Stock Span

Problem Description: The span si of a stock's price on a certain day i is the minimum number of consecutive days (up to the current day) the price of the stock has been less than its price on that ith day. If for a particular day, if no stock price is greater then just count the number of days till 0th day including current day. Given an input array with all stock prices, you need to return the array with corresponding spans of each day.

Sample Input:

```
8
60 70 80 100 90 75 80 120
```

Sample Output:

```
1 2 3 4 1 1 2 8
```

How to approach?

One easy and intuitive method to approach this question would be to iterate through the array and for every element, to again iterate from that element to the beginning of the array and count the number of consecutive elements that are smaller than or equal to the current element. The time complexity of this approach is $O(n^2)$, where n is the size of the array. We can do better.
One reason why the time complexity is quadratic is because we are redoing work which has been done before. Let's take a look at the sample input. When we reverse-iterated for element '70' and got 2 for that position, we knew that 70 is the largest element in the array till its index. So for the next iteration when we encounter 80, we know that 80 is larger than 70 and 70 was the largest element in its prefix (the part of the array from index 0 to the current index), so we automatically can conclude that 80 is the largest element in its prefix and directly put 3 for that position. Same for position of element 100.
To avoid the extra comparisons, we can use a stack to keep track of elements that we've encountered so far and remove elements that have a higher element after them.

Let’s do a dry run on the sample input which is:

60	70	80	100	90	75	80	120
0	1	2	3	4	5	6	7

Let’s also maintain a stack which is going to store the **indices** of the concerned elements, not the elements themselves.

Let’s iterate through the array using the variable i and maintain an array called ‘span’ which will store the span values for each element.

1. (i = 0) The stack is empty right now. So span[i] will be directly equal to i+1. No need to do any other work. Remember, an empty stack denotes that the current element is the largest in its prefix. After we set the value of span[i], we’ll push the index (which is 0) into the stack. Note, that we push ‘0’, not ‘60’.
2. (i = 1) The stack is not empty. This denotes that the current element may or may not be the largest in its prefix. We can’t be sure just yet. So let’s iterate through the stack and remove those indices from stack’s top for which the current element is larger than the element at stack’s top’s index. So in our case the stack had 0. Since arr[1] > arr[stack-top], we can pop the top element from stack. Now, the stack is empty and this means arr[i] is the largest in its prefix so span[i] can be directly assigned the value of i+1 and then we push ‘i’ to the stack. If you notice, there will be similar cases for i=2 and i=3. So let’s move on to...
3. (i = 4) The stack is not empty. The only value it contains is ‘3’, the index of 100. This case is different because arr[i] is less than arr[stack-top]. This signifies that the nearest element to ‘90’ which is larger than it and to its left is at index ‘3’. In this case, we don’t pop stack’s top. Instead, we find the distance between the current element and stack’s top which will be `i - stacktop` or 4 - 3 or 1. So we put span[i] = 1 and push ‘i’ to stack. Continuing on in this way we can fill the span array completely.

The pseudo-code for this approach is shown on the next page.

```
function stockSpan(arr,n):
  span <- array of size n
  stk <- stack of integers
  for(i = 0; i < n; ++i):
     while(stk is not empty and arr[stk.top()] < arr[i]):
       stk.pop()

     if(stk is empty):
       span[i] <- i + 1
     else:
       span[i] <- i - stk.top()

     stk.push(i)

  return span
```