|  |  |
| --- | --- |
|  | **University of Maryland College Park** |
| **Dept of Computer Science** |
| **CMSC132 Fall 2015** |
| **Midterm II Key** |

First Name (PRINT): **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Last Name (PRINT): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

University Directory ID (e.g., testudoJr) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

I pledge on my honor that I have not given or received any unauthorized assistance on this examination.

Your signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Instructions**

* This exam is a closed-book and closed-notes exam.
* Total point value is 200 pts.
* The exam is a 50 minutes exam.
* Please use a pencil to complete the exam.
* There are seven problems in the exam.
* **WRITE NEATLY**.

***Grader Use Only***

|  |  |  |  |
| --- | --- | --- | --- |
| #1 | Algorithmic Complexity | (24) |  |
| #2 | Generics | (20) |  |
| #3 | Hashing | (12) |  |
| #4 | Testing/Program Correctness | (20) |  |
| #5 | Software Engineering | (14) |  |
| #6 | Linear Data Structures | (50) |  |
| #7 | Trees | (60) |  |
| **Total** | Total | (200) |  |

**Problem #1 Algorithmic Complexity**

1. (14 pts) For the following problems you need to provide the asymptotic complexity using big-O notation. In addition, you need to identify the critical section (circle it) and the time function (Time 🡪 below). Here is an example:

**System.out.println("Hello");**

**for (j = 1; j <= n; j++) {**

**System.out.println(j);**

**}**

**System.out.println("Goodbye");**

**Time** 🡪 1 + n + 1

**Big-O** 🡪 O(n)

1. (7 pts)

**for (i = 1; i <= n / 4; i++) {**

**for (j = 50; j < n; j++) {**

**System.out.println(i \* j);**

**}**

**}**

**for (j = 1; j < n; j++) {**

**System.out.println(i);**

**}**

**Time 🡪 ((**n / 4) \* (n - 50)) + (n – 1)

**Big-O 🡪** O(n2)

**Critical Section 🡪** First System.out.println

1. (7 pts)

**System.out.println("Analysis");**

**j = 3;**

**while (j < n) {**

**for (i = 1; i < n; i \*= 2) {**

**System.out.println(i);**

**}**

**j++;**

**System.out.println(j);**

**}**

**for (k = 1; k <= 20; k++) {**

**System.out.println(k);**

**}**

**Time 🡪** 1 + (n – 3)(log(n) + 1) + 1

**Big-O 🡪** n(log(n))

**Critical Section 🡪** Second System.out.println

1. (6 pts) List the following big-O expressions in order of asymptotic complexity (lowest complexity first).

O(**n**log(**n**)) O(log(**n**)) O(**nn**) O(**nk**) O(**n**3)

Answer: O(log(**n**)) O(**n**log(**n**)) O(**n**3) O(**nk**) O(**nn**)

1. (4 pts) When running the implementation of an algorithm the following running times were found:

**Data Size** **Running Time**

39 80

78 158

156 315

What is the big-O of the algorithm?

Answer: O(n)

**Problem #2 Generics**

You need to update the ThreeElementList class (feel free to edit/cross out the provided code) so it becomes a generic class that takes two generic parameters T and V. The class keeps track of three objects of type T. An object of type V is used to identify the list. One you have edited the class the provided main function below must generate the following output: **56localgoodbyehola**

**public class ThreeElementList {**

**private String[] data;**

**int numberElements;**

**private Integer id;**

**public ThreeElementList(Integer id) {**

**data = new String[3];**

**numberElements = 0;**

**this.id = id;**

**}**

**public void addElement(String element) {**

**data[numberElements++] = element;**

**numberElements = numberElements++ % 3;**

**}**

**public String returnFirst() {**

**return data[0];**

**}**

**public String toString() {**

**String answer = id.toString();**

**for (String elem : data) {**

**answer += elem;**

**}**

**return answer;**

**}**

**public static void main(String[] args) {**

**Integer id = 56;**

**ThreeElementList<String, Integer> list = new ThreeElementList<String, Integer>(id);**

**list.addElement("Hello");**

**list.addElement("goodbye");**

**list.addElement("hola");**

**list.addElement("local");**

**System.out.println(list);**

**}**

**}**

Answer:

public class ThreeElementList<T, V> {

private T[] data;

int numberElements;

private V id;

public ThreeElementList(V id) {

data = (T[]) new Object[3];

numberElements = 0;

this.id = id;

