To Heap or Not to Heap

Consider a rooted binary tree with n vertices containing numbers. Each vertex of the tree either has two sons (left son and right son), or no sons. We will call such a tree *heap*, if and only if for all vertices (except the root), the number assigned the vertex is smaller or equal to the parent's number.

Consider a heap and the following function:

```
dfs(vertex){
   print number in the vertex
   if (vertex is not a leaf) {
      dfs(left son of the vertex)
      dfs(right son of the vertex)
   }
}
```

You are given a sequence $a[1..n]$ of n numbers. Your task is to calculate how many heaps will produce this sequence after calling `dfs(root)`. It is guaranteed that the sequence is generated by `generate()` function listed in the input format section below. Since the number of heaps can be very large, output its value modulo 1000000007 $(10^9 + 7)$.

Constraints

$1 \leq n < 2 \times 10^5$
$1 \leq a_i \leq n$

Input Format

The first line contains a single odd integer n. The second line contains n space-separated integers $a_1, a_2, \ldots, a_n$ — the result of `dfs(root)` call.

The sequence is generated by this algorithm:

```
int n, k, ptr
array of integers a[1 .. n]

generate(){
   read odd n
   create array val[1 .. n]
   for each i from 1 to n
      val[i] = random(1, n) //random(l, r) returns uniform integer from [l, r]
   ptr = 1
   sort array val by non-increasing
   gen_heap(val)
}

gen_heap(array values){
   k = size of values
   a[ptr] = values[1]
   ptr = ptr + 1
   if(k == 1)
      return
   create two empty arrays left, right
   for each i from 2 to k - 1
      if(random(1, 2) == 1){
         add values[i] to the end of left
      }else{
         add values[i] to the end of right
      }
   if(left has even size)
      add values[k] to the end of left
   else
      add values[k] to the end of right
   gen_heap(left);
   gen_heap(right);
```

```
}
```

Output Format

Output the number of heaps that will produce the given sequence modulo $1000000007\ (10^9 + 7)$.

Sample Input

```
5
2 1 1 1 1
```

Sample Output

2

Explanation

There are two different heaps:

```
  2        2
 /\       /\
1  1     1  1
  /\     /\
 1  1   1  1
```