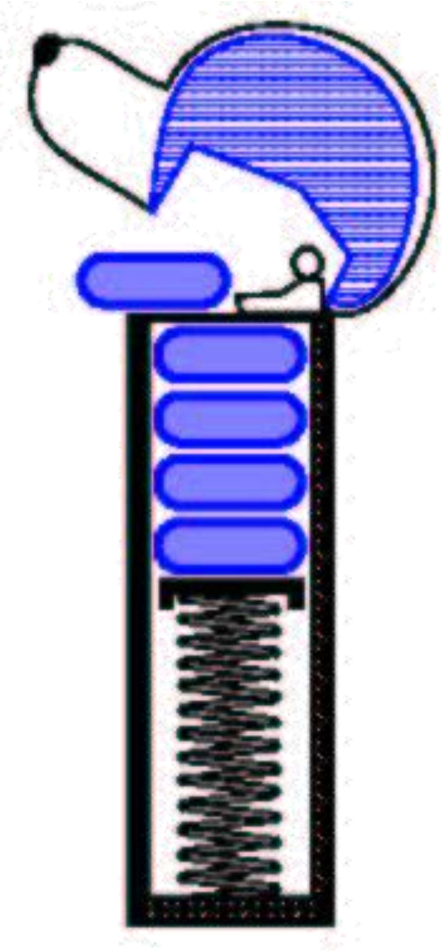
Stack

The Stack ADT

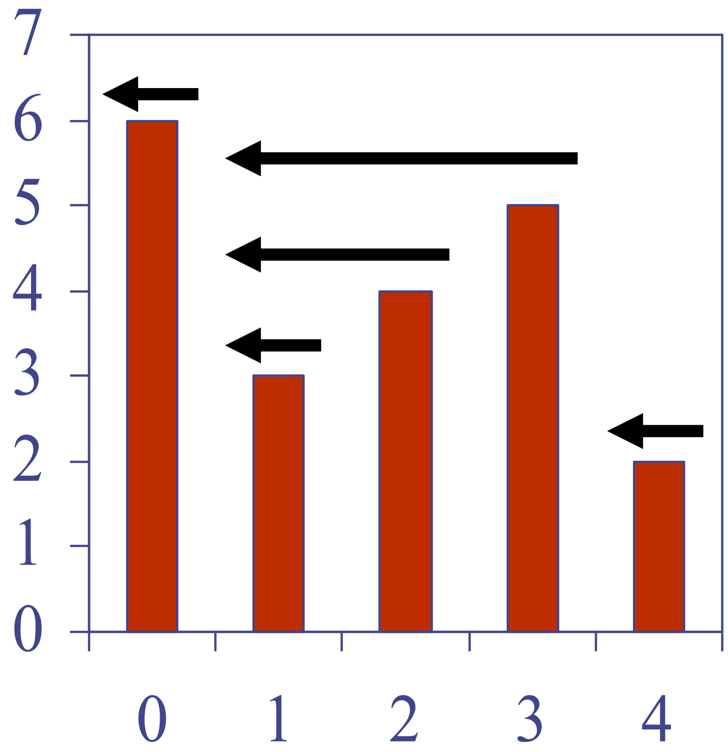


- The Stack ADT stores arbitrary objects
- Insertions and deletions follow the last-in first-out scheme
- Think of a spring-loaded plate dispenser
- **LIFO**
  = Last In First Out

- Main stack operations:
  - push(object): inserts an element
  - object pop(): removes and returns the last inserted element
- Auxiliary stack operations:
  - object top(): returns the last inserted element without removing it
  - integer len(): returns the number of elements stored
  - boolean is_empty(): indicates whether no elements are stored

Computing Spans (not in book)

- Using 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



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

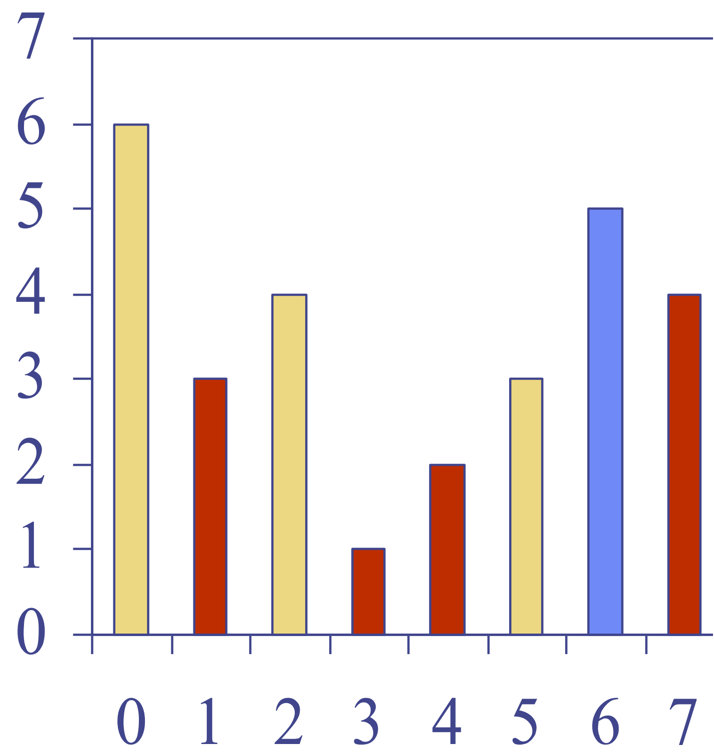
Quadratic Algorithm

```
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

X = [6, 3, 4, 1, 2, 3, 5, 4]

```
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.is_empty() ∧
                X[A.top()] ≤ X[i] ) do       n
            A.pop()                          n
        if A.is_empty() then
            S[i] ← i + 1                     n
        else
            S[i] ← i − A.top()               n
        A.push(i)                            n
    return S                                 1
```