**1. Problem discussion:**

1• You are given a number n, representing the size of array a. 2• You are given n numbers, representing the prices of a share on n days. 3• You are required to find the stock span for n days. 4• Stock span is defined as the number of days passed between the current day and the first day before today when the price was higher than today.

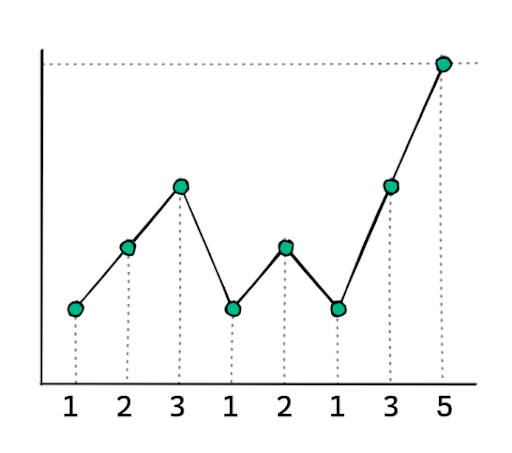
**2. Approach:**

Naive approach:

Time Complexity: O(n2) Space Complexity: O(n) You might think of a very naive approach. To find the next greater element to the left by running a loop. But you will realize this is not a very good approach. The outer loop takes n iterations and in the inner loop for i-th value at the worst case, you will have to perform i number of operations. So the total operations become: 1+2+3+4+ ... = n\*(n+1)/2 = O(n2). So we will look into a better approach. By the way, can you tell what is the worst case?Yes. if the numbers are all increasing.

Trivial Approach:

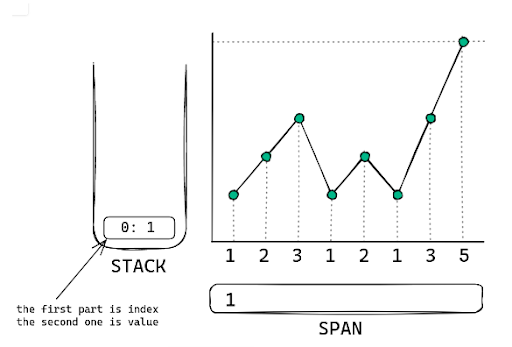
Time Complexity: O(n) Space Complexity: O(n) Let's consider an array of elements: [1,2,3,1,2,1,3,5]. To show it visually it will look like this:

****

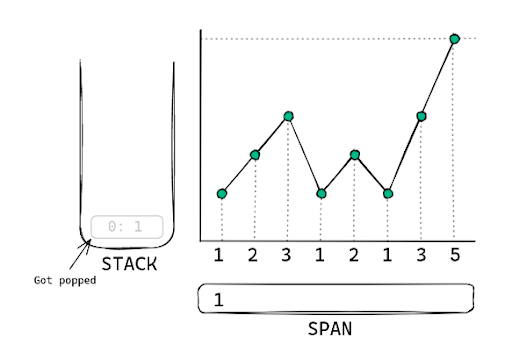
For every element we will do these operations in order:

1• POP => Pop all the elements in the stack smaller than the current value. 2• Answer => Now the top of the stack contains the index of the value which is greater than the current value. 3• PUSH => Push the index of the current element.

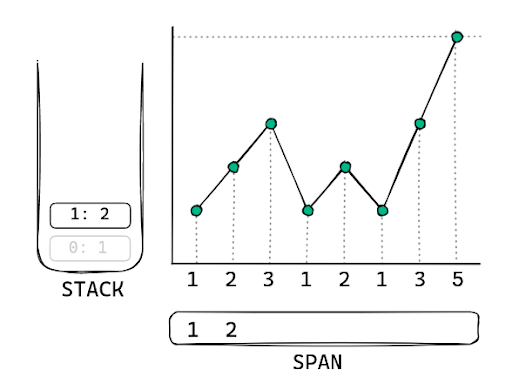
NOTE: The stack will only contain the index because it will help us calculate the answer without much hassle. If the index at the top of the stack is j and the current index is i, then the span for that index will be j-i. We will maintain a stack. The span of the first element is always 1. Because there is no element before it. Also, we will push it to the stack.

