CSC 151 Assignment 9

Objectives:

- Implementing solutions that uses binary trees
- Implementing tree traversal methods by recursion
- Heapyfing binary trees

Definition

You will create binary trees from a given ArrayList of any type. The tree will be a complete tree so you add the nodes to the tree in the given order. The previous levels should be completed before adding a new node to the next level. The nodes should be added from left to right. The i^{th} index in the ArrayList is the parent and 2*i +1 and 2*i +2 are the left and right children. `insertLevelOrder` method inserts the items in the ArrayList in a binary tree by applying this idea and returns the root.

Another option will be creating the tree as a max heap. In a max heap the root data is greater than its left and right child and this is true for all subtrees. Therefore, the root is the maximum of all Nodes. Node data should be implemented as a comparable data type.

Using **recursive** methods will be easier to implement**.**

For example:

- Given the following integers in the list

```
[6, 0, 1, 3, 6, 5, 4, 7, 9, 2, 12, 15, 28, 32, 48]
```

  - The root of the tree will be 6 and its left child is 0 and right child is 1.
  - In the left subtree root is 0 and its left child is 3 and right child is 6.
  - In the right subtree root is 1 and its left child is 5 and right child is 4.

- Given the following integers in the list and we want to **heapify** the list first,

```
[6, 0, 1, 3, 6, 5, 4, 7, 9, 2, 12, 15, 28, 32, 48]
```

  - we will start from the last item and compare it with its parent.
  - If the child is greater than the parent swap it.
  - Repeat this process for all items from last to first.
  - When you reach the first and you swapped at least one item start over from the last item and repeat this again.
  - The list will have the followings after this process and insert the items in this order:

```
[48, 12, 32, 9, 6, 28, 6, 7, 3, 2, 0, 15, 5, 4, 1]
```

