Huffman´s Greed

In the following we define the basic terminology of trees. A **tree** is defined inductively: It has a **root** which is either an **external node** (a leaf), or an **internal node** having a sequence of trees as its children. An internal node is also called the **parent** of the roots of its child trees. The **level** of a node in a tree is defined inductively: The root has level 0, and the level of a node is 1 more than the level of its parent node.

Every internal node of a **binary tree** has precisely two children, its left sub-tree and its right sub-tree. Every internal node of a **labelled binary tree** is additionally marked with a string, its label. A **binary search tree** is a labelled binary tree where every internal node t satisfies the following condition: All labels of nodes in the left sub-tree of t are less than the label of t which is, in turn, less than all labels of nodes in the right sub-tree of t. For this condition, we assume lexicographic, i.e., alphabetic order on the strings.

An **inorder traversal** of a tree is defined recursively: A leaf is just visited, and for an internal node first its left sub-tree is traversed inorder, then the node itself is visited, finally its right sub-tree is traversed inorder. It follows that an inorder traversal of a binary search tree yields the labels in lexicographic order. Note that binary search trees whose shapes differ may nevertheless yield the same sequence of strings while being traversed inorder.

When a given string s is looked for in a binary search tree, we compare s to the label l of the root. We are done if $s=l$, otherwise if $s < l$ we continue to search in the left sub-tree, and if $s>l$ in the right sub-tree. If a leaf is reached, we know that s is not in the tree.

The number of comparisons performed in such a search procedure depends on s and the actual shape of the search tree. Therefore, there is an interest in constructing binary search trees that store a given sequence of strings but provide as efficient access as possible. Of course, we don't know in advance which strings will be looked up in the tree, so we need to make some assumptions.

Let n be the number of strings that are to be stored in the binary search tree. Let $K_1,...,K_n$ be these strings in lexicographic order. Let $p_1,...,p_n$ and $q_0,...,q_n$ be $2n+1$ non-negative real numbers such that $\sum_{i=1..n} p_i + \sum_{i=0..n} q_i = 1$. The interpretation of these numbers is:

- p_i = probability that the search argument s is K_i.
- q_i = probability that s lies (lexicographically) strictly between K_i and K_{i+1}.

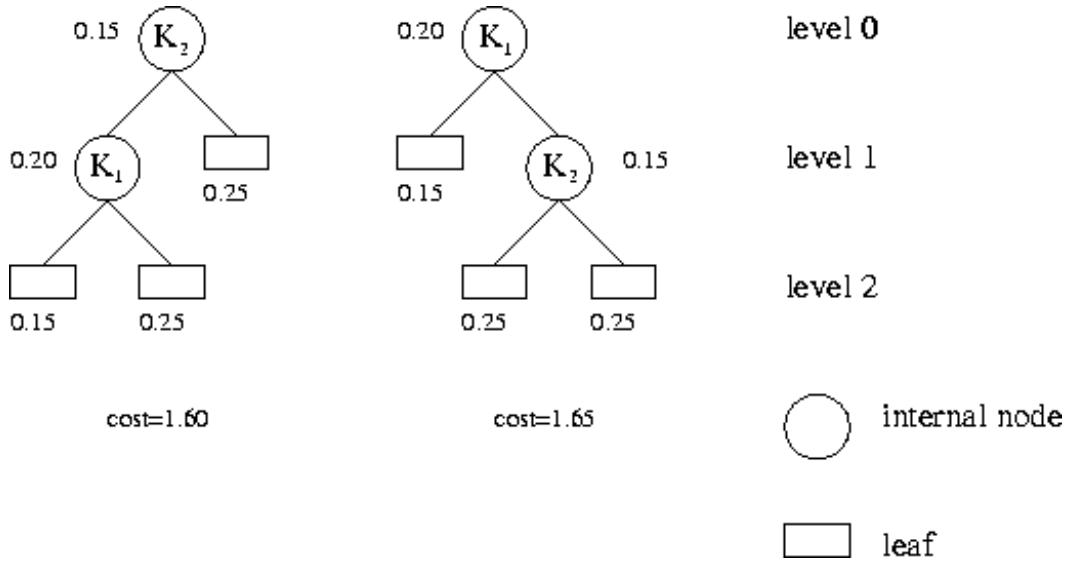
By convention, q_0 is the probability that s is less than K_1, and q_n is the probability that s is greater than K_n. We want to find a binary search tree containing nodes with labels $K_1,...,K_n$ that minimises the expected number of comparisons in the search, namely

$$\text{cost} = \sum_{i=1..n} p_i*(1 + \text{level of internal node } K_i) + \sum_{i=0..n} q_i*(\text{level of leaf between } K_i \text{ and } K_{i+1}).$$

The leaf between K_i and K_{i+1} is that leaf reached in the search for a string s that lies (lexicographically) strictly between K_i and K_{i+1}. Adhere to the convention stated above for the border cases.

The following figure illustrates the first test case of the sample input. It shows the two possible

binary search trees, the probabilities and the associated costs.



Input Specification

The input contains several test cases. Every test case starts with an integer n. You may assume that $1<=n<=200$. Then follow $2n+1$ non-negative integers denoting frequencies. Let s be the sum of all frequencies. You may assume that $1<=s<=1000000$. The probabilities $p_1,...,p_n$ and $q_0,...,q_n$ are calculated in this order by dividing the frequencies by s. The last test case is followed by a zero.

Output Specification

For each test case devise a binary search tree whose cost is minimal for the specified probabilities. Output the integer $cost*s$ for such a tree.

Sample Input

```
2
20 15 15 25 25
35
142 35 58 5 20 5 10 9 15 23 129 4 52 5 38 18 9 7 2 4 266 93 5 18 18 27 5 10 11 180 4 32 21 3 21
0 55 27 36 85 31 58 3 334 0 98 27 113 89 180 0 62 12 0 37 0 3 64 70 0 277 0 0 0 170 0 18 76 27 3 29
0
```

Sample Output

```
160
13637
```