Radiation

Martin has a few chemical elements stored in his cupboard. These elements are put in a single line, so that they are placed next to each other. Each element is either on the left or right of some other element.

After one year storing the elements, to Martin's surprise, some of the elements are gone! It turns out that the elements that Martin stores are no ordinary elements. These elements can emit radioactive waves and affect elements located **directly to the right** of the element. If an element is located **directly to the right** of a stronger element, that element will decay in exactly one year.

Being a curious fellow, Martin wants to know how many years will it take until no element will decay. Therefore, he asks his brother, Kevin, a computer science student, to help him create a program that can automatically give the answer to the million-dollar question: "how many years will it take until no elements in Martin's cupboard decay"? That is, for all elements left in the cupboard, there are no elements that are located to the right of a stronger element.

Good luck!

Input

The first line contains a single integer **N** (1 <= N <= 1,000,000), the number of chemical elements inside Martin's cupboard. The next line contains N integers a_0 … a_{n-1} where 0 <= a_i <= 10, each representing the strength of the element at the i^{th} position, from the leftmost to the rightmost element.

Output

Print the **minimum** number of years until no chemical element decays as described above.

Sample Input 1
```
5
1 2 3 4 5
```

Sample Output 1
```
0
```

Sample Input 2
```
5
5 4 3 2 1
```

Sample Output 2
```
1
```

Sample Input 3
```
7
6 5 8 4 7 10 9
```

Sample Output 3
```
2
```

Explanation

Sample Input 1: **0 year**

1. No chemical element will cause another element to decay.

Sample Input 2: **1 year**

1. In the first year, the element with strength "5" will make the element with strength "4" decay. In the same year, the element with strength "4" will also make the element with strength "3" decay, and so on. After 1 year, we're left with a single element, the element with strength "5".

Sample Input 3: **2 years**

1. In the first year, the elements with strengths "5", "4", and "9" will decay because they are located **directly at the right of a stronger element.**
2. At the start of the second year, our elements are like this: 6 8 7 10. This year, the element with strength "7" will decay as it is located at the right of a stronger element. In this case, the element with strength "8" will make it decay.
3. No more elements will decay after the second year.

Hints (applicable for the O(N) solution)

1. Use a stack to solve this problem.
2. Pushing a pair of numbers to a stack, consisting the strength of the element and how long will it stay in the cupboard before it decays, might help you solve this problem.

Skeleton

You are given the skeleton file `Radiation.java`. You should see a non-empty file, otherwise you might be in the wrong directory. The class "Element" is provided as a hint to get the **O(N)** solution.

```
/**
 * Name         :
 * Matric No.   :
 * PLab Acct.   :
 */

import java.util.*;

public class Radiation {

    public void run() {
        // implement your "main" method here...
    }

    public static void main(String[] args) {
        Radiation myChemicalElements = new Radiation();
        myChemicalElements.run();
    }
}
```

```
class Element {
    private int strength;
    private int yearsBeforeDecay;

    public Element(int strength, int yearsBeforeDecay) {
        this.strength = strength;
        this.yearsBeforeDecay = yearsBeforeDecay;
    }

    public int getStrength() {
        return this.strength;
    }

    public int getYearsBeforeDecay() {
        return this.yearsBeforeDecay;
    }
}
```

Notes:

1. You should develop your program in the subdirectory **ex1** and use the skeleton java file provided. You should not create a new file or rename the file provided.
2. If your algorithm is different from the given skeleton, you are free to write a solution according to your own algorithm. You must use **Stack and/or Queue** in your algorithm to solve this problem. You are **not allowed** to use arrays, ArrayList, HashMap, etc. for this problem **for any purpose**. You are allowed to use LinkedList **if and only if** it is **explicitly used** as a **Stack and/or Queue.** Failure to comply will result in **0 marks being given for the sit-in lab.**
3. You are **free to define your own classes (or remove existing ones)** if it is suitable.
4. Please be reminded that the marking scheme is:

   Input : 10%
   Output : 10%
   Correctness : maximum 50% for **O(N)** solution, maximum 40% for **O(N^2)** solution.
   Programming Style : 30% (awarded if you score **at least 20% from the above**):
   - Meaningful comments (pre- and post- conditions, comments inside the code): 10%
   - Modularity (modular programming, proper modifiers [public / private]): 10%
   - Proper Indentation: 5%
   - Meaningful Identifiers (for both method and variable names): 5%

   Compilation Error : Deduction of **50% of the total marks obtained**.