}

public void addElement(T element) {

data[numberElements++] = element;

numberElements = numberElements++ % 3;

}

public T returnFirst() {

return data[0];

}

public String toString() {

String answer = id.toString();

for (T elem : data) {

answer += elem;

}

return answer;

}

public static void main(String[] args) {

Integer id = 56;

ThreeElementList<String, Integer> list = new ThreeElementList<String, Integer>(id);

list.addElement("Hello");

list.addElement("goodbye");

list.addElement("hola");

list.addElement("local");

System.out.println(list);

}

}

**Problem #3 Hashing**

1. (2 pts) Name one property discussed in class associated with a good hash function.

Answer: Distributes values well, not expensive

1. (10 pts) Does the following class satisfy the Java Hash Code Contract? Yes or No answers without explanation will receive no credit.

**public class Banana {**

**private String type;**

**private int age;**

**public Banana(String type, int age) {**

**this.type = type;**

**this.age = age;**

**}**

**public boolean equals(Object obj) {**

**if (obj == this) {**

**return true;**

**}**

**if (!(obj instanceof Banana)) {**

**return false;**

**}**

**return type.equals(((Banana)obj).type);**

**}**

**public int hashcode() {**

**if (type.length() + age <= 10) {**

**return type.length();**

**}**

**return 0;**

**}**

**}**

Answer: No. equals method does not rely on age therefore we can have objects that are equal (same type) but their hash code will be different (e.g., new Banana(“yellow”, 0), new Banana(“yellow”, 1000)).

**Problem #4 Testing/Program Correctness**

For this problem you need to:

1. Modify the evaluateEmployee method so it throws the exception InvalidNameException when a null parameter or a name with less than 4 characters is provided. Notice no output should be generated by the method when an invalid argument is provided. The error message associated with the exception object is “Invalid string” followed by the invalid string.

Answer:

**if (name == null || name.length() < 4) {**

**throw new InvalidNameException("Invalid string " + name);**

**}**

1. Complete the main method so the program reads a string from the user (using scanner.next()), calls evaluateEmployee with the string provided, and keeps calling evaluateEmployee as long as an invalid value is entered. Notice you cannot check in main whether the string is valid; only evaluateEmployee can check the string validity. Each time an invalid value is provided your code must print the error message associated with the InvalidNameException.

Answer:

**while(true) {**

**try {**

**evaluateEmployee(scanner.next());**

**break;**

**} catch(InvalidNameException e) {**

**System.out.println(e.getMessage());**

**}**

**}**

**public class DataSystem {**

**public static void evaluateEmployee(String name) {**

**if (name.equals("Peter")) {**

**System.out.println("Supervisor");**

**} else if (name.equals("Laura")) {**

**System.out.println("Manager");**

**} else {**

**System.out.println("AreaManager");**

**}**

**}**

**public static void main(String[] args) {**

**Scanner scanner = new Scanner(System.in);**

**scanner.close();**

**}**

**}**

**Problem #5 Software Engineering**

1. (6 pts) Problem specification is one of the components of the software life cycle; name two additional ones.

Answer: Any two of program design, algorithms/data structures, coding/debugging, testing/verifications, documentation/support, Maintenance

1. (4 pts) Which software process model emphasizes predictability?

Answer: waterfall model

1. (4 pts) One of the architecture styles seen in class was blackboard. What is the other? Briefly describe it.

Answer: pipes and filters

**Problem #6 Linear Data Structures**

Use the following classes to implement the delete method. You may not add any instance variables or

static variables to either class, and you may not add any methods to the Node class. Feel free to add any

auxiliary non-static methods to the LinkedList class. You may not use the Java API LinkedList class.

