CSCI 102, Fall 2025, 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. (2 points) For a list of integers define a "walk" of a list of integers as starting at the first position of a list, then repeatedly moving the number of steps as indicated by the current position until you've left the list. For example, a "walk" on $[2, 3, -1]$ will start at the first position, take 2 steps forward ending up at the third position, then take 1 step back ending up at the second position, and finally take 3 steps forward, exiting the list. Write a method `public int walkLength(PositionList<Integer> list)` that takes a "walk" though `list`, as will be described below, and returns the number of steps it takes before the walk ends.

(1 point) It may be the case that a walk never ends. Give an example of a list for which a walk never ends.

(2 points) Write a method `public boolean hasInfiniteWalk(PositionList<Integer> list)` that returns `true` if the walk would never terminate (i.e., gets stuck in a cycle) and `false` if it eventually exits the list. Your solution should handle this **efficiently**. Hint: your method should not visit more than $O(n)$ positions!

2. (2 points) Write an implementation of Queue<E> (methods E dequeue(), void enqueue(E e), and int size()) using an array of size MAX_SIZE=1000. To do so, keep track of two integers, int start and end that describe the indices in which the elements of the queue are placed in the array. That is, enqueue from the start and dequeue from the end.

(3 points) Imagine repeatedly en-queuing and de-queuing the same element. With a naive implementation, even though the queue always has size 1, eventually start and end will exceed MAX_SIZE causing an out-of-bounds error. Ensure that you queue can hold up to MAX_SIZE elements. To do so, start adding from the beginning of the array again once start or end have exceeded MAX_SIZE using the modulo operator %.

3. (5 points) Call X_n the n-th Fibonacci number, defined by $X_1 = X_2 = 1$ and $X_n = X_{n-1} + X_{n-2}$. Define a new sequence with $Y_1 = Y_2 = 0$ and $Y_n = Y_{n-1} \times X_n + Y_{n-2}$. Write a method `int y(int n)` to calculate the n-th term of the sequence. **Your method must be implemented recursively and efficiently!**

4. (2 points) Write a method `public int maximumValue(LinkedBinaryTree<Integer> tree)` that returns the maximum value in the tree. **Your method must be implemented recursively and efficiently!**

(3 point) Write a method `public int sumMaximumValue(LinkedBinaryTree<Integer> tree)` that returns the sum of the maximum values of the subtree rooted at each node in the tree – that is, imagine applying `maximumValue` to the subtree rooted at each node then summing up the values across the whole tree. **Your method must be implemented recursively and efficiently!** Hint: use an auxilliary variable.