  - The root of the tree will be 48 and its left child is 12 and right child is 32.
  - In the left subtree root is 12 and its left child is 9 and right child is 6.
  - In the right subtree root is 32 and its left child is 28 and right child is 6.
  - See the demonstration how this process works. i is used for the current index.

```
i:14 parenti:7
[6, 0, 1, 3, 6, 5, 48, 7, 9, 2, 12, 15, 28, 32, 4]
i:13 parenti:6
[6, 0, 1, 3, 6, 5, 48, 7, 9, 2, 12, 15, 28, 32, 4]
i:12 parenti:6
[6, 0, 1, 3, 6, 28, 48, 7, 9, 2, 12, 15, 5, 32, 4]
i:11 parenti:5
[6, 0, 1, 3, 6, 28, 48, 7, 9, 2, 12, 15, 5, 32, 4]
i:10 parenti:5
[6, 0, 1, 3, 12, 28, 48, 7, 9, 2, 6, 15, 5, 32, 4]
i:9 parenti:4
[6, 0, 1, 3, 12, 28, 48, 7, 9, 2, 6, 15, 5, 32, 4]
i:8 parenti:4
[6, 0, 1, 9, 12, 28, 48, 7, 3, 2, 6, 15, 5, 32, 4]
i:7 parenti:3
[6, 0, 1, 9, 12, 28, 48, 7, 3, 2, 6, 15, 5, 32, 4]
i:6 parenti:3
[6, 0, 48, 9, 12, 28, 1, 7, 3, 2, 6, 15, 5, 32, 4]
i:5 parenti:2
[6, 0, 48, 9, 12, 28, 1, 7, 3, 2, 6, 15, 5, 32, 4]
i:4 parenti:2
[6, 12, 48, 9, 0, 28, 1, 7, 3, 2, 6, 15, 5, 32, 4]
i:3 parenti:1
[6, 12, 48, 9, 0, 28, 1, 7, 3, 2, 6, 15, 5, 32, 4]
i:2 parenti:1
[48, 12, 6, 9, 0, 28, 1, 7, 3, 2, 6, 15, 5, 32, 4]
i:1 parenti:0
[48, 12, 6, 9, 0, 28, 1, 7, 3, 2, 6, 15, 5, 32, 4]


Swap: Reheapify
i:14 parenti:7
[48, 12, 6, 9, 0, 28, 4, 7, 3, 2, 6, 15, 5, 32, 1]
i:13 parenti:6
[48, 12, 6, 9, 0, 28, 32, 7, 3, 2, 6, 15, 5, 4, 1]
i:12 parenti:6
[48, 12, 6, 9, 0, 28, 32, 7, 3, 2, 6, 15, 5, 4, 1]
i:11 parenti:5
[48, 12, 6, 9, 0, 28, 32, 7, 3, 2, 6, 15, 5, 4, 1]
i:10 parenti:5
[48, 12, 6, 9, 6, 28, 32, 7, 3, 2, 0, 15, 5, 4, 1]
i:9 parenti:4
[48, 12, 6, 9, 6, 28, 32, 7, 3, 2, 0, 15, 5, 4, 1]
i:8 parenti:4
[48, 12, 6, 9, 6, 28, 32, 7, 3, 2, 0, 15, 5, 4, 1]
i:7 parenti:3
[48, 12, 6, 9, 6, 28, 32, 7, 3, 2, 0, 15, 5, 4, 1]
i:6 parenti:3
[48, 12, 32, 9, 6, 28, 6, 7, 3, 2, 0, 15, 5, 4, 1]
i:5 parenti:2
[48, 12, 32, 9, 6, 28, 6, 7, 3, 2, 0, 15, 5, 4, 1]
i:4 parenti:2
[48, 12, 32, 9, 6, 28, 6, 7, 3, 2, 0, 15, 5, 4, 1]
i:3 parenti:1
[48, 12, 32, 9, 6, 28, 6, 7, 3, 2, 0, 15, 5, 4, 1]
i:2 parenti:1
[48, 12, 32, 9, 6, 28, 6, 7, 3, 2, 0, 15, 5, 4, 1]
i:1 parenti:0
[48, 12, 32, 9, 6, 28, 6, 7, 3, 2, 0, 15, 5, 4, 1]


Swap: Reheapify
i:14 parenti:7
[48, 12, 32, 9, 6, 28, 6, 7, 3, 2, 0, 15, 5, 4, 1]
i:13 parenti:6
[48, 12, 32, 9, 6, 28, 6, 7, 3, 2, 0, 15, 5, 4, 1]
i:12 parenti:6
[48, 12, 32, 9, 6, 28, 6, 7, 3, 2, 0, 15, 5, 4, 1]
i:11 parenti:5
[48, 12, 32, 9, 6, 28, 6, 7, 3, 2, 0, 15, 5, 4, 1]
i:10 parenti:5
[48, 12, 32, 9, 6, 28, 6, 7, 3, 2, 0, 15, 5, 4, 1]
i:9 parenti:4
[48, 12, 32, 9, 6, 28, 6, 7, 3, 2, 0, 15, 5, 4, 1]
i:8 parenti:4
[48, 12, 32, 9, 6, 28, 6, 7, 3, 2, 0, 15, 5, 4, 1]
i:7 parenti:3
```

```
[48, 12, 32, 9, 6, 28, 6, 7, 3, 2, 0, 15, 5, 4, 1]
i:6 parenti:3
[48, 12, 32, 9, 6, 28, 6, 7, 3, 2, 0, 15, 5, 4, 1]
i:5 parenti:2
[48, 12, 32, 9, 6, 28, 6, 7, 3, 2, 0, 15, 5, 4, 1]
i:4 parenti:2
[48, 12, 32, 9, 6, 28, 6, 7, 3, 2, 0, 15, 5, 4, 1]
i:3 parenti:1
[48, 12, 32, 9, 6, 28, 6, 7, 3, 2, 0, 15, 5, 4, 1]
i:2 parenti:1
[48, 12, 32, 9, 6, 28, 6, 7, 3, 2, 0, 15, 5, 4, 1]
i:1 parenti:0
[48, 12, 32, 9, 6, 28, 6, 7, 3, 2, 0, 15, 5, 4, 1]
```

- The trees with and without the heap will be displayed as follows:

Display as a tree with heap	Display as a tree without heap
48 |_12 |_|_9 |_|_|_7 |_|_|_3 |_|_6 |_|_|_2 |_|_|_0 |_32 |_|_28 |_|_|_15 |_|_|_5 |_|_6 |_|_|_4 |_|_|_1	6 |_0 |_|_3 |_|_|_7 |_|_|_9 |_|_6 |_|_|_2 |_|_|_12 |_1 |_|_5 |_|_|_15 |_|_|_28 |_|_4 |_|_|_32 |_|_|_48

