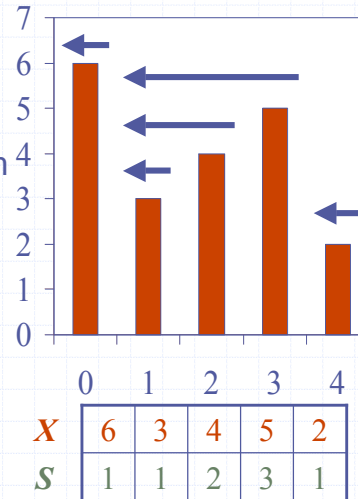


Computing Spans (not in book)

- ◆ We show how to use a stack as an auxiliary data structure in an algorithm
- ◆ Given an array X , the span $S[i]$ of $X[i]$ is the maximum number of consecutive elements $X[j]$ immediately preceding $X[i]$ and such that $X[j] \leq X[i]$
- ◆ Spans have applications to financial analysis
 - E.g., stock at 52-week high



Quadratic Algorithm

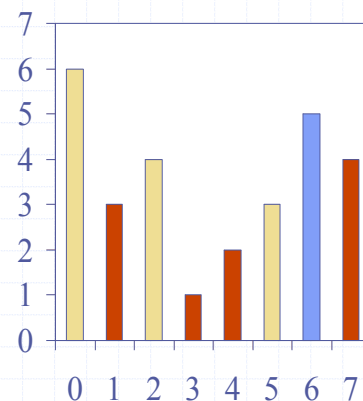
```

Algorithm spans1( $X, n$ )
Input array  $X$  of  $n$  integers
Output array  $S$  of spans of  $X$ 
 $S \leftarrow$  new array of  $n$  integers
for  $i \leftarrow 0$  to  $n - 1$  do
     $s \leftarrow 1$ 
    while  $s \leq i \wedge X[i - s] \leq X[i]$ 
         $s \leftarrow s + 1$ 
     $S[i] \leftarrow s$ 
return  $S$ 
    
```

- ◆ Algorithm *spans1* runs in $O(n^2)$ time

Computing Spans with a Stack

- ◆ We keep in a stack the indices of the elements visible when "looking back"
- ◆ We scan the array from left to right
 - Let i be the current index
 - We pop indices from the stack until we find index j such that $X[i] < X[j]$
 - We set $S[i] \leftarrow i - j$
 - We push i onto the stack



Linear Algorithm

- ◆ Each index of the array
 - Is pushed into the stack exactly one
 - Is popped from the stack at most once
- ◆ The statements in the while-loop are executed at most n times
- ◆ Algorithm *spans2* runs in $O(n)$ time

```

Algorithm spans2( $X, n$ )
 $S \leftarrow$  new array of  $n$  integers
 $A \leftarrow$  new empty stack
for  $i \leftarrow 0$  to  $n - 1$  do
    while  $(\neg A.isEmpty() \wedge X[top()] \leq X[i])$  do
         $A.pop()$ 
    if  $A.isEmpty()$  then
         $S[i] \leftarrow i + 1$ 
    else
         $S[i] \leftarrow i - top()$ 
     $A.push(i)$ 
return  $S$ 
    
```