# Due July 20 at midnight to eCampus

First Name: Jonathan Last Name: Westerfield UIN: 224005649

User Name: jgwesterfield E-mail address: jgwesterfield@tamu.edu

Please list all sources in the table below including web pages which you used to solve or implement the current homework. If you fail to cite sources you can get a lower number of points or even zero, read more: Aggie Honor System Office

| Type of sources | Online | | |
|---|---|---|---|
| People | | | |
| Web pages (provide URL) | http://www.geeksforgeeks.org/the-stock-span-problem/ | | |
| Printed material | | | |
| Other Sources | | | |

I certify that I have listed all the sources that I used to develop the solutions/codes to the submitted work.
"On my honor as an Aggie, I have neither given nor received any unauthorized help on this academic work."

Your Name: Jonathan Westerfield Date: 7/18/17

# Programming Assignment 2 (140 points)

1. (30 pts) Implement a stack ADT using STL vector to support the operations: `push()`, `pop()`, `top()`, `empty()`.
2. (10 pts) Test the operations of the stack class for correctness. What is the running time of each operation? Express it using the Big-O asymptotic notation.

   The running time of the push(), pop(), top() and empty() function are all $O(1)$. This is because all of the functions just perform one operation each.

3. Stack Application

   Use the implemented stack class as an auxiliary data structure to compute spans used in the financial analysis, e.g. to get the number of consecutive days when stack was growing.

**Definition of the span:**

Given a vector $X$, the span $S[i]$ of $X[i]$ is the maximum number of consecutive elements $X[j]$ immediately preceding $X[i]$ such that $X[j] \leq X[i]$

Spans have applications to financial analysis, e.g., stock at 52-week high

1. Example of computing the spans S of X.

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

2. (10 pts) Compute the spans based on the definition above:

```
Algorithm spans1(x)
Input: vector x of n integers
Output: vector s of spans of the vector x
for i = 0 to n-1 do
   j = 1
   while (j <= i ∧ x[i-j] <= x[i])
      j =  j + 1
   s[i] = j
return s
```

1. (10 pts) Test the algorithm above for correctness using at least three different input vectors.
2. (10 pts) What is the running time function of the algorithm above? The function should define the relation between the input size and the number of comparisons perform on elements of the vector. How can you classify the algorithms using the Big-O asymptotic notation and why?

```
vector<int> computeVectorSpan(vector<int> &x)
{
    // vector<int> x = {6, 3, 4, 5, 2};
    vector<int> s(x.size());

    int j = 1;
    for(int i = 0; i <= x.size(); i++) // n iterations
    {
        j = 1; // 1 operation: assignment
        while (j <= i && (x[i-j] <= x[i])) // n operations
        {
            j = j + 1; // 2 operations: assignment & addition
        }
        s[i] = j; // 1 operation: assignment
    }

    return s; // 1 operation
}
```

$$F(n) = n * ((2 * 2) + 2) = 2n^2 + 2n + 1 \therefore O(n^2)$$

1. (20 pts) Compute the spans using a stack as an auxiliary data structure storing (some) indexes of x.

   Algorithm spans2:

   (a) We scan the vector x from left to right

   (b) Let i be the current index

   (c) Pop indices from the stack until we find index j such that x[i] $<$ x[j] is true

   (d) Set s[i] $=$ i $-$ j

   (e) Push i onto the stack

2. (10 pts) Test the second algorithm (spans2) for correctness using at least three different input vectors. Your program should run correctly on TA's input.

3. (10 pts) What is the running time function of the second algorithm? The function should define the relation between the input size and the number of comparisons perform on elements of the vector. How can you classify the algorithms using the Big-O asymptotic notation and why? Compare the performance of both the algorithms.

```
template <class T> vector<int> StackADT<T>::computeStackSpan(const vector<T> &x)
{
    StackADT<T> stack;
    // vector<int> x = {6, 3, 4, 5, 2};
    vector<T> s(x.size());

    for(int i = 0 ; i < x.size(); ++i) // n iterations
    {
        while(!stack.empty() && x[stack.top()] <= x[i]) // only runs once per for loop - 1 operation
        {
            stack.pop(); // 1 operation
        }
        if(stack.empty()) // only runs once per for loop
        {
            s[i] = i + 1; // 2 operations: assignment and addition
        }
        else // only runs once per for loop
        {
            s[i] = i - stack.top(); // 2 operations: assignment and subtraction
        }

        stack.push(i); // 1 operation
    }

    return s; // 1 operation
}
```

$$F(n) = n * 1 * 2 + 1 + 1 = 2n + 2 \therefore O(n)$$

The algorithm with the stack implementation is ultimately a better algorithm due to the reduced time complexity. It performs less operations, therefore, it is faster.

