CSSE 230 – Fundamentals of Computing

Final Exam

Spring term, 2022-2023

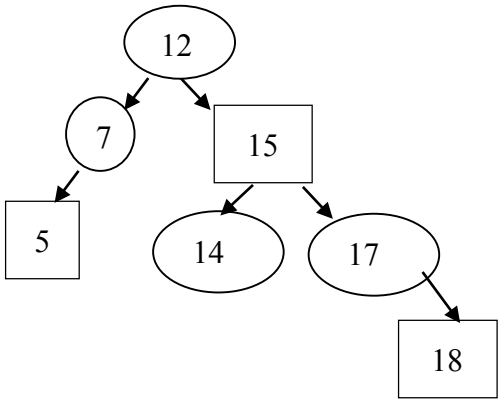
Your name: _____

Instructions:

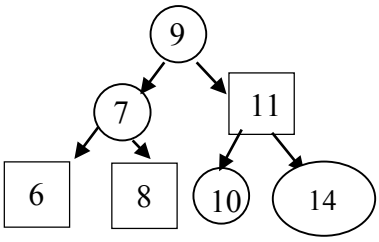
- All the work you turn in must be your own.
- You must not use any forms of communication or cooperation.
- This is an open book exam:
    - For any portion, you may look at the book whether hardcopy or online.
    - You may look at our CSSE 230 or Moodle web-site and any page linked directly from it.
    - 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.
- Submit the written portion to the **FinalExam** drop box on Moodle.

Problem	Points	Your score
1	**8**	
2	**8**	
3	**12**	
4	**10**	
5	**8**	
`isBinaryGraph()`	**10**	
`addGraph()`	**20**	
`toString()`	**24**	
Total	**100**	

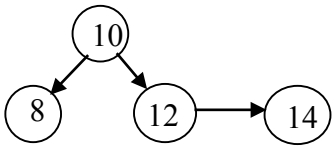
1) a) [4 points] Insert the following elements: 6, 8, 9 in the order listed, into the ***Red-Black*** tree below. Use top-down insertion and show intermediate trees in case of rotations. Indicate which nodes are red and which ones are black. In the tree below, red nodes are drawn as a square.



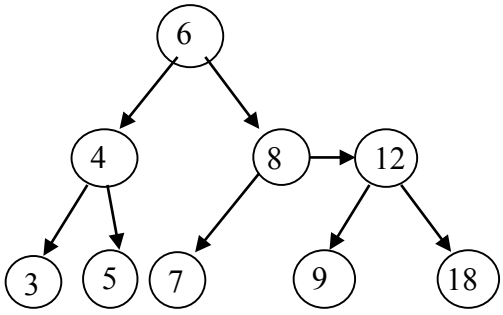
b) [4 points] Remove the following element: 7 from the ***Red-Black*** tree below. Use top-down removal and show intermediate trees in case of rotations. Indicate which nodes are red and which ones are black. In the tree below, red nodes are drawn as a square.



2) a) [4 points] Insert the following elements: 13, 7 in the order listed, into the **AA** tree below. Either indicate the level of each node by writing down its level next to it or draw nodes that are on the same level using horizontal links. Show intermediate trees in case of rotations.



b) [4 points] Remove node 5 from the ***AA*** tree below. Either indicate the level of each node by writing down its level next to it or draw nodes that are on the same level using horizontal links. Show intermediate trees in case of rotations



3) Analyze the following method. Assume that the *getBucket()* method returns the *ith* bucket. Count the number of element to element comparisons.

```
public static ArrayList<Integer> hashTableSort(ArrayList<Integer> a){
      HashSet<Integer> ht = new HashSet<Integer>(a.size());
      for (int i = 0; i < a.size(); i++){
            ht.add(a.get(i));
      }
      for (int i = 0; i < ht.size(); i++) mergeSort(ht.getBucket(i));
      ArrayList<Integer> temp = new ArrayList<Integer>();
      for (int i = 0; i < ht.size(); i++) temp.addAll(ht.getBucket(i));
      return temp;

}
```

a. [4 points] Determine the **worst case** runtime of the method *hashTableSort()*.

