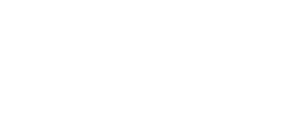


Nov 10

Assignment 8

Comp 2230\_02

Colton isles and kaylee crocker

2024

A close up of a logo

Description automatically generated

**COMP 2230 – Data Structures and Algorithm Analysis**

Assignment #8: Binary Search Trees and Heaps

## Due Date: Section 01 Nov. 14th Section 02 Nov 15th, 2024

**Chapter 20**

Programming problem:

Develop an array implementation of a binary search tree using the computational strategy to locate the children of a node. (2\* n +1) for left child and 2 \* (n + 1) for right child. Note the binary search tree of integers will not be a true binary search tree as define in chapter 20 of the text book. The binaryArrayTree will support the following operations:

1. Default constructor
2. toString in level order
3. Insert(int item)
4. toString2 in preorder

**Assignment Submission:**

Submit a print-out of the program source code and a sample of the output, for each problem. Note you must follow the marking guidelines as identified in the LabMark document.

**Problem #1 Code**

|  |
| --- |
| **BinaryTreeArray.java**  package Ass8\_2230;  import java.lang.reflect.Array;  import java.util.Arrays;  public class BinaryArrayTree<T extends Comparable<T>> {  private T[] tree;  private static final int DEFAULT\_SIZE = 10;  @SuppressWarnings("unchecked")  BinaryArrayTree() {  tree = (T[]) new Comparable[DEFAULT\_SIZE];  }  public void insert(T element) {  int position = findElementPos(element, 0);  if(position >= tree.length) {  expandCapacity();  }  tree[position] = element;  }  private int findElementPos(T element, int node) {  int result;  if (node >= tree.length || tree[node] == null) {  result = node;  } else if (element.compareTo(tree[node]) < 0) {  result = findElementPos(element, 2 \* node + 1);  } else {  result = findElementPos(element, 2 \* node + 2);  }  return result;  }  private void expandCapacity() {  tree = Arrays.copyOf(tree, tree.length \* 2);  }  public String toString() {  return Arrays.toString(tree);  }  public String toStringPreOrder() {  String result = preOrder(0, "");  return result.substring(0, result.length() - 2);  }  private String preOrder(int node, String s) {  String result = s;  if(node < tree.length && tree[node] != null) {  //visit root  result += tree[node] + ", ";  //traverse left side  result = preOrder(2 \* node + 1, result);  //traverse right side  result = preOrder(2 \* node + 2, result);  }  return result;  }  } |

|  |
| --- |
| **BinaryArrayTreeTest.java**  package Ass8\_2230;  public class BinaryArrayTreeTest {  public static void main(String[] args) {  BinaryArrayTree<Integer> bst = new BinaryArrayTree<Integer>();  System.out.println(bst);  bst.insert(15);  System.out.println(bst);  bst.insert(17);  System.out.println(bst);  bst.insert(18);  System.out.println(bst);  bst.insert(19);  System.out.println(bst);  bst.insert(10);  System.out.println(bst);  bst.insert(7);  System.out.println(bst);  bst.insert(9);  System.out.println(bst);  bst.insert(2);  System.out.println(bst);  bst.insert(16);  System.out.println(bst);  System.out.println("Tree in Preorder: " + bst.toStringPreOrder());  }  } |

**Problem #1 Test Output**

|  |
| --- |
|  |

**Chapter 21**

A drawing of a diagram

Description automatically generated1 Draw the min heap that results from adding the following integers ( 34 45 3 87 65 32 1 12 17)

2 Starting with the previous heap draw the heap that results from performing a removeMin operation.

A drawing of a diagram

Description automatically generated

3 Starting with an empty minheap, draw the heap after each of the following operations

addElement(40)

addElement(25)

RemoveMin

AddElement(10)

RemoveMin()

AddElement(5)

AddElement(1)

RemoveMin()

AddElement(45)

AddElement(50)

4 Repeat 3, this time with a maxheap

addElement(40)

addElement(25)

RemoveMax

AddElement(10)

RemoveMax()

AddElement(5)

AddElement(1)

RemoveMax()

AddElement(45)

AddElement(50)