COMP2402B (Fall 2015)

Assignment 3 of 5

Due: November 6th, 2015

Caution

You will be uploading your solutions to cuLearn but this assignment <u>does</u> have several written components.

Only the latest submission received before the deadline will be used to determine your mark on this assignment.

You must adhere to the following rules (as your submission may be subjected to automatic marking system):

- ***Download** the compressed file **"COMP2402-F15-A3-(Start Skeleton).zip"** from cuLearn.*
- ***Retain the directory structure** (i.e., if you find a file in subfolder "comp2402a1", you must not remove it).*
- ***Retain the package directives** (i.e., if you see "package comp2402a1;" in a file, you must not remove it).*
- ***Do not rename any of the methods already present** in the files provided.*
- ***Do not change the visibility of any of the methods provided** (e.g., do not change private to public).*
- ***Do not change the main method of any class**; on occasion the main method provided will use command line arguments and/or open input and output files – you must not change this behaviour).*

*Please also note that **your code may be marked for efficiency as well as correctness** – this is accomplished by placing a hard limit on the amount of time your code wil be permitted for execution. If you select/apply your data structures correctly, your code will easily execute within the time limit, but **if your choice or usage of a data structure is incorrect**, your code may be **judged to be too slow** and **it may receive a grade of zero**.*

*You are expected to **demonstrate good programming practices at all times** (e.g., choosing appropriate variable names, provide comments in your code, etc.) and **your code may be penalized if it is poorly written**.*

Part I - Implementation and Written Questions

1. Design and implement the SEListDeque using the code provided as a starting point. This data structure is a space-efficient linked list into which you can addFirst, addLast, removeFirst, and removeLast in constant amortized time per operation, independent of the block size. (n.b., For this question you are implementing the Deque interface and not a general List; to clarify, you may assume that all data is being added and removed from the ends of the structure.)

   Your implementation of SEListDeque must be accompanied by a written component wherein you describe exactly how your new data structure will work, while also demonstrating the time complexity associated with your functions.

2. The DLListReversible extends DLList (i.e., the doubly-linked list) but is equipped with a reverse function. Write this function and add it to DLListReversible. You are expected to test this function thoroughly and your solution may be marked for efficiency as well as correctness. Do not create a copy of the entire list to solve this problem and do not create a recursive helper method to solve this problem either.

   Your implementation of DLListReversible must be accompanied by a written component wherein you describe exactly how your reverse function works and what its worst-case time complexity will be. An implementation that is not accompanied by an adequate explanation will not receive full marks.

3. Design and implement the MinStack using the code provided as a starting point. This data structure should be able to store comparable elements and, in addition to supporting all the standard stack operations (i.e., push, pop, and peek) in constant time, your solution must also implement the min function (that returns the minimum element of the collection) in constant time as well. You are expected to test this function thoroughly and your solution may be marked for efficiency as well as correctness.

   Hint: When you remove the element from the stack that was considered the minimum element, you must not look through the entire Stack to find the new minimum element.

   Your implementation of MinStack must be accompanied by a written component wherein you describe exactly how your functions work (n.b., all of them) and what underlying structures you needed to include in your MinStack. You must also demonstrate the constant worst-case time complexity of each of the operations.

Part II - Bonus Question

4. For the bonus question you must design and implement a version of a RootishArrayStack that still has only $O(\sqrt{n})$ wasted space, but can also perform both the add(i, x) and remove(i) operations in $\boldsymbol{O(1 + \min\{i, n - i\})}$ time.

   Because this is a bonus problem, you will have need to design and implement your solution entirely independently; **no help will be provided for this problem**.