Dept. of Computer Science and Engineering, Czech Technical University in Prague
Faculty of Mathematics and Physics, Charles University in Prague
Faculty of Electrical Engineering and Computer Science, Technical University of Ostrava
Faculty of Informatics and Information Technologies, Slovak University of Technology
Faculty of Informatics, Masaryk University
Faculty of Management Science and Informatics, University of Zilina

CTU Open Contest 2008

Tree Insertions

`insert.c`, `insert.C`, `insert.java`, `insert.p`

All modern banks employ information systems to process their data. The amount of data is enormous. Imagine all the transactions, payments, e-banking, web services, etc. Therefore, advanced data structures must be used to store the data and allow to access them very quickly.

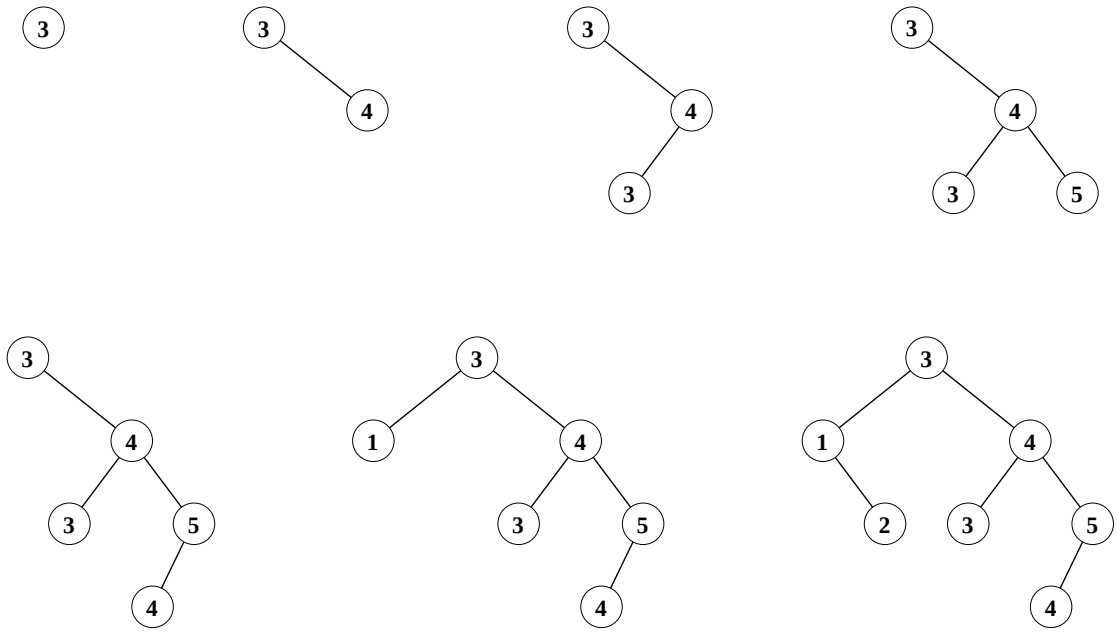
A *Binary Search Tree* (BST) is one example of such a data structure It holds a collection of values with some comparison operation that provides linear ordering on these values.

BST consists of nodes, each of them contains one value and has at most two subnodes: left and right (BST is a binary tree). The left subtree always contains only values strictly less than the node value, the right subtree only values greater than or equal to the node value.

As a consequence, values may easily be looked up by traversing the tree recursively. We begin with the root node and compare its value with the value we are searching for. Depending on the result, we descend either into the left or into the right subtree, but we never need to walk through both.

If we want to insert values to an existing tree, the procedure is also simple. The first value (when the tree is empty) is always put as the root. If the tree already exists, we start with the root node and traverse the tree recursively, as in the case of searching. When the traversal leads us to a missing subnode, we create a new leaf node at that position and assign it the new value.

The figures below show the tree after subsequently adding the following sequence of numbers: 3, 4, 3, 5, 4, 1, and 2.



You may notice that different permutations of the same numbers will often result in the same BST. For example, the tree from the fifth figure above may be constructed by three different input sequences:

- 3, 4, 3, 5, 4
- 3, 4, 5, 4, 3
- 3, 4, 5, 3, 4

Interesting, isn't it? Your task is to compute how many different permutations are there that will result into the same BST.

Input Specification

The input will consist of several trees, each of them specified on two lines. The first line contains a single integer N ($1 \leq N \leq 100$), the number of values in the tree. The second line contains N values separated by a space. These values, if inserted in the given order, form a BST to be examined. All values will be between 0 and 1 000.

The last tree is followed by a line containing a single zero.

Output Specification

For each tree, output the total number of different permutations that would generate the same Binary Search Tree. As you may notice in the Sample Output, this number may exceed 2^{32}.

Sample Input

```
5
3 4 3 5 4
7
3 4 3 5 4 1 2
31
16 8 24 4 12 20 28 2 6 10 14 18 22 26 30 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31
0
```

Output for Sample Input

```
3
45
7483682586183598080000
```