An Inefficient Algorithm

- There is a straightforward way to compute the span of a stock on each of *n* days:

```
Algorithm computeSpans1(P):
  Input: an n-element array P of numbers such that
         P[i] is the price of the stock on day i
  Output: an n-element array S of numbers such that
         S[i] is the span of the stock on day i
  for $i \leftarrow 0$ to $n - 1$ do
    $k \leftarrow 0$
    $done \leftarrow$ false
    repeat
      if $P[i - k] \leq P[i]$ then
        $k \leftarrow k + 1$
      else
        $done \leftarrow$ true
    until $(k > i)$ or $done$
    $S[i] \leftarrow k$
  return $S$
```

- The running time of this algorithm is (ugh!) $O(n^2)$. Why?

An Efficient Algorithm

- The code for our new algorithm:

```
Algorithm computeSpan2(P):
  Input: A nn-elementarray  P ofnumbersrepresenting
         stock prices
  Output: An n-element array S of numbers such that
         S[i] is the span of the stock on day i
  Let D be an empty stack
  for $i \leftarrow 0$ to $n - 1$ do
    $done \leftarrow$ false
    while not(D.isEmpty() or done) do
      if $P[i] \geq P[D.top()]$ then
        D.pop()
      else
        $done \leftarrow$ true
    if D.isEmpty() then
      $h \leftarrow -1$
    else
      $h \leftarrow$ D.top()
    $S[i] \leftarrow i - h$
    D.push(i)
  return S
```

- Let's analyze computeSpan2's run time...