****

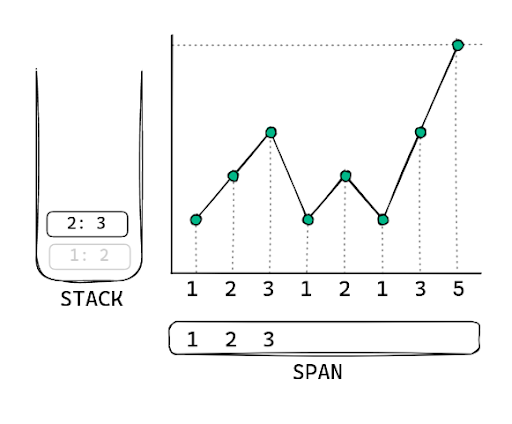
For the next element, the value at the top of the stack is smaller hence will be popped. The stack becomes empty.

****

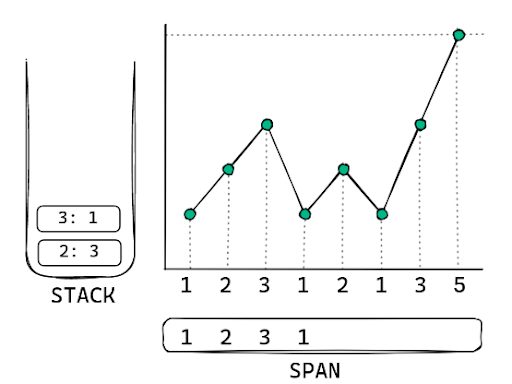
So, here the answer would be 2. Basically after popping if we get an empty stack, we will then put the value of index+1 as span (0-indexing based).

****

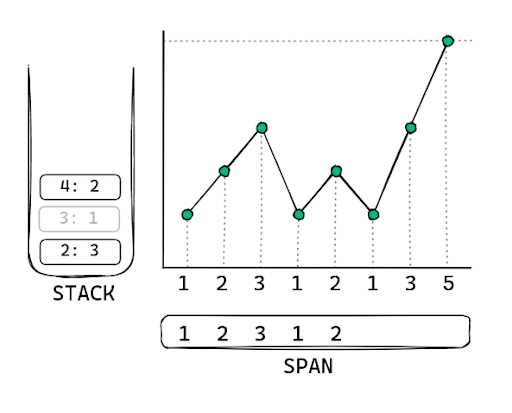
For the third element, we will pop the stack, the stack again becomes empty. The value of span will be index+1 i.e 3, and we will push the new value.

****

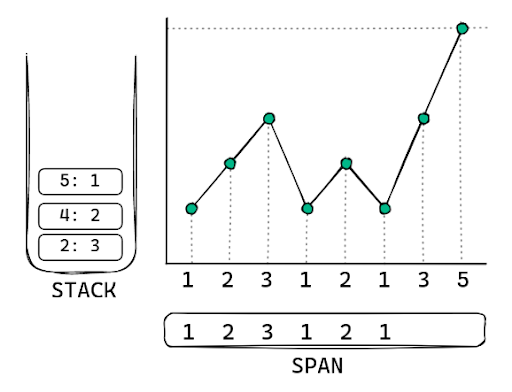
Now for the 3rd index. The value is smaller than the value at top of the stack. NOTE: We will store the index of the elements in the stack, but while comparing we will consider the value at that index. Here to make it easy I am drawing the value along with the index in the stack. Now to calculate the span for the current value, we will do i-peek(), 3-2 = 1. Now we will push it into the stack.

****

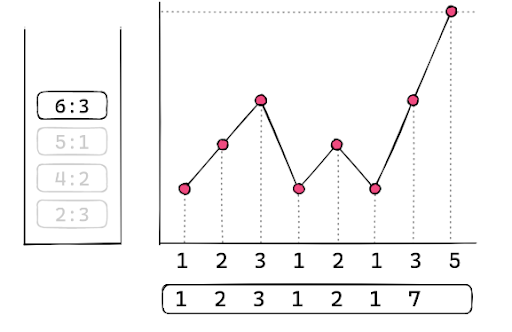
For the next index. The value is 2, and the value the top of the stack points to is 1. Hence it needs to pop. But then it will not pop anymore because the value now is greater than 3. POP operation completes. Now we calculate the span for the current index. Which is 4-2 = 2. Now we push the current index.

