CSCI 102, Fall 2024, midterm

Write your name and netID at the top of the page. Throughout you may assume you have access to `LinkedStack<E>` and `LinkedQueue<E>` which are implementations of `Stack<E>` and `Queue<E>`. This midterm is graded out of 13 points. You may use the backs of pages as scratch but please try to keep your answers on the front.

1. (4 points) Write a method for `DoublyLinkedList<E>`, `public void swap(Position<E> pos1, Position<E> pos2)` that swaps the nodes at `pos1` and `pos2`. Recall the method `private Node<E> validate(Position<E> pos)` and recall that the attributes of `Node<E>` are `next, prev, element`. Make sure your code works in all cases, including when the two positions are adjacent! I suggest drawing a picture.

(1 point) Describe what your code does if we pass the method two positions that belong to different lists.

2. (2 points) Implement `static <E> Boolean search(E element, Stack<E> stack)` that returns `true` if `element` is in `stack`. Make sure that after returning, `stack` is as it was when the method was first called (it should have the same elements in the same order). **You may** initialize other data structures in your method.

(2 points) Implement `static <E> Boolean search(E element, Queue<E> queue)`. Make sure that after returning, `queue` is as it was when the method was first called (it should have the same elements in the same order). **Do not** initialize other data structures in your method.

(1 point, no part marks) If n is the size of `stack` and `queue`, what are the best and worst case asymptotic computation complexities for the two methods you've written above?

3. (2 points) Say I give you a `DoublyLinkedList<Integer>` called `list`. Say we have a sequence $X_0 = 0$, $X_1 = 1$, $X_{n+2} = X_{n+1} \times X_n^2 + \texttt{list[n]} \times X_n$ where `list[n]` is the n-th element in list. Write a recursive method to calculate X_m where m should be smaller than the size of `list` and bigger than 0. Hint: the base case is $n = 1$, use the position in the list as an auxillary variable.

(3 points) Make sure your method runs in $O(n)$; in particular, do not use `getAtIndex(int n)`.

4. (1 point) Write a method `PositionList<Position<E>> pathToRoot(Position<E> pos)` that returns a list of the nodes on the path from `pos` to the root.

(1 point) Make it so your method outputs the list with the root in the first position.

(1.5 points) The nearest common ancestor of two positions in a tree is the deepest node that is an ancestor of both positions. Using the method you wrote above, write a method `Position<E> nearestCommonAncestor(Position<E> pos1, Position<E> pos2)` that output the nearest common ancestor of the two nodes. I suggest drawing a picture.

(1.5 points) If n is the number of elements in the tree, $d($`pos`$)$ is the depth of a position `pos`, and $h($`pos`$)$ is the height of `pos`, what is the asymptotic computational complexity of your two methods in big O notation?