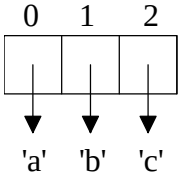
Data Structures (810:052) Lab 4 Name:_____

Objective: To experiment with stack implementations in Python to obverse the importance of big-oh analysis of operations. To gain experience implementing a Stack ADT (Abstract Data Type) using a Python list, Lambert's Array class, and a linked-list implementation.

To start the lab: Download and unzip the file at: www.cs.uni.edu/~fienup/cs052sum09/labs/lab4.zip

Part A: Read chapter 13 and sections 14.1, 14.2, and 14.4 from the Lambert text. Recall that I conjectured in class (after reading on the web) that Python's built-in list is implemented as an array of references (pointers) to list items. For example, the list ['a', 'b', 'c'] would really be implemented as the array.



This has performance ramifications for possible Stack implementations using a list representation. The two possible list representation of a Stack being:

"Abstract" Stack			Files in lab4.zip
	I. Stack with Bottom at list [0]	0 1 2 ['a', 'b', 'c']	(stackClass.py)
c top b a bottom	II. Stack with Top at list [0]	0 1 2 ['c', 'b', 'a']	(stackClassAlt.py)

Complete the expected big-oh notation for each Stack implementation assuming "n" items in the stack.

	push(item)	pop( )
Stack with Bottom at list [0]		
Stack with Top at list [0]		

Use the file `testList.py` to time the pushing of 300,000 items onto a stack and then popping them off. Complete the following timing table:

	Time (seconds) to push 300,000 items	Time (seconds) to pop 300,000 items
Stack with Bottom at list [0]		
Stack with Top at list [0]		

a) Explain why one implementation is so much slower that the other.

b) Why do you suppose it takes less time to pop that push the same number of elements?

Part B: Section 13.2 discussed the Array class for implementing data structures, and section 14.4's ArrayStack class (in file `stack.py`) is an Array implementation of a stack. Starting with the `testList.py` program of part A, make a new `testArray.py` program that times the pushing of 300,000 items followed by the popping of 300,000 items.

a) Study the AWith-respect-to timing, predict which list implementation closest compares to the ArrayStack implementation.

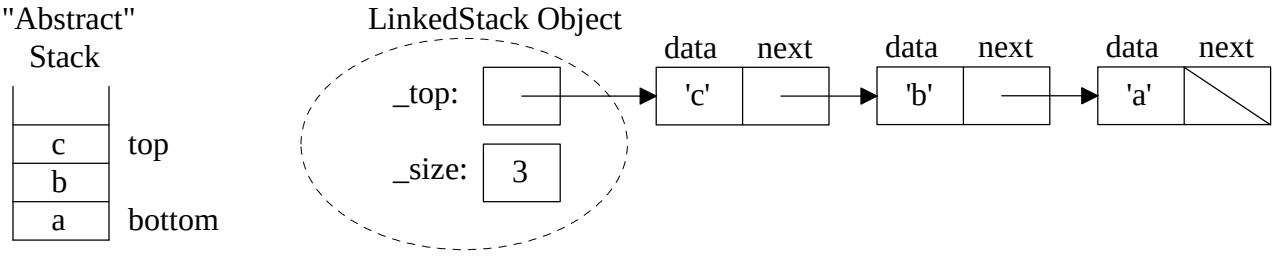
b) What is the timing for the `testArray.py` to push 300,000 items?

c) What is the timing for the `testArray.py` to pop 300,000 items?

d) Why do your `testArray.py` implementation differ from the closed list implementation you predicted in (a)?

e) Modify the pop method of the `ArrayStack` class in the file `stack.py` such that it reduces the array size when the logical size of the array is less than or equal to one-fourth of its physical size and its physical size is greater than the default capacity. In this case, reduce the physical size of the array either to half its physical size or to its default capacity, whichever is greater. (see section 13.3.2 in Lambert).

Part C: The `Node` class defined in node.py is used to dynamically create storage for a new item added to a singly-linked list implementation of the linear data structures like the stack.. The `LinkedStack` class in file `stack.py` uses this Node class to implement a stack ADT. Conceptually, a `LinkedStack` object would look like:



Starting with the `testList.py` program of part A, make a new `testLinkedStack.py` program that times the pushing of 300,000 items followed by the popping of 300,000 items.

a) What is the timing for the `testLinkedStack.py` to push 300,000 items?

b) What is the timing for the `testLinkedStack.py` to pop 300,000 items?

c) Unfortunately, the `Node` class does NOT practice good object-oriented design by allowing the LinkedStack class access to its data and next attributes without going through accessor and mutator methods. I'd like you to rewrite the `Node` class and `LinkedStack` class to fixed this design problem.

d) With your new `Node` and `LinkedStack` classes, retime the pushing and popping of 300,000 items. Explain the difference in timings between these times and those of (a) and (b).