**public class LinkedList<T extends Comparable<T>> {**

**private class Node {**

**private T data;**

**private Node next;**

**public Node(T data) {**

**this.data = data;**

**next = null;**

**}**

**}**

**private Node head; /\* List head pointer \*/**

**public boolean delete(T targetElement) { /\* You must write this method \*/ }**

**}**

Implement the method **delete** that removes all instances of targetElement from the list. The method will return true if any element is deleted; false otherwise. Use compareTo to compare elements.

**Answer:**

public boolean delete(T targetElement) {

if (head == null)

return false;

boolean elementRemoved = false;

Node prev = null, curr = head;

while (curr != null) {

if (curr.data.compareTo(targetElement)==0) {

if (curr == head) {

curr = curr.next;

head = curr;

} else {

prev.next = curr.next;

curr = curr.next;

}

elementRemoved = true;

} else {

prev = curr;

curr = curr.next;

}

}

return elementRemoved;

}

**Problem #7 Trees**

Use the following classes to answer the questions below. You may not add any instance variables or

static variables to either class, and you may not add any methods to the Node class. Feel free to add any

auxiliary non-static methods to the BinarySearchTree class.

**public class BinarySearchTree <K extends Comparable<K>, V> {**

**private class Node {**

**private K key;**

**private V data;**

**private Node left, right;**

**public Node(K key, V data) {**

**this.key = key;**

**this.data = data;**

**}**

**}**

**private Node root;**

**public V getMaximumKeyDataValue() { /\* You must write this method \*/ }**

**public void removeLeaves() { /\* You must write this method \*/ }**

**}**

1. Write the method **getMaximumKeyDataValue** that returns the **data** value associated with the key that has the maximum key value. For an empty tree the method will return null.

**One possible answer:**

**public V getMaximumKeyDataValue() {**

**return root == null ? null : getMaximumKeyDataValueAux(root);**

**}**

**private V getMaximumKeyDataValueAux(Node rootAux) {**

**return rootAux.right == null ? rootAux.data:getMaximumKeyDataValueAux(rootAux.right);**

**}**

1. Write the method **removeLeaves** that removes the leaves of the tree. The leaves of a binary search tree are those that have no children. Keep in mind that the binary search tree property doesn't have to be used to write this method. A tree with a single element will become an empty tree (represented by a root being null), after calling this method, while a tree that initially has no elements automatically has no leaves, so should remain empty.

**One possible answer:**

**public void removeLeaves() {**

**if (root != null)**

**if (root.left == null && root.right == null)**

**root = null;**

**else removeLeavesAux(root);**

**}**

**private void removeLeavesAux(Node rootAux) {**

**if (rootAux != null) {**

**if (rootAux.left != null)**

**if (rootAux.left.left == null && rootAux.left.right == null)**

**rootAux.left= null;**

**else removeLeavesAux(rootAux.left);**

**if (rootAux.right != null)**

**if (rootAux.right.left == null && rootAux.right.right == null)**

**rootAux.right= null;**

**else removeLeavesAux(rootAux.right);**

**}**

**}**

**Another possible answer:**

**public void removeLeaves() {**

**if (root == null)**

**return;**

**else if (root.left == null && root.right == null)**

**root = null;**

**else {**

**removeLeavesAux(root.left, root);**

**removeLeavesAux(root.right, root);**

**}**

**}**

**private void removeLeavesAux(Node rootAux, Node parent) {**

**if (rootAux.left == null && rootAux.right == null)**

**if (rootAux == parent.left)**

**parent.left = null;**

**else**

**parent.right = null;**

**else if (rootAux.left == null && rootAux.right != null)**

**removeLeavesAux(rootAux.right, rootAux);**

**else if (rootAux.left != null && rootAux.right == null)**

**removeLeavesAux(rootAux.left, rootAux);**

**else {**

**removeLeavesAux(rootAux.left, rootAux);**

**removeLeavesAux(rootAux.right, rootAux);**

**}**

**}**