****

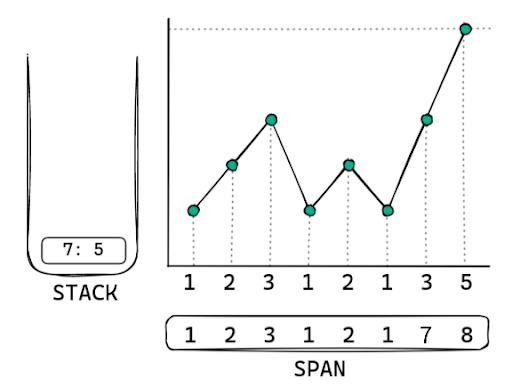
Again the current value is smaller than the top element, so nothing to pop. The span will be 5-4 = 1. And we will push the current element.

****

Now the current value is bigger than the stack's top. So it will be popped. Then again it's bigger so it will be popped again. Now, it's no longer bigger, so we will stop here and calculate our span. Also, note that the value of the current index is the same as that of index 2 but we will still pop it because according to the question we have to find higher. (If the question was equal to or higher then we won't consider this). So here since the stack is empty the value is index+1 i.e 7. And we will push the current element.

****

Let's look at the last element. It is greater than all the elements in the stack and hence will pop all the elements. And since the stack is empty the result for this one will be index+1 i.e 8.

****

This is how we will get the stock span.

**3. Code Discussion:**

1• Iterate through the string and :

1• Create a new stack and initialize the span array's first value, 0. 2• Loop through all the elements from 1 to n.

1• If the stack top is greater than equals to the current value keep popping. 2• (a) If the stack now is empty then span[index] will be index+1. 3• (b) Else span[index] will have index - (value at the top of the stack). 4• Finally push the current index into the stack.

3• Return span array.

**4. Code:**

ConsoleJava

import java.io.\*;

import java.util.\*;

public class Main {

public static void display(int[] a) {

StringBuilder sb = new StringBuilder();

for (int val : a) {

sb.append(val + "

");

}

System.out.println(sb);

}

public static void main(String[] args) throws Exception {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int n = Integer.parseInt(br.readLine());

int[] a = new int[n];

for (int i = 0; i < n; i++) {

a[i] = Integer.parseInt(br.readLine());

}

int[] span = solve(a);

display(span);

}

public static int[] solve(int[] arr) {

int[] span = new int[arr.length];

Stack< Integer> st = new Stack<>();

st.push(0);

span[0] = 1;

for (int i = 1; i < arr.length; i++) {

while (st.size() > 0 && arr[i] >= arr[st.peek()]) {

st.pop();

}

if (st.size() == 0) {

span[i] = i + 1;

} else {

span[i] = i - st.peek();

}

st.push(i);

}

return span;

}

}

**5. Analysis:**

Time Complexity:

You'll have to figure out why it's O(n). Let's take a look at one element's life cycle. Every element in the stack will be pushed into it once. In addition, the element will only be popped one time. Because it is lost from the stack once it is popped. As a result, each element is linked to two events. There was only one push and one pop. Both are operations that take place in the same amount of time. When we do the same thing with n elements, we get n\*O(1), or O(n) operations. As a result, the time complexity is O(n).

Space Complexity:

We're using an auxiliary stack, which will be completely filled with n numbers in the worst-case scenario. Could you please explain what that is? If the numbers are in ascending order, yes. There will be nothing to pop then, only elements to push. There's also a span array where we had to store the span results. As a result, the overall space complexity is O(n). Hopefully, you've grasped the concept of the Algorithm. If you're not sure what you're doing, watch the entire video and run the code by hand. That way, you'll be able to grasp the algorithm's logic. Best wishes and keep learning!!