CSSE 230 – Data Structures and Algorithm Analysis

Exam 1

Spring term, 2022-2023

Your name: _____

Instructions:

- All the work you turn in must be your own.
- You must not use any forms of communication or cooperation.
- This is an open book exam:
  - For any portion, you may look at the book whether hardcopy or online.
  - You may look at our CSSE 230 web-site and any page linked directly from it.
  - You may not search the web and you may not use ChatGPT or like technologies during the exam.
  - You may look at any of the Java documentation, in particular that for data structures.
  - For the paper part, you may NOT run any applets nor any code.
  - For the on-the-computer part, you may NOT copy any of the code on your computer.

Problem	Points	Your score
1	**10**	
2	**16**	
3	**14**	
4	**12**	
5	**8**	
containsAll()	**15**	
insert()	**25**	
Total	**100**	

1) Consider the problem of removing a specified element from a singly-linked list with only a head pointer.

   a) [2 pts] Describe when the **best**-case occurs and its runtime. Express the runtime in terms of the number of nodes visited.

   b) [4 pts] Describe when the **worst**-case occurs and its runtime. Express the runtime in terms of the number of nodes visited.

   c) [4 pts] Would the **worst**-case runtime improve, if the list additionally had a tail pointer?

2) In class, we discussed how a `LinkedList` is also a `Stack` and a `Queue`. Suppose the programming language you use has just one data structure: a `Queue`. Explain how you could use that queue to implement a stack. Assume that the `Queue` data structure has the following methods:

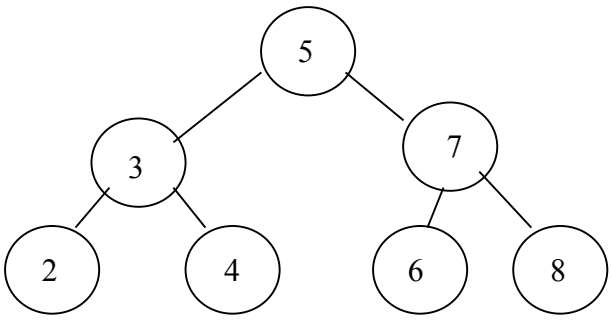
   - *isEmpty*(): Tells whether the queue is empty
   - *enqueuer*(*T element*): Adds the specified element to the end of the queue
   - *dequeue*(): Removes and returns the first element of the queue
   - *peek*(): Return, but does not return the first element of the queue

   a) [2 pts] Explain how you would implement the Stack operation *push*(*T element*), which places the specified element on the top of the Stack.

   b) [14 pts] Explain how you would implement the Stack operation *pop*(), which returns and removes the element on the top of the Stack. Please provide a detailed explanation or provide pseudo-code.

3) Consider the problem of removing elements from a BST. In general, removing all n elements from a BST that is balanced requires O(n*log(n)) time.

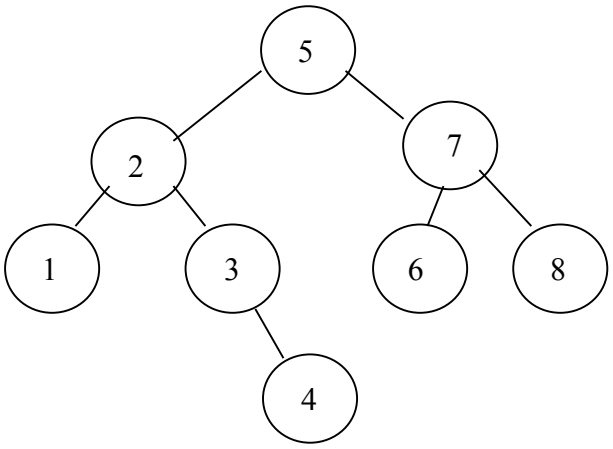
a) [6 points] Is there a clever removal sequence that would reduce the amount of work being done to the extent that we end up in a better complexity class, i.e. O(n). If so, present and justify it. If not, explain why not. And no, simply setting the root to null does not count! Below is a BST tree that you could use to explain your reasoning.



b) [8 points] Is there a BST for which there is an O(n) removal sequence? If so, draw a representative tree, list the removal sequence and briefly explain why this case runs in O(n). If not, explain why not.

4) We have used lazy iterators in BSTs. However, I have not enforced lazy iterators for AVL trees. Consider a **lazy in-order** iterator and focus on the *remove*() method.

a) [8 points] Consider the following AVL Tree. Suppose we call *next*() for the first time and immediately afterwards, we call the iterator's *remove*() method. Explain what will go wrong soon afterwards. Feel free to show the contents of the stack used by the iterator.



b) [4 points] Briefly explain why it is hard to fix the problem.

5) [8 pts] Consider the following procedure which multiplies two matrices, the one passed in the parameter `other` with the matrix to which this method is applied. Currently the resulting matrix of the multiplication is written to a global variable `result` and the success of the operation is written to a global variable `success`. Sensibly modify this procedure so to eliminate the need for the two global variables, by communicating back the `result` matrix as well as the success `boolean` to the calling procedure.

```
public void multiply(Matrix other) {
     if (this.cols != other.getRows() ||
         result.getRows() != this.rows ||
         result.getCols() != other.getCols()) {
         success = false;
     } else {
       for (int i = 0; i < this.rows; i++) {
           for (int j = 0; j < other.getCols(); j++) {
               double sum = 0;
               for (int k = 0; k < this.cols; k++) {
                   sum += this.getValue(i, k) *
                          other.getValue(k, j);
               }
               result.setValue(i, j, sum);
           }
       }
       success = true;
     }
 }
```

On-the-Computer part

- Download the **`Exam1`** project located on our Moodle site located in the *Exam 1* section. Before you do that, please submit the written portion to the specified Moodle drop box.
- All code you write should be **correct, efficient,** and use **good style.** However, no documentation is required, because of time constraints.
- All code you submit **MUST** be your own work.
- Please use the provided unit test-cases, as well as your own test-cases to ensure all functional and performance requirements are met.
- The points given by the unit test-cases are assuming an honest effort. Writing code that satisfies just the test cases will result in zero points given. Partial credit will be given where appropriate.
- Implement the [15 pts] *containsAll()* and [25 pts] *insert()* methods as specified in the Java Docs of the **`Exam1`** starter code.
- Submit your **BinarySearchTree.java** file to the **Exam1-Code** drop box on our Moodle course site.
- Recall that you need to add JUNIT 4 to the build path. In the Testing.java file, hover the cursor over the "org.junit…" import, then click on "Fix project setup" in the window that pops up and then "Add JUnit 4 to build path."

```
/**
 * This method checks whether all elements of the Collection c
 * are contained in this MyLinkedList. This method runs in
 * O(c*N) time.
 * @param c The Collection to be checked.
 * @return true, is all elements in c are contained in this
 * MyLinkedList and false otherwise.
 */
public boolean containsAll(Collection<? extends T> c) {
```

```
/**
 * This method is used to insert into an already sorted list
 * to maintain a sorted list. The list may contain duplicates.
 * This method runs in O(N) time.
 * @param e The element to be inserted.
 * @return Returns true, if the element was successfully
 * inserted and false otherwise.
 */
public boolean insert(T e) {
```