Class TreeGenericArrayList

- Use the partial code provided at the end of the document.
- It will be in **package assignment** (it is required otherwise **Gradescope** cannot run the tests)
- See the details below written as comments and complete the partial code as instructed.
- Test your class in the main method.
- Test your class by Junit

General Submission Instructions

- You may have groups for the assignments **if it is allowed.** If you have a group state it **explicitly in the honor code** by writing all group members' names. Please name the pdf files accordingly. **However, all group members should submit individually.**
- All submissions require a package called **assignment** and your source codes (.java files) will be under it.
- You should submit
    - Your java files to **Gradescope** as required and
    - Your java files and sample outputs for various scenarios as a **single pdf** file named as **YourFullName(s)_AssignmentN.pdf** with the **honor code** at the top as a comment to Nexus under the appropriate link. Replace N in AssignmentN with the appropriate value. (e.g., if I submit Assignment1 I will submit a pdf file whose name is **ZeynepOrhan_Assignment1.pdf.** If I have a group whose members are me and Jane Doe, then the file name will be **JaneDoeZeynepOrhanAssignment1.pdf** and the honor code will be modified). Use the template .docx file attached and modify it accordingly.
- **If you use any resource to get help, please state it in the honor code. Otherwise, it will be an honor code violation!!!**
- **Gradescope provides a feature to check code similarity by comparing all submissions. If the percentage of your code's similarity is above a threshold, then this will be an honor code violation!!!**
- Make sure that you have no compiler errors.
- When you have compiler error free codes and submit appropriately to Gradescope, your code will be tested automatically and graded. You do not need to know the test cases, but you will see the test results. Informative messages will be provided for success and failure cases.
- You can resubmit your files any number of times if you have failing tests until the due date.
- Your final submission or the one you selected will be graded after the due date of the assignment.
- Please start as early as possible so that you may have time to come and ask if you have any problems.
- Read the late submission policy in the syllabus.
- Please be aware that the correctness, obeying the OOP design and submission rules are equally important
- Gradescope testing will generally be 100% and sometimes manual grading will be done. Please be aware that **your grade can be reduced if you do not follow the instructions and apply good programming practices**.
- Use the starter code if provided.
- Do not use any predefined Collection classes of Java unless stated otherwise.

