Caleb Ernst | CJE190001 | CS3345.002 | HW#1 – BACKMASKING

- ArrayStack (Normal): In terms of time complexity, the push, peek, pop, & isEmpty methods are O(1), whereas the resize is O(n). By using 2 variables -- one which tracks the "top-most" index, and the other which holds the stack items – we can push successive items to the stack, determine when the stack is about to overflow and resize it, and finally pop out freshly inserted data. The main issue with this implementation is the massive amount of time wasted each time the array needs to resize itself (and copy back over its previously held information).

- ArrayStack (Iterable): This version is the same as ArrayStack, however it utilizes the iterator to avoid using the pointer variable in the original, as a way of slightly improving run time for pops.

- ListStack (Normal): With my implementation, time complexity for push, pop, & isEmpty stands at O(1), because I assigned new nodes backwards, rather than adding new entries to the end of the list (which would mean insertion/deletion of new elements would require you to traverse the whole linked list for a cost of O(n) complexity). This implementation also gets rid of the primary issue found in the ArrayStack: that being the resizing of the data structure, due to the fact that linked lists have no set size. In all this implementation is by far the most efficient of the 2.

- ListStack (Iterable): Same as ListStack, this version instead uses the iterable. To be honest with you, I don't understand what you mean by using the enhanced for-loop for popping, since you only pop 1 item at a time (if I understand the requirements correctly).