CSCI 102, Spring 2024, midterm

Write your name and netID at the top of the page. Throughout you may assume you have access to `ArrayStack<E>` and `ArrayQueue<E>` which are implementations of `Stack<E>` and `Queue<E>`.

1. (4 points) Add a method to `SinglyLinkedList<E>` `void rotate()` that adds the last node to the beginning. Recall `SinglyLinkedList<E>` has attributes `Node<E> head`, `Node<E> tail`, and `int size` and `Node<E>` has get and set methods for its private attribute `Node<E> next`.

(1 point) If n is the size of the list, what is the asymptotic computational complexity of your method? Could you have implemented a faster method for `DoublyLinkedList<E>`? If so, what would its complexity be?

2. (5 points) Write a method that takes a stack and flips the elements in positions `i` and `j`, `void flipStackElements(Stack<E> stack, int i, int j)`. You may assume that `i` $>$ `j` are not equal and smaller than the size of the stack.

3. (2 points) Say we have a sequence $X_0 = 1$, $X_1 = 2$, $X_{=}2$, $X_{n+3} = 1.1X_{n+2}X_{n+1} + X_{n+1}X_n$. Write a **recursive** method `double sequence(int n)` that returns the n-th element of the sequence.

(3 points) Make sure your implementation has asymptotic computational complexity $O(n)$.

4. (5 points) Write an $O(n)$ search strategy for a `BinaryNodeTree<E>` that searches for an `E element` by **checking equality in the deepest nodes first**, `boolean deepSearch(E element)`. Recall the methods of `BinaryNodeTree<E>` include `Position<E> left(Position<E> p)`, `Position<E> right(Position<E> p)`, `Position<E> parent(Position<E> p)`, and it has attributes `Position<E> root` and `int size`. Recall also `Position<E>` has method `E getElement()`.

5. (3 points) A `Set<E>` is a data structure which is a list where **the order of the elements do not matter**, i.e. two sets are equal if they contain the same elements in any order. Assume `Set<E>` implements `GoodList<E>` and therefore has method `E getAtIndex(int index)` and attribute `int size` and recall one can get the hashcode of any object `element` in java with `element.hashCode()`. Write a good hash code for `Set<E>`, overriding the default `int hashCode()`.