1. (10 pts) Programming style, and program organization and design: naming, indentation, whitespace, comments, declaration, variables and constants, expressions and operators, line length, error handling and reporting, files organization. Please refer to the PPP-style document.

2. Instructions

   (a) Your files should be arranged as below

      i. Declaration of `My_stack` class in `My_stack.h`
      ii. Definition (implementation) of `My_stack` class in `My_stack.cpp`
      iii. Algorithm implementations and the `main()` function in the file `Application.cpp`

   (b) Compile your program by

   ```
   g++ -std=c++11 *.cpp
   ```

   or

   ```
   make all
   ```

   (c) Run your program by executing

   ```
   ./Main
   ```

   (d) Testing code in `Application.cpp` and collect output data for your report.

3. Submission

   (a) Tar the files above and any extra files. Please create your tar file following the instructions.

   (b) "turnin" your tar file to eCampus no later than **July 20**.

   (c) (20 points) Submit a hard copy of cover page, report (see below), `My_stack.h`, `My_stack.cpp` and `Application.cpp` in lab to your lab TA. Find a cover page .

      i. Typed report made preferably using "LyX/LaTeX"

         A. (2 pts) Program description; purpose of the assignment

            - This assignment is intended to teach us about the stack data structure, how to implement it, and how it can be applied to specific situations. In this assignment, the stack data structure was used to compute the spans of stock data.

         B. (4 pts) Data structures description

            - Theoretical definition
              - A stack is an abstract data type that is a collection of elements based on 2 functions - Push and Pop. Push adds an element while pop removes the most recently added element.
            - Real implementation
              - My stack function was implemented using a vector. The push function adds an element to the end of the vector while the pop function does the opposite. The empty function returns a boolean if the stack is empty and the top function returns the element at the top of the stack (end of the vector).
            - Analysis of the best and worst scenarios for computing spans.
              - The worst case scenario is if the element in an array of n elements is the largest value in the array and it is also the last value in the array. This means that every value would be parsed in the array in order to compute the span. The best case scenario is if the the element is either at the very beginning of the array or is the smallest value in the array.

      ii. (2 pts) Instructions to compile and run your program; input and output specifications

      ```
      Open terminal
      Navigate to the folder where the main.cpp and StackADT.h files are
      Use the command: g++ -stdc++11 *.cpp -o main
      Run the code with: ./main
      ```

      i. (2 pts) Logical exceptions (and bug descriptions)

      - If the stack is empty but the pop function is called, an exception will be thrown

   (a) (5 pts) C++ object oriented or generic programming features, C++11 features

   - For this assignment, I used templates just in case other data types were desired. Because chars work similarly in terms of sorting from least to greatest, the algorithm would still work even for those.

1. (5 pts) Testing results

```
Destructor has been called
Stack has been removed

Destructor has been called
Stack has been removed

Destructor has been called
Stack has been removed

Span using vector
Vector x: 6 3 4 5 2
Span x:   1 1 2 3 1

Span using stack
Vector x: 6 3 4 5 2
Span x:   1 1 2 3 1

Span using vector
Vector y: 1 5 8 3 7 7 2 9 5 3 5
Span y:   1 2 3 1 2 3 1 8 1 1 3

Span using stack
Vector y: 1 5 8 3 7 7 2 9 5 3 5
Span y:   1 2 3 1 2 3 1 8 1 1 3

Span using vector
Vector z: 1 7 3 9 6 5 9 2 5 3 7 9 1 6 3 9
Span z:   1 2 1 4 1 1 7 1 2 1 4 12 1 2 1 16

Span using stack
Vector z: 1 7 3 9 6 5 9 2 5 3 7 9 1 6 3 9
Span z:   1 2 1 4 1 1 7 1 2 1 4 12 1 2 1 16

Destructor has been called
Stack has been removed


Process finished with exit code 0
```