|  |  |
| --- | --- |
|  | **University of Maryland College Park** |
| **Dept of Computer Science** |
| ***CMSC132 Summer 2012*** |
| ***Midterm II*** |

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

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

*University ID:* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*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 100 points.*
* *The exam is a 50 minutes exam.*
* *Please use a pencil to complete the exam.*
* ***WRITE NEATLY****. If we cannot understand your answer, we will not grade it (i.e., 0 credit).*

***Grader Use Only***

|  |  |  |  |
| --- | --- | --- | --- |
| #1 | Algorithmic Complexity | (20) |  |
| #2 | Language Features/Program Correctness | (25) |  |
| #3 | Sets and Maps | (15) |  |
| #4 | Linear Data Structures | (40) |  |
| **Total** | Total | (100) |  |

**Problem 1 (20 pts) Algorithmic Complexity**

1. (6 pts) Calculate the asymptotic complexity of the code snippets below (using big-O notation) with

respect to the problem size **n**.

* 1. f(**n**) = O( )

for (i = 0; i <= n; i++) {

for (j = 1; j < 50; j = j + 2) {

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

}

}

* 1. f(**n**) = O( )

for (i = 0; i < n - 2; i++) {

for (j = 0; j < 2 \* n; j = j + 1) {

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

}

}

* 1. f(**n**) = O( )

for (int i = 0; i <= 2; i++) {

for (int k = n; k <= n; k++) {

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

}

}

1. (4 pts) Give the asymptotic bound of the following functions:

1. **n**2log(**n**) + 3**n**2 f(**n**) = O( )
2. **n**2 + 4**n** f(**n**) = O( )
3. (4 pts) List the following big-O expressions in order of asymptotic complexity (lowest complexity first).

O(log(**n**)) O(**n3**) O(**n**) O(1)

1. (2 pts) Indicate the complexity (big O) for an algorithm whose running time roughly doubles as input size doubles.
2. (2 pts) What kind of analysis suggests we should double the size of an array while expanding its size?
3. (2 pts) What is maximum number of critical sections an algorithm can have?

**Problem 2 (25 pts) Language Features/Program Correctness**

1. (2 pts) Complete the following declaration so we can define an array of **T** elements with a number of elements that corresponds to **size**

**T**[] data =

1. (2 pts) Name two immutable classes seen in class.
2. (2 pts) Circle the constructor that will be called when we execute ***new Ex(null)***:

public Ex(double[] d) { } public Ex(Object d) { }

1. (2 pts) Name one advantage that an array has over a linked list?
2. (2 pts) What kind of exception (checked or unchecked) require us to declare a catch block or use *throws*?
3. (2 pts) Suppose you are designing a network system that requires users to provide a web address. What kind of exception you need to use (checked or unchecked)? Answers without explanation will receive no credit.
4. (3 pts) Describe the Java hash code contract.
5. (2 pts) What you need to do in order to write tests that only generate 50% test coverage?
6. (8 pts) The following program has several errors (including compilation ones). Mention those errors.

public class Problem {

private double x;

public void Problem(int x) {

this.x = x;

}

public int hashCode() {

return x;

}

public boolean equals(Problem obj){

if (obj == this)

return true;

if (!(obj instanceof Problem)) {

return false;

}

Problem p = (Problem) obj;

return p.x == x;

}

}

**Problem 3 (15 pts) Sets and Maps**

Implement a **getPeoplesDesserts** method that returns a map associating a person’s name with the dessert they like. The first parameter maps a person’s name to her/his id, and the second one a person’s id to the dessert they like. About this problem:

* You don’t need to do add/write code for main (don’t change it). We are providing it to clarify the functionality associated with the **getPeoplesDesserts** method you need to implement. The program’s output is ***{Jose=Chocolate, Maria=Flan}***.
* The map returned by **getPeoplesDesserts** should allow us to retrieve people in alphabetical order.
* If a person does not have an entry in the dessert map, the person will not be added to the map returned by the method.

public class Utilities {

public static void main(String[] args) {

Map<String, Integer> people = // a map object

// Adding to people’s map "Maria", 1

// Adding to people’s map "Jose", 2

Map<Integer, String> desserts = // a map object

// Adding to dessert’s map 1, "Flan"

// Adding to dessert’s map 2, "Chocolate"

System.out.println(getPeoplesDesserts(people, desserts));

}

public static Map<String, String> getPeoplesDesserts(Map<String, Integer> people, Map<Integer, String> desserts) {

**You may find the following Map methods helpful:**

* V **get**(Object key) - Returns the value to which this map maps the specified key.
* V **put**(K key,V value) - Associates the specified value with the specified key in this map.
* Set<K> **keySet**() - Returns a set view of the keys contained in this map.
* boolean **isEmpty**() - Returns true if this map contains no key-value mappings.

**You may find the following Set methods helpful:**

* boolean **contains**(Object o) - Returns true if this set contains the specified element.
* boolean **add**(E o) - Adds the specified element to this set if it is not already present.
* V **remove**(Object key) - Removes the element from the set.
* boolean **isEmpty**() - Returns true if this set contains no element.

**Problem 4 (40 pts) Linear Data Structures**

Implement the methods below based on the following Java class definitions. You may not modify the Node class and you may not add any instance or static variables. In addition, 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;

public LinkedList() { head = null; }

public boolean sameValues(LinkedList<T> L) { /\* You must implement \*/ }

public void removeSmallerOrEqualTo(T elem) { /\* You must implement \*/ }

}

1. Implement the method **sameValues** that determines whether two lists have the same values. Two lists are considered the same if they have the same length and the same values (in the same order). Two empty lists are considered the same. For this problem:
   1. You must implement a **RECURSIVE** solution, otherwise you will not get credit.
   2. The lists can have different sizes and they can be empty.
   3. Assume the lists are not sorted.
   4. You may add only one auxiliary method.
   5. Compare elements using compareTo.
2. Implement the method **removeSmallerOrEqualTo** that removes all elements from the list that are smaller or equal to the parameter. The method may have the effect of removing any elements, all elements, or no elements from the list. For this problem:
   1. You must implement a **NON-RECURSIVE** solution, otherwise you will not get credit.
   2. Assume the list is not sorted.
   3. You may not add auxiliary methods.

**PAGE FOR PREVIOUS PROBLEM**

**EXTRA PAGE IN CASE YOU NEED IT**