Assignment specific instructions

- General submission instructions apply.
- **Group work: NOT ALLOWED!!!**
- **Due date:** March 6, 2022, until 11:50 PM
- **Nexus:** Submit the code file (**TreeGenericArrayList.java**) as a single pdf file named as **YourFullName_Assignment9.pdf** with the honor code at the top as a comment under Assignment 8 link.
- **Gradescope:** Submit the java files (**TreeGenericArrayList.java**) under the Assignment 9
- **Grading:**
    - Gradescope: Yes
    - Manual: No
- **Starter code:** Use the code given below

```java
/*
 * Honor code
 */
package assignment;

import java.util.*;

/**
 * This class generates a binary complete tree from an ArrayList. The tree can
 * be constructed by preserving the heap property if selected. The tree can be
 * traversed by inOrder, preOrder or postOrder methods The tree can be formatted
 * as indented.
 *
 * @author Zeynep Orhan
 *
 * @param <T> type of the data to be stored in the tree nodes. <T> should have a
 *            compareTo method
 */
public class TreeGenericArrayList<T extends Comparable<? super T>> {

	private Node root; // Root Node

	//  Private inner class Node for a binary tree
	private class Node {
		private T data;
		private Node left, right;

		private Node(T data) {
			this.data = data;
			this.left = null;
			this.right = null;
		}
	}
	// getRoot, setRoot, getRootData and setRootData methods



	/**
	 * heapifyArray: This method reorders the items in an ArrayList arr to preserve the heap property
	 *
	 * @param arr ArrayList<T> that keeps the items to be stored
	 */




	/**
	 * insertLevelOrderHeap: Inserts the items in the arr and creates a tree whose root is root
	 * If heap is true call heapifyArray first
	 *
	 * @param arr ArrayList<T> that keeps the items to be stored
	 * @param root root of the tree of type Node
	 * @param i the int index of the ArrayList item to be inserted
	 * @param heap boolean the ArrayList will be heapified if true, otherwise it will be inserted as is
	 * @return root of the tree
	 */




	/**
	 * insertLevelOrder: Inserts the items in the arr and creates a tree whose root is root
	 *
	 * @param arr ArrayList<T> that keeps the items to be stored
	 * @param root root of the tree of type Node
	 * @param i the int index of the ArrayList item to be inserted
	 * @return root of the tree
	 */
```

```
/**
 * inOrder: String representation of the inorder traversal
 *
 * @param root root of the tree
 * @return a String
 */




/**
 * preOrder: String representation of the preorder traversal
 *
 * @param root root of the tree
 * @return a String
 */




/**
 * postOrder: String representation of the postorder traversal
 *
 * @param root root of the tree
 * @return a String
 */




/**
 * height: Calculates the height of the tree
 * @param root of the tree
 * @return height of the tree as int
 */




/**
 * display: Display the tree in a formatted way (vertical)
 * If the items are { 6, 0, 1, 3, 6, 5, 4, 7, 9, 2, 12, 15, 28, 32, 48 }
 * The tree will be displayed as
 *      6
 *      |_0
 *      |_|_3
 *      |_|_|_7
 *      |_|_|_9
 *      |_|_6
 *      |_|_|_2
 *      |_|_|_12
 *      |_1
 *      |_|_5
 *      |_|_|_15
 *      |_|_|_28
 *      |_|_4
 *      |_|_|_32
 *      |_|_|_48
 *
 * @param root root of the tree
 * @return
 */
```

```
        public static void main(String args[]) {
                TreeGenericArrayList<Integer> t1 = new TreeGenericArrayList<>();
                Integer arr[] = { 6, 0, 1, 3, 6, 5, 4, 7, 9, 2, 12, 15, 28, 32, 48 };
                ArrayList<Integer> arr1 = new ArrayList<>();
                Collections.addAll(arr1, arr);
                t1.setRoot(t1.insertLevelOrderHeap(arr1, t1.getRoot(), 0, true));

                System.out.println();
                System.out.println("\ninorder with heap");
                System.out.println(t1.inOrder(t1.root));

                System.out.println();
                System.out.println("\npreorder with heap");
                System.out.println(t1.preOrder(t1.root));

                System.out.println();
                System.out.println("\npostorder with heap");
                System.out.println(t1.postOrder(t1.root));

                System.out.println();
                System.out.println("\nDisplay as a tree with heap");
                System.out.println(t1.display(t1.root));

                TreeGenericArrayList<Integer> t2 = new TreeGenericArrayList<>();
                ArrayList<Integer> arr2 = new ArrayList<>();
                Collections.addAll(arr2, arr);
                t2.setRoot(t2.insertLevelOrderHeap(arr2, t2.getRoot(), 0, false));

                System.out.println();
                System.out.println("\nIn order without heap");
                System.out.println(t2.inOrder(t2.root));

                System.out.println();
                System.out.println("\npreorder without heap");
                System.out.println(t2.preOrder(t2.root));

                System.out.println();
                System.out.println("\npostorder without heap");
                System.out.println(t2.postOrder(t2.root));

                System.out.println();
                System.out.println("\nDisplay as a tree without heap");
                System.out.println(t2.display(t2.root));

        }
}
```

Output

```
inorder with heap
7 9 3 12 2 6 0 48 15 28 5 32 4 6 1


preorder with heap
48 12 9 7 3 6 2 0 32 28 15 5 6 4 1


postorder with heap
7 3 9 2 0 6 12 15 5 28 4 1 6 32 48


Display as a tree with heap
48
|_12
|_|_9
|_|_|_7
|_|_|_3
|_|_6
|_|_|_2
|_|_|_0
|_32
|_|_28
|_|_|_15
|_|_|_5
|_|_6
|_|_|_4
|_|_|_1
```

```
In order without heap
7 3 9 0 2 6 12 6 15 5 28 1 32 4 48


preorder without heap
6 0 3 7 9 6 2 12 1 5 15 28 4 32 48


postorder without heap
7 9 3 2 12 6 0 15 28 5 32 48 4 1 6


Display as a tree without heap
6
|_0
|_|_3
|_|_|_7
|_|_|_9
|_|_6
|_|_|_2
|_|_|_12
|_1
|_|_5
|_|_|_15
|_|_|_28
|_|_4
|_|_|_32
|_|_|_48
```