CMPSC 250
Analysis of algorithms

Dr. Aravind Mohan
Allegheny College

January 22, 2018
Lecture 02: A deeper look at Stacks ADT
What is a Stack?

- A stack is a container of objects that are inserted and removed according to the last-in-first-out (LIFO) principle.
- Objects can be inserted at any time, but only the last (the most-recently inserted) object can be removed.
- Inserting an item is known as “pushing” onto the stack. “Popping” off the stack is synonymous with removing an item.
Analogy

- A plate dispenser is an analogy.
Stack Operations

A stack is an ADT that supports four main methods:

1. **new()**:ADT - Creates a new stack.
2. **push(S:ADT, o:element)**:ADT - Inserts object o onto top of stack S.
3. **pop(S:ADT)**:ADT - Removes the top object of stack S; if the stack is empty an error occurs.
4. **top(S:ADT)**:element - Returns the top object of the stack, without removing it; if the stack is empty an error occurs.
Stack Operations

- The following support methods could also be defined:
  - \texttt{size(S:ADT):integer} - Returns the number of objects in stack S.
  - \texttt{isEmpty(S:ADT):boolean} - Indicates if stack S is empty.

- The following axioms dictates the scope of the operations in the stack:
  1. \texttt{Pop(Push(S,v)) = S}
  2. \texttt{Top(Push(S,v)) = v}
JAVA Stuff

- Given the stack ADT, we need to code the ADT in order to use it in the programs. We need two constructs: interfaces and exceptions.
- An interface is a way to declare what a class is to do. It does not mention how to do it.
- An exception is yet another programming construct, useful for handling errors. When we find an error or an exceptional case, we just throw an exception.
  - Flow of control
  - Delegating upwards
Array Based Stack Implementation

Let us look at the source code written in JAVA.
Application: Time Series

The span $S_i$ of a stock’s price on a certain day $i$ is the maximum number of consecutive days (up to the current day) the price of the stock has been less than or equal to its price on day $i$. 
Application: Time Series

**An inefficient algorithm**

Algorithm `computeSpans1(P)`:

Input: an n-element array `P` of numbers such that `P[i]` is the price of the stock on a day `i`.

Output: an n-element array `S` of numbers such that `S[i]` is the span of the stock on a day `i`.

for `i <- 0` to `n-1` do

`h <- 0`; `done <- false`;

repeat

if `P[i-h] <= P[i]` then `h <- h + 1`

else

`done <- true`

until `(h = i)` or `done`

`S[i] <- h`

return `S`
Application: Time Series

- Class activity: A Stack can help make it more efficient
- Discuss how to do it?
  - Post your suggestions on Slack so that you can get the Attendance and Class participation points.
  - I will give the solution in next class.
Questions