What is the big-Oh runtime of this algorithm: _____

Describe a case when the worst case occurs:

b. [4 points] Determine the **best case** runtime of *hashTableSort()*.

What is the big-Oh runtime of this algorithm: _____

Describe a case when the best case occurs:

c. [4 points] Does the method return a sorted ArrayList? Justify your answer.

4) In class, we saw the benefits of adding a HashMap to a data structure: it offers O(1) performance for finding an element. Below is an excerpt of the data we discussed. The 2nd data row contains the data for finding an element using a HashMap.

Category	BST	AVL	TDRB	AA	Skiplist
Finding 2 Mio in 2 Mio	1137	1505	1123	1160	3125
Finding 2 Mio in 2 Mio w/ HashTable	215	223	227	230	260
Iterator[2)]	758	848	692	654	278
Inserting 1 Mio Into 1 Mio	937	1226	1026	1229	1926
Removing 1 Mio in 2 Mio	1024	1234	1371	1748	2023

Another performance test we conducted is the speed it takes to generate an in-order iterator and reading off the first 5-100 pieces of data. Here Skiplists shine. As a matter of fact, if we look at the Skiplist data for HashMap find and for Iterators in-order retrieval, Skiplist top all other data structures. However, Skiplists are very slow on insert and removal. Suppose we combine a Skiplist with a Top-down Red-Black tree.

a) [2 points] Determine the **average** case big-Oh run time of inserting an element into this composite data structure. Justify your answer.

b) [2 points] Determine the **average** case run time, based on the table from above. Justify your answer.

c) [6 points] Propose a modification of Top-down Red-Black trees that maintains its current performance, except, it matches the performance of Skiplists when it comes to producing the first 5-100 pieces of data through the in-order iterator.

5) We justified B-trees for applications that have a lot of data, such as data bases.
   a) [2 points] How do very large amounts of data affect the runtime performance of algorithms?

   b) [2 points] What is an essential property of B-trees that addresses the runtime performance issue from (a).

   c) [4 points] Where else in this course have we seen the effect that very large amounts of data have on the performance of algorithms. Justify your response.

On-the-Computer part

- Download the `FinalExam.zip` project from our Moodle site.
- All code you write should be **correct, efficient, and use good style.** However, no documentation is required.
- All code you submit **MUST** be your own work and you must use the given starter code.
- 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. For example, submitting code that is designed to merely satisfy the test cases will result in zero points given. Sometimes, for example in case of testing Boolean functions, the score given by the test-cases may not represent the true functionality of the code and in that case, may be adjusted up or down. Partial credit will be given where appropriate.
- Implement the procedures documented below and in the unit test cases.
- Please submit your **Graph.java** file to the **FinalExam** 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 select "Add JUnit 4 to build path."

```
/**
 * This method determines whether a graph is a binary graph.
 * By this we mean that every node has either zero, one or
 * two children.
 * @return true if every node has either 0, 1, or 2 children
 * and false otherwise
 */
public boolean isBinaryGraph() {

/**
 * This method adds the graph g to this graph. It will only
 * add additional nodes and edges that are not already
 * contained in this graph.
 * @param g Graph to be added
 * @return This method returns true if all nodes and all edges
 * of g were added. It returns false otherwise.
 */
public boolean addGraph(Graph<T> g) {

/**
* This method returns a string of the node keys, in increasing
* distance from the node passed to it. Distance is measured by
* number of edges. We may call this a level-order traversal of
* the graph. You may assume that the graph is connected. If two
* nodes have the same distance, then preserve the ordering as
* found in the list of neighbors. Each node key should appear
* only once, in other words, please avoid cycles.
* @param start The node at which the traversal begins
* @return A list of the node keys, enclosed in square brackets
* and separated by ", ". If the graph is empty, the method returns
* just "[]".
*/
public String toString(T start) {
```