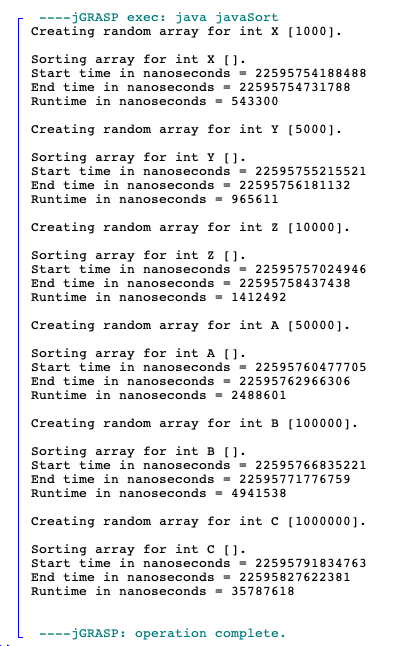
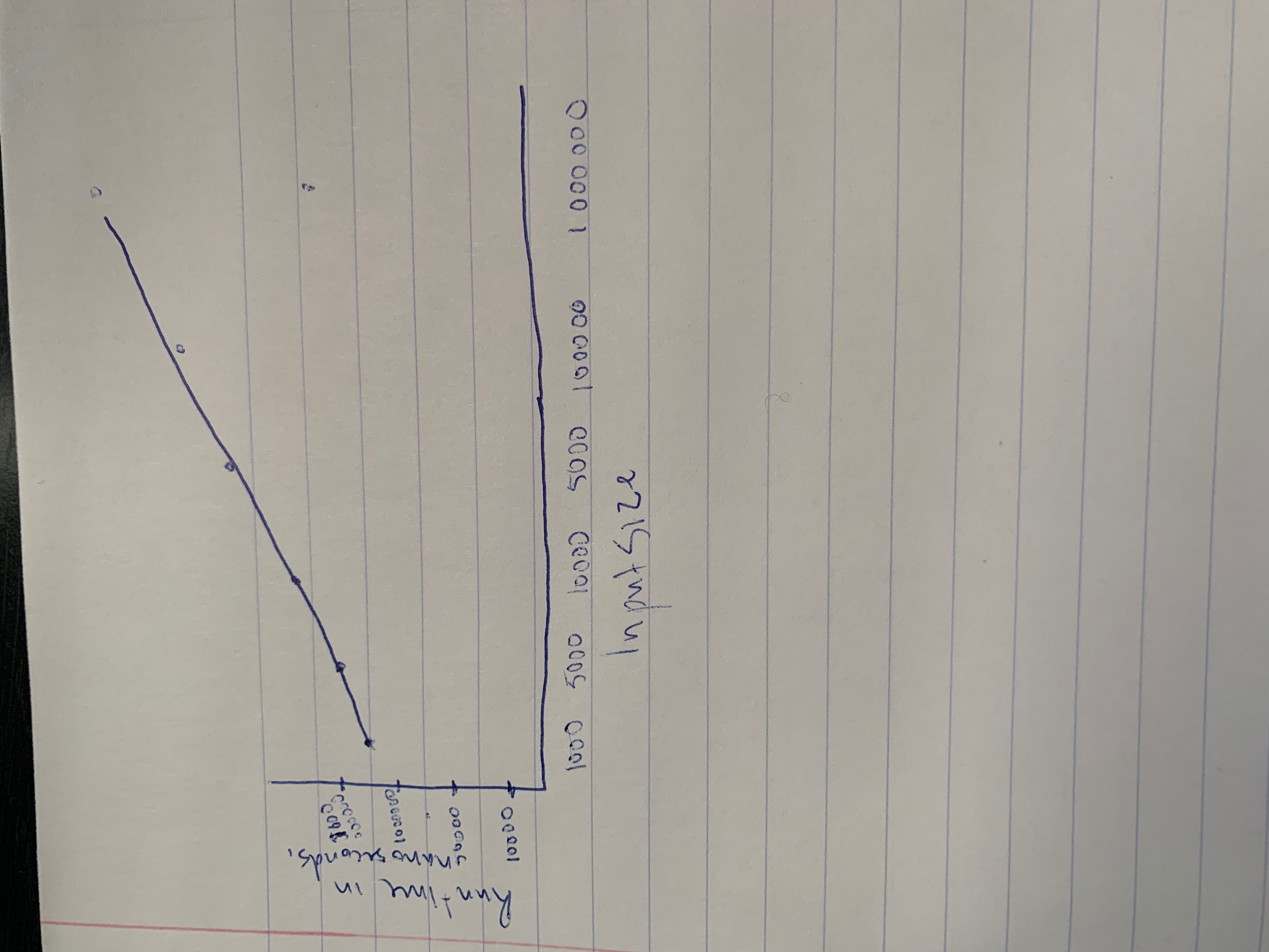
CSCI 321 Computer Science III Summer 2019

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.

