ACM-ICPC Live Archive

7009 Secret Binary Tree

Alice is preparing a new information hiding scheme and investigating the potential nice property available. Her idea starts by picking up N positive random real numbers r_1, ..., r_N, where the real numbers satisfy $0 < r_i < 1$, for $i = 1, \ldots, N$ and may not be distinct. For any k real numbers x_1, ..., x_k, selected from r_1, ..., r_N with $k \geq 2$, Alice wants to find a number s such that $x_1^s + \ldots + x_k^s = 1$. Alice knows that for each subset (or multi-subset) of k real numbers, $k = 2, \ldots, N$, there is a corresponding s value. So there will be $2^N - N - 1$ such values, which may be duplicate. For convenience, we denote them as s_j, for $j = 1, \ldots, 2N - N - 1$.

To have a better arrangement, Alice wants to build a full binary tree (i.e., each internal node has 2 children) to store the s values in the leaf nodes. Let the tree root be at the 0-th level. For some full binary tree, suppose s_j is stored at the ℓ_j-th level of the binary tree. Alice is interested in the following value

$$\max_{j=1,\ldots,2^N-N-1} \left\{ s_j \left(\frac{1}{2}\right)^{\ell_j} \right\}.$$

We call it the *secret number*. More specifically, for example, suppose we have 4 s-values: 1.0, 1.0, 1.0 and 1.6. Then the following binary tree yields the secret number $\max_j\{s_j \left(\frac{1}{2}\right)^{\ell_j}\} = 0.4$ (see Fig. 9).

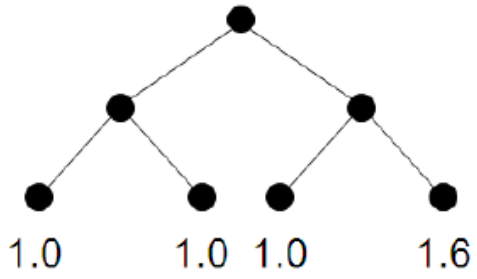

Figure 9: A binary tree yields the secret number $\max_j\{s_j \left(\frac{1}{2}\right)^{\ell_j}\} = 0.4$.

Your task is to write a program to construct a binary tree such that secret number is minimized and output the minimal secret number with precision after 6 decimal places.

Technical Specification

- N: the number of real numbers is at most 18, i.e., $2 \leq N \leq 18$.
- The input real numbers are in the open interval (0, 1) with precision after 3 places of decimal point.
- The precision of final answer is 6 places after decimal point.
- To avoid potential numerical error, you may use the mathematical function $log(x)$ to compute natural logarithm $\ln(x)$ and $pow(x, y)$ to compute x^y in C/C++. Similarly, in Java, you may use $Math.log(x)$ and $Math.pow(x, y)$.

Input

The first line contains an integer $K \leq 5$ indicating the number of test cases. For each test case, the first line contains an integer N which is the number of real numbers. Then N real numbers, between 0 and 1, follow in the next line each with precision after 3 places of decimal point.

Output

For each test case, output the minimal secret number as requested in one line.

Sample Input

```
2
2
0.500 0.500
3
0.500 0.500 0.500
```

Sample Output

```
1.000000
0.396241
```