Un rapidito

(tiempo límite: 1 segundo)

Ya conocemos el algoritmo QuickSort, el cuál sigue un esquema “divide y vencerás” sin requerir memoria adicional.

```
function quickSort(X, N):
    quickSort(X, 0, N-1)

function quickSort(X, a, b):
    if b-a > 1:
      p = choosePivot(a, b)
      h = partition(X, a, b, p)
      quickSort(X, a, h-1)
      quickSort(X, h+1, b)

function partition(X, a, b, p):
   swap(Xa, Xp)
   i = a
   for j=a+1 to b:
      if (Xj < Xa){
         i++
         swap(Xi, Xj)
   swap(Xi, Xa)
   return i
```

Este ejercicio consiste básicamente en implementarlo. Sin embargo, como ordenar un conjunto de datos no puede ser la salida porque cualquier algoritmo de ordenamiento arrojaría el mismo resultado, la idea es la siguiente: Si se implementa tal cual está descrito previamente y en la función choosePivot SIEMPRE se elige el primer elemento del sub-arreglo como elemento pivote, debe mostrarse la cantidad de llamados recursivos realizados.

Así por ejemplo, si el arreglo original que se requiere ordenar es {3, 1, 5, 2, 4}

- El primer pivote será el 3, quedará en el índice 2 del arreglo original y este quedará {2, 1, 3, 5, 4}, generándose dos llamados recursivos, primero para {2, 1} y luego para {5, 4}
- En el siguiente llamado recursivo, con el sub-arreglo {2, 1}, el pivote será el 2, quedará en el índice 1 del arreglo original y este quedará {1, 2, 3, 5, 4}. Este llamado recursivo no generará más ramas de recursión pues los sub-arreglos resultantes son de tamaño 0 y 1 respectivamente.
- En el siguiente llamado recursivo, con el sub-arreglo {5, 4}, el pivote será el 5, quedará en el índice 4 del arreglo original y este quedará {1, 2, 3, 4, 5}. Este llamado recursivo no generará más ramas de recursión pues los sub-arreglos resultantes son de tamaño 1 y 0 respectivamente.

En total para este ejemplo la cantidad de llamados recursivos fue 3.

Entrada
La primera línea de la entrada contiene la cantidad *C* de casos de prueba (no más de 100). Luego siguen C líneas, cada una con hasta 50000 valores enteros separados entre sí por un espacio en blanco.

Salida
La salida contiene *C* líneas, cada una con la cantidad correspondientes de llamados recursivos

Ejemplo de entrada

```
3
3 1 5 2 4
5 4 3 2 1
6 3 9 7 1 4 2 0 8 5
```

Ejemplo de salida

```
3
4
7
```