|  |  |
| --- | --- |
|  | **University of Maryland College Park** |
| **Dept of Computer Science** |
| **CMSC132 Fall 2016** |
| **Midterm II** |

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

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

**University Directory ID** (e.g., umcpturtle): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Lab TA (Circle One):**

|  |  |  |
| --- | --- | --- |
| 0401, 3 pm, Ladan | 0403 10 am, Travis | Honors, 5 pm, Andrej |
| 0402, 4 pm, Saurabh | 0404 5 pm, Saurabh |  |

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 points.
* The exam is a 50 minutes exam.
* Please use a pencil to complete the exam.
* WRITE NEATLY.
* Your code must be efficient.
* You don’t need to use meaningful variable names; however, we expect good indentation.
* Write your name and section number before you answer the exam.

**Grader Use Only**

|  |  |  |  |
| --- | --- | --- | --- |
| #1 | Problem #1 (Algorithmic Complexity) | 25 |  |
| #2 | Problem #2 (Miscellaneous) | 70 |  |
| #3 | Problem #3 (Linear Data Structures) | 105 |  |
|  | Honors | 15 |  |
| **Total** | Total | 200/215 |  |

**Problem #1 (Algorithmic Complexity)**

1. (15 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:

|  |  |
| --- | --- |
| **for (j = 1; j <= n; j++) {**  **System.out.println(j);**  **}**  **System.out.println("Goodbye");** | **Time 🡪** n + 1  **Big O 🡪** O(n) |

1. (5 pts)

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

**for (k = 100; k <= n; k++) {**

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

**}**

**}**

**Time 🡪**

**Big O 🡪**

1. (5 pts)

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

**j = 1;**

**while (j <= n \* n \* n) {**

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

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

**}**

**j++;**

**}**

**Time 🡪**

**Big O 🡪**

1. (5 pts)

**for (i = 1; i <= 600; i++) {**

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

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

**}**

**}**

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

**System.out.println(i \* 50);**

**}**

**Time 🡪**

**Big O 🡪**

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

O(**n**log(**n**)) O(**kn**) O(**nn**) O(**n3**) O(**nk**)

1. (4 pts) Indicate the complexity (Big O) for an algorithm whose running time does not change when input size doubles.

**Problem #2 (Miscellaneous)**

1. (4 pts) Is the following assignment allowed? If it is allowed, can we perform any kind of operations using **map**? Yes or no answers without explanation will receive no credit.

**TreeMap<?, Integer> map = new TreeMap<String, Integer>();**

1. (18 pts) Modify (feel free to cross out and edit) the following **Shelf** class so it becomes a Generic class where the **String** class can be replaced with any class.

**public class Shelf {**

**private String[] items;**

**private int capacity, itemsNumber = 0;**

**private String mainItem;**

**public Shelf(int capacity) {**

**items = (String[]) new String[capacity];**

**}**

**public void add(String item) {**

**if (!item.equals(mainItem)) items[itemsNumber++] = item;**

**}**

**public String getFirst() { return items[0]; }**

**public int getCapacity() { return capacity; }**

**}**

1. (2 pts) You need to improve the speed at which messages are stored in the file system for an e-mail manager software. Which component of the Model View Controller Paradigm are you improving?
2. (4 pts) Which of the following are considered true? Circle all that apply.
   1. The clone method associated with every class returns a deep copy.
   2. The clone method is defined as a protected method.
   3. We cannot implement a copy constructor using the clone method.
   4. The return type of the clone method is the name of the class associated with the clone method.
3. (4 pts) A computer company sells a basic computer to which customers can add the following options: optical mouse, lock, identity-protection software, and 2 GB memory modules. Each of these options can be combined in any way customers prefer (e.g., multiple 2 GB memory modules can be added to a computer). Which of the following design patterns is preferred in order to implement a software that allow us to customize a computer? Circle only one.
   1. Marker design pattern
   2. State design pattern
   3. Decorator design pattern
   4. Iterator design pattern
   5. Computer design pattern
4. (4 pts) Does the default implementation of the equals and hashCode methods for a Java class satisfy the Java Hash Code contract? Explain briefly. Yes or no answers without explanation will receive no credit.
5. (14 pts) The following class will be used for the questions below.

**public class Road {**

**private final int roadNumber;**

**private int length;**

**public Road(int roadNumber, int length) { this.roadNumber = roadNumber; this.length = length; }**

**public void increaseLength() { length++; };**

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

**if (obj == this) return true;**

**if (!(obj instanceof Road))return false;**

**return roadNumber == ((Road) obj).roadNumber;**

**}**

**public int hashCode() { return length \* roadNumber; }**

**}**

1. Is the Java Hash Code contract satisfied by the Road class? Yes or no answers without explanation will receive no credit.
2. Provide the most **inefficient** (yes, not efficient) hashCode method for this class. Explain why it is the most **inefficient**.
3. Provide the most **efficient** hashCode method for this class. You do not need to explain why it is efficient.
4. (20 pts) The **StudySession** interface is defined as follows:

**public interface StudySession {**

**public int duration(int topics, int timePerTopic);**

**}**

1. Using an anonymous inner class, initialize the variable s1 with an object that implements the StudySession interface and defines the duration of a study session as the product of **topics** by **timePerTopic** plus **10**. For example, calling s1.duration(5, 4) will return 30.

