Sky Garden

In the land of Echo Ridge, there is a magical garden located in the sky where you can only see yourself (magical!) and those who are shorter than you. This sky garden is a major tourist attraction in the magical land of Echo Ridge. One day, you decided to visit this magical sky garden by joining a guided tour. After all, seeing short people is very sweet, especially when they are actually sweet-looking!

Once you have arrived at the garden, you are asked to stand in a line. In this line, since you are now in the sky garden of Echo Ridge, you can only see people who are standing in front of you and who are shorter than you. You can still, of course, see yourself. However, you cannot see too far ahead. Once there is a person you cannot see, you are unable to see everyone in front of that person, no matter how short they are. To proceed with the tour, the tour guide needs information on how many people each person in the line can see. However, since it is too tedious to count manually, you decided to help. Being a computer science student in one of Echo Ridge's top universities, you decided to lend your algorithmic expertise to the tour guide by helping him build a simple calculator that can give the answer to his question efficiently.

Before you can enjoy all the beautiful scenes the sky garden has to offer, you quickly take out your laptop and start coding, knowing that the sooner you finish this problem, the sooner you can start enjoying the beautiful scenes at the sky garden of Echo Ridge.

Good Luck!

Input

The first line consists of a single integer N ($1 \leq N \leq 100000$), the number of people in the line. The next line contains N integers, separated by a single space. Each integer denotes the height (in some unit unique to Echo Ridge's Sky Garden) of the i^{th} person in the line.

Output

Print, in a single line, the number of people that each person can see, separated by a single space. There is no whitespace after the last integer and your output should contain a newline character.

Sample Input	Sample Output
7 100 80 60 70 60 75 85	1 1 1 2 1 4 6

Explanation

1. The first person can only see himself.
2. The second person can only see himself as well, since the one standing in front of him is taller than him.
3. The same applies to the third person.
4. The fourth person is taller than the third person, but shorter than the first and second person. Hence, this person can see two people: himself and person number 3.
5. The fifth person can see himself only.
6. The sixth person can see 4 other people: the third person, fourth person, fifth person, and himself.
7. The last person can see everyone except the first person.

Skeleton

You are given the skeleton file **SkyGarden.java**. You should see a non-empty file when you open the skeleton file. Otherwise, you might be in the wrong working directory.

```
import java.util.*;

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

    public static void main(String[] args) {
        SkyGarden sg = new SkyGarden();
        sg.run();
    }
}

class Pair {
    int first;
    int second;

    public Pair(int first, int second) {
        this.first = first;
        this.second = second;
    }

    public int getFirst() { return this.first; }

    public int getSecond() { return this.second; }

    public void setFirst(int newFirst) { this.first = newFirst; }

    public void setSecond(int newSecond) { this.second = newSecond; }
}
```

Hint: Pushing a pair of integers to a stack might help you solve this problem.

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 main algorithm to solve this problem. <Restriction on Arrays/etc. but they need to use array somehow>.
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 : 50%
   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**.