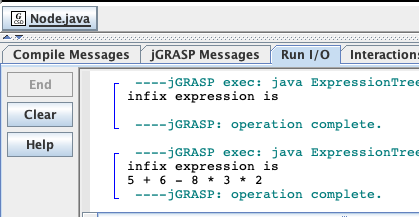
1 import java.util.\*;  
 2 import java.io.\*;  
 3 import java.lang.\*;  
 4   
 5 public class javaSort{  
 6 public static void main(String[] args){  
 7 int X [] = new int[1000];  
 8 int n = X.length;  
 9 Random rand = new Random();  
 10 long startTime = 0;  
 11 long endTime = 0;  
 12 long time = 0;  
 13   
 14 System.out.println("Creating random array for int X [1000].");  
 15 for(int i=0; i<n; i++){  
 16 X[i] = rand.nextInt(100);  
 17 }  
 18 System.out.println();  
 19   
 20 System.out.println("Sorting array for int X [].");  
 21 startTime = System.nanoTime();  
 22 System.out.print("Start time in nanoseconds = ");  
 23 System.out.println(startTime);  
 24 Arrays.sort(X);   
 25 endTime = System.nanoTime();  
 26 System.out.print("End time in nanoseconds = ");  
 27 System.out.println(endTime);  
 28 time = endTime - startTime;  
 29 System.out.print("Runtime in nanoseconds = ");  
 30 System.out.println(time);  
 31 System.out.println();  
 32   
 33   
 34 int Y [] = new int[5000];  
 35 n = Y.length;  
 36 System.out.println("Creating random array for int Y [5000].");  
 37 for(int i=0; i<n; i++){  
 38 Y[i] = rand.nextInt(100);  
 39 }  
 40 System.out.println();  
 41   
 42 System.out.println("Sorting array for int Y [].");  
 43 startTime = System.nanoTime();  
 44 System.out.print("Start time in nanoseconds = ");  
 45 System.out.println(startTime);  
 46 Arrays.sort(Y);  
 47 endTime = System.nanoTime();  
 48 System.out.print("End time in nanoseconds = ");  
 49 System.out.println(endTime);  
 50 time = endTime - startTime;  
 51 System.out.print("Runtime in nanoseconds = ");  
 52 System.out.println(time);  
 53 System.out.println();  
 54   
 55 int Z [] = new int[10000];  
 56 n = Z.length;  
 57 System.out.println("Creating random array for int Z [10000].");  
 58 for(int i=0; i<n; i++){  
 59 Z[i] = rand.nextInt(100);  
 60 }  
 61 System.out.println();  
 62   
 63 System.out.println("Sorting array for int Z [].");  
 64 startTime = System.nanoTime();  
 65 System.out.print("Start time in nanoseconds = ");  
 66 System.out.println(startTime);  
 67 Arrays.sort(Z);  
 68 endTime = System.nanoTime();  
 69 System.out.print("End time in nanoseconds = ");  
 70 System.out.println(endTime);  
 71 time = endTime - startTime;  
 72 System.out.print("Runtime in nanoseconds = ");  
 73 System.out.println(time);  
 74 System.out.println();  
 75   
 76 int A [] = new int[50000];  
 77 n = A.length;  
 78 System.out.println("Creating random array for int A [50000].");  
 79 for(int i=0; i<n; i++){  
 80 A[i] = rand.nextInt(100);  
 81 }  
 82 System.out.println();  
 83   
 84 System.out.println("Sorting array for int A [].");  
 85 startTime = System.nanoTime();  
 86 System.out.print("Start time in nanoseconds = ");  
 87 System.out.println(startTime);  
 88 Arrays.sort(A);  
 89 endTime = System.nanoTime();  
 90 System.out.print("End time in nanoseconds = ");  
 91 System.out.println(endTime);  
 92 time = endTime - startTime;  
 93 System.out.print("Runtime in nanoseconds = ");  
 94 System.out.println(time);  
 95 System.out.println();  
 96   
 97 int B [] = new int[100000];  
 98 n = B.length;  
 99 System.out.println("Creating random array for int B [100000].");  
100 for(int i=0; i<n; i++){  
101 B[i] = rand.nextInt(100);  
102 }  
103 System.out.println();  
104   
105 System.out.println("Sorting array for int B [].");  
106 startTime = System.nanoTime();  
107 System.out.print("Start time in nanoseconds = ");  
108 System.out.println(startTime);  
109 Arrays.sort(B);  
110 endTime = System.nanoTime();  
111 System.out.print("End time in nanoseconds = ");  
112 System.out.println(endTime);  
113 time = endTime - startTime;  
114 System.out.print("Runtime in nanoseconds = ");  
115 System.out.println(time);  
116 System.out.println();  
117   
118 int C [] = new int[1000000];  
119 n = C.length;  
120 System.out.println("Creating random array for int C [1000000].");  
121 for(int i=0; i<n; i++){  
122 C[i] = rand.nextInt(100);  
123 }  
124 System.out.println();  
125   
126 System.out.println("Sorting array for int C [].");  
127 startTime = System.nanoTime();  
128 System.out.print("Start time in nanoseconds = ");  
129 System.out.println(startTime);  
130 Arrays.sort(C);  
131 endTime = System.nanoTime();  
132 System.out.print("End time in nanoseconds = ");  
133 System.out.println(endTime);  
134 time = endTime - startTime;  
135 System.out.print("Runtime in nanoseconds = ");  
136 System.out.println(time);  
137 System.out.println();  
138   
  
152   
153 }  
154   
155 }



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.

// 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**

if(t==null){  
35 return;  
36 }  
37 inorder(t.left);  
38 System.out.print(t.value + " ");  
39 inorder(t.right);  
40

}

// 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.

// 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

// Printing the most priority element   
19 System.out.println(pQueue.peek());  
20   
21 // Printing all elements   
22 System.out.println("The queue elements:");   
23 Iterator i = pQueue.iterator();  
24 while (i.hasNext())  
25 System.out.println(i.next());  
26   
27 // Removing the top priority element (or head) and   
28 // printing the modified pQueue   
29 pQueue.poll();  
30 System.out.println("After removing element" + "with poll function: ");  
31 Iterator<String> j = pQueue.iterator();  
32 while (j.hasNext())  
33 System.out.println(j.next());  
34   
35 // Check if “C” is present   
36 boolean b = pQueue.contains("C");  
37 System.out.println("Priority queue contains C " + "or not? " + b);

// 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.**

}

}

