|  |  |
| --- | --- |
|  | **University of Maryland College Park** |
| **Dept of Computer Science** |
| **CMSC132 Spring 2013** |
| **Midterm II** |

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

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

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

**Lab TA (Circle One):**

**Adil (0101, 11am), Adil (0102, 12pm) Beth (0103, 2pm), Beth (Honors, 1pm)**

**Souvik (0201, 2pm), Souvik (0202, 12pm)**

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 (225 pts for the honor’s section).
* The exam is a 50 minutes exam.
* Please use a pencil to complete the exam.
* **WRITE NEATLY**.

***Grader Use Only***

|  |  |  |  |
| --- | --- | --- | --- |
| #1 | Algorithmic Complexity | (40) |  |
| #2 | Generics | (20) |  |
| #3 | Hashing | (20) |  |
| #4 | Linear Data Structures | (120) |  |
| #5 | Honors Question | (25) |  |
| **Total** | Total | (200/225) |  |

**Problem #1 (40 pts) Algorithmic Complexity**

1. (15 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 = n; i <= n + 1; i++) {**

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

**}**

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

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

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

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

**}**

**}**

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

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

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

**}**

1. (8 pts) Give the asymptotic bound of the following functions:

1. log(**n**) + **n** f(**n**) = O( )
2. **n**2 + 4**n - n**2  f(**n**) = O( )
3. (5 pts) List the following big-O expressions in order of asymptotic complexity (lowest complexity first).

O(**n** log(**n**)) O(**n2**) O(**n**) O(1)

1. (6 pts)
   1. **True or False** 🡪 Critical sections are never found outside loops.
   2. **True or False** 🡪 Worst case analysis is more useful than best case analysis.
2. (6 pts) Give the complexity of an algorithm for problem size **n** whose running time:
   1. Increases by a constant when **n** doubles O( )
   2. Doubles when **n** doubles O( )

**Problem #2 (20 pts) Generics**

1. (4 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. (16 pts) Make the following class generic so that it can deal with an arbitrary class rather than only Strings. Feel free to cross out parts of the following code.

**public class Col {**

**private ArrayList<String> c;**

**public String get() { return c.remove(0); }**

**public void insert(String value) { c.add(value); }**

**}**

**Problem #3 (20 pts) Hashing**

1. (3 pts) **True or False** 🡪 From the Java Hash Code contract we can deduce that two objects that have the same hashCode() value must be equals according to the equals method.
2. (3 pts) By using hashing the best performance we can have for a search operation is **O( )**.
3. (3 pts) **True or False** 🡪 The Java Hash Code Contract is **not** violated when two objects that are considered the same (according to the equals method) have different hashCode() values.
4. (3 pts) **True or False** 🡪 A collision occurs when two entries are assigned to the same hash table entry.
5. (8 pts) Define a possible hashCode() method for the following class.

**public class Route {**

**private int id;**

**private String description;**

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

**if (obj == this)**

**return true;**

**if (!(obj instanceof Route))**

**return false;**

**return id == ((Route)obj).id;**

**}**

**public int hashCode() {**

**// define**

}

}

**Problem #4 (120 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 T getMax() {**

**if (head == null) {**

**return null;**

**} else {**

**return getMaxAux(head);**

**}**

**}**

**public T getMaxAux(Node headAux) {**

**throw new UnsupportedOperationException("You must implement");**

**}**

**public void moveToFrontFirstInstanceOf(T target) {**

**throw new UnsupportedOperationException("You must implement");**

**}**

**}**

1. Implement the auxiliary method **getMaxAux** that supports the computation of the maximum value in the list. This method is called by getMax. For this problem:
   1. You must implement a **RECURSIVE** solution, otherwise you will not get credit.
   2. Assume the list is not sorted.
   3. You may not add any other methods.
   4. Compare elements using compareTo.
   5. You may not modify getMax.
2. Implement the method **moveToFrontFirstInstanceOf** that moves to the front of the list the first instance that has a data component that corresponds to the target parameter. For example, if the list has the elements **10, 7, 40, 50** and we move to the front **40**, the new list will be **40, 10, 7, 50**. 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.
   4. You may not create any nodes.

**PAGE FOR PREVIOUS PROBLEM**

**NOTE: HONOR SECTION QUESTION CAN BE FOUND AT THE END.**

**EXTRA PAGE IN CASE YOU NEED IT**

**Problem #5 (25 pts) Honor’s Section Only**

**Notice: Only students in the honor’s section will receive credit for this problem.**

Using the **LinkedList** class defined in Problem #4, implement a recursive method called **reverseList** that reverses the elements in the list. For example, if we have a list with the values **5, 3, 10** and we call **reverseList,** the new list will be **10, 3, 5**. For this problem:

1. You must implement a **RECURSIVE** solution otherwise you will not get credit.
2. Assume the list is not sorted.
3. You may only add one auxiliary method.
4. You may not create any nodes.
5. Your code must be efficient and as short as possible, otherwise you will lose credit.

**public void reverseList() {**