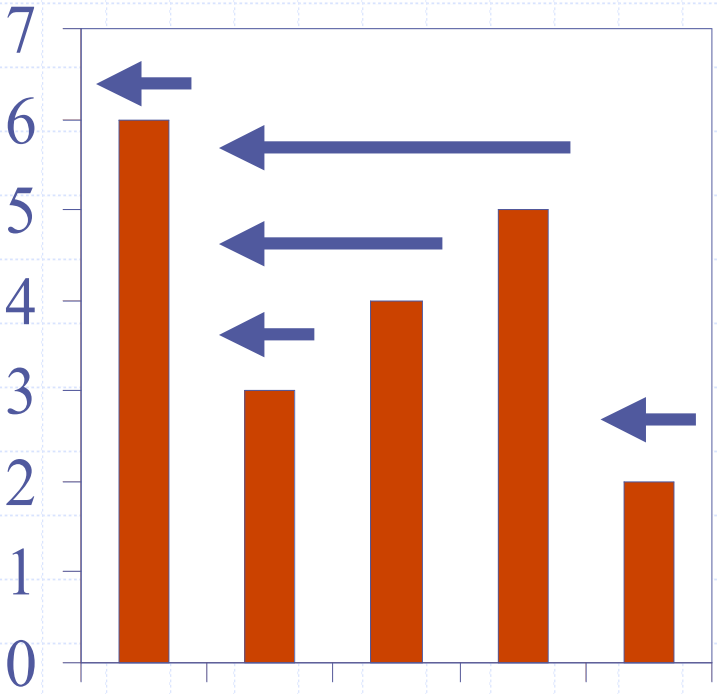
Computing Spans (not in book)

- We show how to use a stack as an auxiliary data structure in an algorithm
- Given an 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



	0	1	2	3	4
X	6	3	4	5	2
S	1	1	2	3	1

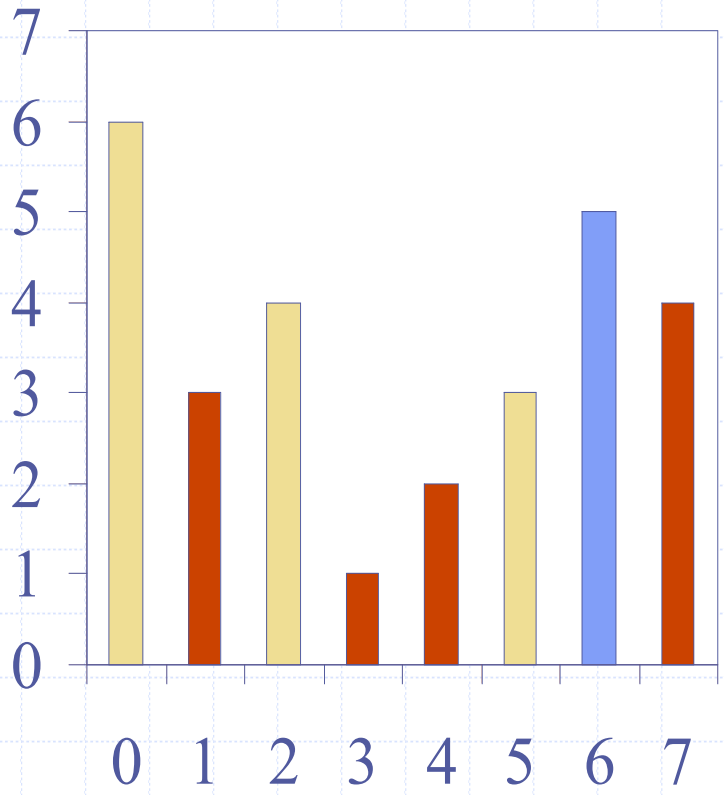
Quadratic Algorithm

```
Algorithm spans1(X, n)                                  #
    Input array X of n integers
    Output array S of spans of X                        #
    S ← new array of n integers                         n
    for i ← 0 to n – 1 do                               n
        s ← 1                                           n
        while s ≤ i ∧ X[i – s] ≤ X[i]                   1 + 2 + …+ (n – 1)
            s ← s + 1                                   1 + 2 + …+ (n – 1)
        S[i] ← s                                        n
    return S                                            1
```

- 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 x 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 ← new array of n integers                 n
    A ← new empty stack                         1
    for i ← 0 to n – 1 do                       n
        while (¬A.isEmpty() ∧
                X[top()] ≤ X[i] ) do            n
            A.pop()                             n
        if A.isEmpty() then                     n
            S[i] ← i + 1                        n
        else
            S[i] ← i – top()                    n
        A.push(i)                               n
    return S                                    1
```