend to give all cores a success probability of 1, so the AI will certainly function properly.

In Sample Case #2, both of the cores must function properly for the AI to function properly, so we must give each core at least some training units. The best option turns out to be to train each one up to 0.5. Then the probability that the AI functions properly is 0.5 × 0.5 = 0.25. Any other assignment is inferior; for instance, if we train one core to 0.9 and the other core to 0.1, the probability of success is only 0.9 × 0.1 = 0.09.

In Sample Case #3, we have no training units to spend, and at least one of the two cores must function properly for the AI to function properly. We can approach this by first calculating the probability that the AI does *not* function properly, which happens only if both cores fail to function properly. The probability that both cores fail is (1 - 0.9) × (1 - 0.8) = 0.02. So the probability that at least one core functions properly, and thus that the AI functions properly, is 1 - 0.02 = 0.98.

In Sample Case #4, the optimal strategy is to give all the training units to the second core. That makes the probability of at least one core functioning properly 1 - (0.4 × 0.6) = 0.76. All other options are inferior; for example, giving all the training units to the first core only yields 0.75, and dividing them equally among the cores gives 0.7525.