CSC 228-01 Data Structures and Algorithms, Spring 2018
Instructor: Dr. Natarajan Meghanathan

Project 4: Design and Implementation of an Algorithm to Find the Next Greater Element of the Elements in an Array in Θ(n) Time

Due: February 28th, 1 PM

In this project, you will design and implement an algorithm to determine the next greater element of an element in an array in Θ(n) time, where 'n' is the number of elements in the array. You could use the Stack ADT for this purpose.

The next greater element (NGE) for an element at index i in an array A is the element that occurs at index j (i < j) such that A[i] < A[j] and A[i] ≥ A[k] for all k ∈ [i+1..., j-1]. That is, index j (j > i) is the first index for which A[j] > A[i]. If no such index j exists for an element at index i, then the NGE for the element A[i] is considered to be -1.

For example: if an array is {1, 15, 26, 5, 20, 17, 36, 28}, then the next greater element (NGE) of the elements in the array are as follows:

1	15
15	26
26	36
5	20
20	36
17	36
36	-1
28	-1

Note: Your code need not determine and print the elements and their NGE values in the same order of the appearance of the elements in the array. An out of order print is also acceptable as long as the correct NGE for an element is printed out. For example, the following out of order print for the above array is also acceptable.

1	15
15	26
5	20
17	36
20	36
26	36
28	-1
36	-1

You are given two code files. One file is meant for running the algorithm on a smaller array (of size 10 integers) and printing the output (i.e., the NGE for each element in the array, in the order determined by your algorithm). The other file is meant for running the algorithm on larger arrays (of sizes 5000, 50000 and 500000, with the integers ranging from [1...50000] in each case) and calculating the average run time of your algorithm. The code for the main function is accordingly written (i.e., the main function in the file meant for larger arrays has the timers setup to calculate the run times and there are no print statements and the main function in the file meant for smaller arrays has the print statements, but no timers). You should implement the algorithm in the main function itself and make use of the **Stack stack** object as part of the processing needed by your algorithm.

You are given the doubly linked list-based implementation code for Stack in both the files and there is no need to make any changes in the Stack class.

Submission (through Canvas): Submit everything together as ONE PDF file

(a - 10 pts) Write a pseudo code of the algorithm designed and implemented for this problem.

(b - 15 pts) Discuss the time complexity of the algorithm and justify that it is **Θ(n) with respect to the number of pop operations**.

(c - 40 pts) Include the main functions (that have the implementation of the algorithm) that you come up with for both the smaller arrays and larger arrays. There is no need to submit the doubly-linked list implementation code for the Stack class.

(d - 10 pts) For the smaller array version, show a screenshot of the execution of your algorithm for an array of 10 elements whose maximum value could be 50. For the larger array version, show screenshots that display the average run time (in microseconds, as given in the code) for Array sizes of 5000, 50000, 500000 with a maximum value of 50000 in each case.

(e - 10 pts) Tabulate the values of the average run times observed for the three different array sizes and also determine the ratio of the array size and average run time.

(f - 15 pts) Based on the values for the array size vs. the ratio of the average run time to array size, discuss how the average run time depicts the Θ(n) asymptotic time complexity.