CSCI 321 Computer Science III Fall 2018

Midterm Exam

1. Perform an experimental analysis to test the hypothesis that Java’s Array.sort method runs in O(nlogn) time on average. You should try several runs over many different problem sizes (at least 5 input sizes, input size should be greater than 50). Attach your code along with screenshots here. Draw a chart showing the running time against the input size.

import java.util.Arrays;

public class quest1{

public static void main(String[] args) {

double[] array1 = new double[100];

double[] array2 = new double[750];

double[] array3 = new double[250];

double[] array4 = new double[450];

double[] array5 = new double[1000];

double[][] arrayList = {array1, array2, array3, array4, array5};

for(int i = 0;i<arrayList.length;i++){

populateArray(arrayList[i]);

}

for(int i = 0; i < 5; i++){

long startTime = System.nanoTime();

Arrays.sort(arrayList[i]);

long endTime = System.nanoTime();

long elapsed = endTime - startTime;

System.out.println("elapsed time for array"+ i + " : " + elapsed);

}

}

public static void populateArray(double[] x){

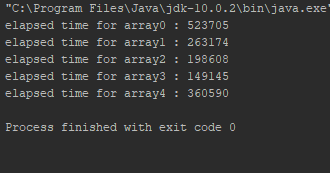
for(int i = 0; i < x.length; i++){

x[i] = Math.random();

}

}

}



|  |  |
| --- | --- |
| Array Size | Time (in milliseconds) |
| 100 | 477263 |
| 250 | 115162 |
| 450 | 333782 |
| 700 | 286584 |
| 1000 | 561464 |

1. The program below is used to construct an expression tree. You need to fill the body for the “inorder” function (which does inorder traversal) so that the output is:

infix expression is

5 + 6 - 8 \* 3 \* 2

Attach your code and screenshots of the output here.A screenshot of a cell phone

Description generated with very high confidence

// Java program to construct an expression tree

import java.util.Stack;

// Java program for expression tree

class Node {

char value;

Node left, right;

Node(char item) {

value = item;

left = right = null;

}

}

class ExpressionTree {

// A utility function to check if 'c'

// is an operator

boolean isOperator(char c) {

if (c == '+' || c == '-'

|| c == '\*' || c == '/'

|| c == '^') {

return true;

}

return false;

}

// Utility function to do inorder traversal

void inorder(Node t) {

if (t != null) {

inorder(t.left);

System.out.print(t.value + " ");

inorder(t.right);

}

}

// Returns root of constructed tree for given

// postfix expression

Node constructTree(char postfix[]) {

Stack<Node> st = new Stack();

Node t, t1, t2;

// Traverse through every character of

// input expression

for (int i = 0; i < postfix.length; i++) {

// If operand, simply push into stack

if (!isOperator(postfix[i])) {

t = new Node(postfix[i]);

st.push(t);

} else // operator

{

t = new Node(postfix[i]);

// Pop two top nodes

// Store top

t1 = st.pop(); // Remove top

t2 = st.pop();

// make them children

t.right = t1;

t.left = t2;

// System.out.println(t1 + "" + t2);

// Add this subexpression to stack

st.push(t);

}

}

// only element will be root of expression

// tree

t = st.peek();

st.pop();

return t;

}

public static void main(String args[]) {

ExpressionTree et = new ExpressionTree();

String postfix = "56+83\*2\*-";

char[] charArray = postfix.toCharArray();

Node root = et.constructTree(charArray);

System.out.println("infix expression is");

et.inorder(root);

}

}

1. The following program is used to demonstrate working of priority queue in Java. You need to use the methods in priority queue class to perform the operations commented in the code. Attached your complete code and screenshots of the output here.

A screenshot of a cell phone

Description generated with very high confidence

1 Code removing java from the priority list, note the value "C" is still present and contains C is marked TrueA screenshot of a cell phone

Description generated with very high confidence

2 Program run a second time removing the C value from the queue, and the Boolean statement at the end now denotes false saying the queue is no longer containing the value C.

import java.util.\*;

import java.util.Scanner;

class quest3 {

// Java program to demonstrate working of priority queue in Java

public static void main(String args[]){

// Creating empty priority queue

PriorityQueue<String> pQueue =

new PriorityQueue<String>();

// Adding items to the pQueue using add()

pQueue.add("C");

pQueue.add("C++");

pQueue.add("Java");

pQueue.add("Python");

// Printing the most priority element

System.out.println("Priority queue lead value: " + pQueue.peek());

// Printing all elements

System.out.println("The queue elements:");

Iterator itrQue = pQueue.iterator();

while (itrQue.hasNext())

System.out.println(itrQue.next());

// Removing the top priority element (or head) and

// printing the modified pQueue

System.out.println("Please input the queue object you would like to remove: ");

Scanner scanner = new Scanner(System.in);

String remQueue = scanner.nextLine();

pQueue.remove(remQueue);

System.out.println("The new priority queue after removing " + remQueue);

Iterator<String> itr3 = pQueue.iterator();

while (itr3.hasNext())

System.out.println(itr3.next());

// Check if “C” is present

boolean b = pQueue.contains("C");

System.out.println ( "Does the priority queue contains C ? " + b);

}

}

///////Question 2 original code:

// Java program to construct an expression tree

import java.util.Stack;

// Java program for expression tree

class Node {

char value;

Node left, right;

Node(char item) {

value = item;

left = right = null;

}

}

class ExpressionTree {

// A utility function to check if 'c'

// is an operator

boolean isOperator(char c) {

if (c == '+' || c == '-'

|| c == '\*' || c == '/'

|| c == '^') {

return true;

}

return false;

}

// Utility function to do inorder traversal

void inorder(Node t) {

**// Implement the inorder traversal here**

}

// Returns root of constructed tree for given

// postfix expression

Node constructTree(char postfix[]) {

Stack<Node> st = new Stack();

Node t, t1, t2;

// Traverse through every character of

// input expression

for (int i = 0; i < postfix.length; i++) {

// If operand, simply push into stack

if (!isOperator(postfix[i])) {

t = new Node(postfix[i]);

st.push(t);

} else // operator

{

t = new Node(postfix[i]);

// Pop two top nodes

// Store top

t1 = st.pop(); // Remove top

t2 = st.pop();

// make them children

t.right = t1;

t.left = t2;

// System.out.println(t1 + "" + t2);

// Add this subexpression to stack

st.push(t);

}

}

// only element will be root of expression

// tree

t = st.peek();

st.pop();

return t;

}

public static void main(String args[]) {

ExpressionTree et = new ExpressionTree();

String postfix = "56+83\*2\*-";

char[] charArray = postfix.toCharArray();

Node root = et.constructTree(charArray);

System.out.println("infix expression is");

et.inorder(root);

}

}

///////Question 3 original code:

// Java program to demonstrate working of priority queue in Java

import java.util.\*;

class Example

{

public static void main(String args[])

{

// Creating empty priority queue

PriorityQueue<String> pQueue =

new PriorityQueue<String>();

// Adding items to the pQueue using add()

pQueue.add("C");

pQueue.add("C++");

pQueue.add("Java");

pQueue.add("Python");

// Printing the most priority element

**Add your code here.**

// Printing all elements

System.out.println("The queue elements:");

**Add your code here.**

// Removing the top priority element (or head) and

// printing the modified pQueue

**Add your code here.**

// Check if “C” is present

**Add your code here.**

}

}