Vanna Moore

CMPS 390 – Program 6 Tree Vs Bubble

This program reads the file of integers, and stores them in an array pre-sorted. A method called buildSort sorts puts them in order in a tree. A method called inOrder prints them out from low to high. A method called compareTotal counts the number of comparisons.

Total number of tree comparisons: 13458

Total number of bubble sort comparisons are: 953046

The tree sort program is more efficient

Based on comparisons made, the tree sort is more efficient

Based on memory used, the bubble sort is more efficient.

A screenshot of a computer program

Description automatically generated A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

import java.util.Scanner;  
import java.io.File;  
import java.io.FileNotFoundException;  
import java.util.Random;  
**public class Main** {  
 public static void main(String[] args) throws FileNotFoundException {  
 File file = new File("\\Users\\vanna\\OneDrive\\Desktop\\CMPS 390\\CMPS 390 Assignments\\Program6Tree\_vs\_Bubble\\numbers.txt");  
 Scanner scan = new Scanner(file);  
  
 Tree tree = new Tree();  
 tree.init();  
 int[] arr = new int[1000];  
 int treeComps;  
 int bubbleComps = 0;  
  
 /\* Scans numbers from file and places them in an array \*/  
 while (scan.hasNextInt()) {  
 for(int j = 0; j< arr.length; j++) {  
 int num = scan.nextInt();  
 arr[j] = num;  
 }  
 }  
  
 /\* Method called buildSort takes pre-sorted numbers from the  
 array and sorts them into a tree \*/  
 for (int j = 0; j < arr.length; j++){  
 tree.buildSort(arr[j]);  
 }  
  
 /\* inOrder prints the tree out in order from low to high\*/  
 tree.inOrder(tree.root);  
  
 /\* Method called comparisonTotal counts the number of comparisons \*/  
 treeComps = tree.compareTotal(arr);  
 System.*out*.println("\nTotal number of tree comparisons: " + treeComps);  
 scan.close();  
  
 int[] bubbleArray = new int[1000];  
 Random num = new Random();  
 int number;  
 for(int i = 0; i < 1000; i++){  
 number = num.nextInt(100);  
 bubbleArray[i] = number;  
 //System.out.println(bubbleArray[i]);  
 }  
  
 /\* Bubble Sort \*/  
 boolean swap = true;  
 while(swap) {  
 swap = false;  
  
 for (int j = 0; j < bubbleArray.length - 1; j++) {  
 bubbleComps++;  
 if (bubbleArray[j] > bubbleArray[j + 1]) {  
 swap = true;  
 int temp = bubbleArray[j];  
 bubbleArray[j] = bubbleArray[j + 1];  
 bubbleArray[j + 1] = temp;  
 }  
 }  
 }  
  
 System.*out*.println("Total number of bubble sort comparisons are: " + bubbleComps);  
 System.*out*.println("The tree sort program is more efficient");  
 System.*out*.println("Based on comparisons made, the tree sort is more efficient");  
 System.*out*.println("Based on memory used, the bubble sort is more efficient.");  
 }  
} // close main

**public class Node** {  
 public int data;  
 public Node left;  
 public Node right;  
 int occur = 1;  
  
 //constructor  
 public Node(int data) {  
 this.data = data;  
 this.left = null;  
 this.right = null;  
 }  
} // close Node

import java.util.Stack;  
**public class Tree** {  
 public Node root;  
 public Node left;  
 public Node right;  
 public Node addNode;  
 public Node curr;  
 int data;  
 int occur;  
 boolean searching = true;  
 int compareTree;  
 int total = 0;  
 int temp = 0;  
  
  
 //constructor to initialize tree  
 public void init() {  
 root = null;  
 }  
  
 public int buildSort(int num){ ///////// This works!!!!!!  
 Node createdNode;  
 if (root == null){  
 compareTree++;  
 occur++;  
 root = new Node(num);  
 }  
 else {  
 curr = root;  
 searching = true;  
 while(searching){  
 if (num == curr.data){  
 curr.occur++;  
 compareTree++;  
 searching = false;  
 }  
 else if (num< curr.data){  
 if (curr.left != null){  
 compareTree++;  
 curr = curr.left;  
 }  
 else{  
 curr.left = new Node(num);  
 compareTree++;  
 searching = false;  
 }  
 }  
 else if (num > curr.data){  
 if (curr.right != null){  
 compareTree++;  
 curr = curr.right;  
 }  
 else{  
 compareTree++;  
 curr.right = new Node(num);  
 searching = false;  
 }  
 }  
 }  
 }  
 return compareTree;  
 }  
public void inOrder(Node t){  
 if(t.left != null){  
 inOrder(t.left);  
 }  
 System.*out*.println(t.data);  
 if(t.right != null){  
 inOrder(t.right);  
 }  
}  
  
public void printIterative(Node t){  
 Stack<Integer> s = new Stack<Integer>();  
 s.push(root.data);  
 curr = root;  
 do{  
 while(curr != null){  
 s.push(curr.data);  
 curr = curr.left;  
 }  
 if (!s.empty()){  
 int num = s.pop();  
 System.*out*.println(num);  
 curr = curr.right;  
 }  
  
 }while(!s.empty()|| curr != null);  
}  
  
 public int compareTotal( int[] array){  
 for (int j = 0; j < array.length; j++) {  
 int num = array[j];  
 if (root == null) {  
 root = new Node(num);  
 } else {  
 curr = root;  
 searching = true;  
 while (searching) {  
 compareTree++;  
 if (num == curr.data) {  
 searching = false;  
 } else if (num < curr.data) {  
 if (curr.left != null) {  
 curr = curr.left;  
 } else {  
 curr.left = new Node(num);  
 searching = false;  
 }  
 } else if (num > curr.data) {  
  
 if (curr.right != null) {  
 curr = curr.right;  
 } else {  
 curr.right = new Node(num);  
 searching = false;  
 }  
 }  
 }  
 }  
 }  
 return compareTree;  
 }  
 public int compareTree(int[] array){  
 for(int j = 0; j< array.length; j++) {  
 compareTree = buildSort(array[j]);  
 temp = compareTree + temp;  
 total = temp;  
 }  
 return total;  
 }  
}