Analysis of Algorithms
Case Studies

Reading Assignment
Chapter 3 (except 3.4) and
Chapter 4.5

Review of some discrete math

- $1 + 2 + 3 + \ldots. + n = n(n + 1)/2$
- Why?????
- See P. 106 in the textbook

Case Study

- Problem: **Prefix averages** of a sequence of numbers
- Given an array X of n numbers, compute an Array A such that A[i] is the average of elements X[0], X[1], …, X[i] for i = 0, 1, …, n-1
- What is the pseudocode for this problem?
- What is the running time for our solution?
- Can we do better?

Another Case Study Application

(Chapter 4.5 in the textbook)

- Given a series of n daily price quotes for a stock, we call the **span** of the stock's price on a certain day the maximum number of consecutive days up to the current day that the price of the stock has been less than or equal to its price on that day.

- More formally:
  - ✍ assume that price quotes begin with day 0 and that day p_i denotes the price on day i.
  - ✍ The span s_i on day i is equal to the maximum integer k such that $k <= i + 1$ and $p_j <= p_i$ for $j = i - k + 1, \ldots, i$.
  - ✍ Given the prices $p_0, p_1, \ldots, p_{n-1}$, consider the problem of computing the spans $s_0, s_1, \ldots, s_{n-1}$.

One possible solution

Here is one possible solution in pseudocode:

```
Algorithm computeSpans1(P):
    Input:  An n-element array P of numbers
    Output:  An n-element array S of numbers such that S [i ] is the largest
    integer k such that k <= i + 1 and P [j ] <= P [i ] for j = i-k+1, ..., i
    for i = 0 to n – 1 do
            k ✍ 0
            done ✍ false
            repeat
               if P[i-k] <= P[i] then
                          k ✍ k + 1
               else
                          done ✍ true
            until (k > i) or done
            S[i] ✍ k
    return array S
```

Analyzing the running time of this solution....

```
Algorithm computeSpans1(P):
    (input P, output S)
    for i = 0 to n – 1 do
        k ✍ 0                                  // O(n)
        done ✍ false
        repeat
            if P[i-k] <= P[i] then
                k ✍ k + 1
            else
                done ✍ true
        until (k > i) or done
        S[i] ✍ k
    return array S
```

Analyzing the running time of this solution....

```
Algorithm computeSpans1(P):
    (input P, output S)
    for i = 0 to n – 1 do
        k ✍ 0                              // O(n)
        done ✍ false
        repeat
            if P[i-k] <= P[i] then         // 1 + 2 + 3 … + n =
                k ✍ k + 1                  // n(n+1)/2 = O(n^2)
            else
                done ✍ true
        until (k > i) or done
        S[i] ✍ k
    return array S
```

Another solution

- Can we do better?
- Observation: the span s_i on a certain day i can be easily computed if we know the closest day preceding i, such that the price on that day is higher than the price on day i.
- If such a preceding day exists for a day i, let us denote it with $h(i)$, and otherwise define $h(i) = -1$.
- The span on day i is given by $s_i = i - h(i)$
- We can use a stack to store days $i, h(i), h(h(i)),$ etc.
- When going from day i -1 to day i, we repeatedly pop days with prices less than or equal to p_i, and then push day i
- Pseudocode is on Page 175 in your textbook
- Homework exercise: Review the analysis and make sure you understand it.

A better solution

Algorithm computeSpans2(P):

Input: An *n*-element array *P* of numbers

Output: An *n*-element array *S* of numbers such that *S* [*i*] is the largest integer *k* such that *k* <= *i* + 1 and *P* [*j*] <= *P* [*i*] for *j* = *i*-*k*+1, ..., *i*

```
for i = 0 to n – 1 do
        done ✍ false
        while not (D.isEmpty() or done) do
             if P[i] >= P[D.top()] then
                      D.pop()
             else
                      done ✍ true
        if D.isEmpty() then
             h ✍ -1
        else
             h ✍ D.top()
        S[i] ✍ i-h
        D.push(I)
return array S
```