**StudySession s1 =**

1. Using a lambda expression, initialize the variable s2 with an object that implements the StudySession interface and defines the duration of a study session as the product of **topics** by **timePerTopic**. For example, calling s2.duration(5, 4) will return 20.

**StudySession s2 =**

1. Using a lambda expression, initialize the variable s3 with an object that implements the StudySession interface and defines the duration of a study session as 800 no matter what values are provided for **topics** and **timePerTopic**. For example, calling s3.duration(5, 4) will return 800. **Simplify the lambda expression as much as possible.**

**StudySession s3 =**

**Problem #3 (Linear Data Structures)**

Use the following classes to implement the methods below. Notice that in addition to the head reference, the list has a tail reference that points to the last node in the list. You may not add any instance variables nor 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, thread;**

**private Node(T data) {**

**this.data = data;**

**next = thread = null;**

**}**

**}**

**private Node head, tail;**

**public LinkedList() {**

**head = tail = null;**

**}**

**public void setThread(boolean pointToPrev) { /\* YOU MUST IMPLEMENT \*/ }**

**public Map<T, Integer> removeInstances(boolean sorted, T lowerBound) { /\* YOU MUST IMPLEMENT \*/ }**

**}**

1. Provide a **RECURSIVE** implementation for the **setThread** method. For this problem:
   1. If **pointToPrev** is true, the method initializes the **thread** node reference of each node to the node that precedes it in the list. If the node is the first one, **thread** will be set to null. After the **thread** reference has been set, we will have a doubly-linked list.
   2. If **pointToPrev** is false, the method initializes the **thread** node reference of every node (except the first one) to point to the first node in the list. The **thread** reference of the first node will be set to null.
   3. No processing will take place if the list is empty.
   4. Notice that **thread** has nothing to do with Java threads.
   5. **You will lose significant credit if you use any iteration statement (while, do while, for). Your solution must be recursive.**
   6. Notice the list is not a sorted list.
2. Provide a **NON-RECURSIVE** implementation for the **removeInstances** method. For this problem:
   1. The method removes from the list all instances that are greater than or equal to lowerBound. Notice the list can have duplicates.
   2. The data associated with the nodes must be inserted into a map so we can keep track of how many instances of each element found were removed.
   3. Notice the list is not a sorted list.
   4. An empty map must be returned if the list is empty.
   5. If the sorted parameter is true, the method will return a TreeMap; otherwise it will return a HashMap.
   6. **You will lose significant credit if your solution is recursive.**

One the next page we provided a driver that illustrates the functionality associated with the methods you need to implement. You can ignore it if you know what to implement. Notice the driver relies on methods you do not need to implement. You may find the following map methods helpful:

|  |  |
| --- | --- |
| V **get**(Object key) | Set<K> **keySet**() |
| V **put**(K key,V value) | boolean **isEmpty**() |
| boolean **containsKey**(Object key) | int **size**() |

|  |  |
| --- | --- |
| **Driver**  **LinkedList<String> list = new LinkedList<String>();**  **list.add("F").add("A").add("M").add("C").add("M").add("W");**  **System.out.println("list: " + list);**  **list.setThread(true);**  **System.out.print("After threading: ");**  **/\* prints list starting at last element, following thread \*/**  **list.printListThread();**  **System.out.println(" list: " + list);**  **Map<String, Integer> map = list.removeInstances(true, "M");**  **System.out.println("list after removing: " + list);**  **System.out.println("Map: " + map);** | **Output**  **list: W M C M A F**  **After threading: F A M C M W list: W M C M A F**  **list after removing: C A F**  **Map: {M=2, W=1}** |

**PAGE FOR PREVIOUS PROBLEM**

**PAGE FOR PREVIOUS PROBLEM**

**HONORS STUDENTS (THERE IS A QUESTION A THE END OF THE EXAM)**

**HONORS**

**Instructions**

Students in the Honor’s section must answer this question and only students in the Honor’s section will receive credit for answering it.

**Questions**

1. When running the implementation of an algorithm, the following running times were found:

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

47 10

94 21

188 42

What is the Big O of the algorithm?

1. When running the implementation of an algorithm, the following running times were found:

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

47 10

94 17

188 24

What is the Big O of the algorithm?

1. How can we show the best way to grow an array is by doubling its size? Circle all that apply.
   1. By using amortized analysis
   2. By using best case analysis
   3. By using a recursive solution
   4. By experimenting with an array
2. What is the difference between a static initialization block and a non-static initialization block?