2.4

AC

```
//
// Daniel Lopez Marques
// A41
// A47
//

#include <iostream>
#include <fstream>
#include <vector>

// Explicación del algoritmo empleado
//Utilizo  un algoritmo de vuelta atras que comprueba en la mitad derecha si el
//ultimo valor menos el primero es igual que el ultimo indice menos el primero,
//para ver si hay posiciones suficientes para que este todos los numeros. En ese caso,
//hace llamada recursiva hacia la derecha. Si no, comprueba si el valor que falta esta
//entre mitad y mitad+1, y si no llama a la mitad izquierda.



// Coste del algoritmo implementado. Recurrencia, desplegado y orden de complejidad

//T(n)= c1 si n==2
//      T(n/2)+c2 si n>2



// Función que resuelve el problema
int resolver(std::vector<int> v,int ini,int fin) {
    if (ini + 1 == fin) return v[fin] -1;
    else {
        int mitad = (ini + fin) / 2;
        if (v[fin] - v[mitad + 1] != (fin - (mitad + 1))) return resolver(v, mitad + 1, fin);
        else if (v[mitad + 1] - v[mitad] > 1) return v[mitad + 1] - 1;
        else if (v[mitad] - v[ini] != mitad - ini) return resolver(v, ini, mitad);

    }
}



// resuelve un caso de prueba, leyendo de la entrada la
// configuración, y escribiendo la respuesta
bool resuelveCaso() {
    int numElems;
    std::cin >> numElems;
    if (numElems == 0) return false;
    std::vector<int> v(numElems);
    for (int& i : v) std::cin >> i;
    std::cout << resolver(v,0,v.size()-1)<<"\n";
    return true;
}
```

```
int main() {

#ifndef DOMJUDGE
    std::ifstream in("datos2.txt");
    auto cinbuf = std::cin.rdbuf(in.rdbuf());
#endif

    while (resuelveCaso())
        ;

#ifndef DOMJUDGE // para dejar todo como estaba al principio
    std::cin.rdbuf(cinbuf);
    system("PAUSE");
#endif

    return 0;
}
```