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

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

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

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

**Lab TA (Circle One):**

|  |  |
| --- | --- |
| **0401, 3 pm, Melika Abolhassani** | **0403, 10 am, Qian Wu** |
| **0402, 4 pm, Melika Abolhassani** | **0404, 11 am, Qian Wu** |

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.
* There are 5 problems in the exam.
* The code you implement must be efficient.
* WRITE NEATLY.
* **You don’t need to use meaningful variable names; however, we expect good indentation.**

***Grader Use Only***

|  |  |  |  |
| --- | --- | --- | --- |
| #1 | Algorithmic Complexity | (15) |  |
| #2 | Correctness | (30) |  |
| #3 | Miscellaneous | (45) |  |
| #4 | Linear Data Structures | (55) |  |
| #5 | Trees | (55) |  |
| **Total** | Total | (200) |  |

**Problem #1 Algorithmic Complexity**

1. (10 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:

1. (5 pts)

**for (i = 10; i <= 10; i++) {**

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

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

**}**

**}**

**for (j = n / 2; j < n; j++) {**

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

**}**

**Time 🡪 (n - 2) + (n – n / 2) 🡪 3n/2 – 2**

**Big-O 🡪** O(n)

1. (5 pts)

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

**j = 1;**

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

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

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

**}**

**j++;**

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

**}**

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

**Time 🡪 1 + n2(log(n) + 1) + 1**

**Big-O 🡪 n2 (log(n))**

1. (5 pts) List the following big-O expressions in order of asymptotic complexity (lowest complexity first).

O(**n**log(**n**)) O(log(**n**)) O(**kn**) O(**nk**) O(**n!**)

**Answer:** O(log(**n**)) O(**n**log(**n**)) O(**nk**) O(**kn**) O(**n!**)

**Problem #2 Correctness**

The following code compiles. Identify and correct any problems with the code below or any unnecessary code. Verify the class satisfies the Java Hash Code Contract and make sure it has the best hashCode method possible. Feel free to cross out / edit the code. If there are no problems with the code write NONE.

**public class ShoppingCart {**

**private int id, numberOfItems, twoDollarsItems;**

**private double total;**

**private String company;**

**public ShoppingCart(int id, String company) {**

**total = 0;**

**twoDollarsItems = numberOfItems = 0;**

**this.company = new String(company);**

**}**

**public void addItem(double price) {**

**if (price == 2.0) {**

**twoDollarsItems++;**

**}**

**numberOfItems++;**

**}**

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

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

**return false;**

**}**

**if (obj == null) {**

**return false;**

**}**

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

**return false;**

**}**

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

**}**

**public int hashCode() {**

**return (int)total + id;**

**}**

**public String toString() {**

**String answer = id + " " + company + " " + total;**

**answer += numberOfItems + " " + twoDollarsItems;**

**return answer;**

**}**

**}**

**Answer:**

Missing this.id = id  
Unnecessary copy of string (new String(company))

We should not compare floating points using == (price == 2.0) above

equals method

obj == this is incorrect (must return true)

obj == null is unnecessary  
             Parameter must be object

hashCode method

hashCode method is correct (penalize if the indicate is incorrect)

hashCode method is not the best; it should return id

**Problem #3 Miscellaneous**

1. (3 pts) Formal methods is one of the software process models presented in class; name two additional ones presented in class.

**Answer:** Waterfall model and Iterative development

1. (3 pts) A compiler implementation has 8 modules each completing a particular task of the compilation process. Each module receives a tree and produces a tree that the next module can use. Which architecture style seen in class is associated with this system?

**Answer:** Pipes and filters

1. (3 pts) What is the fundamental difference between clear box and black box testing?

**Answer:** In clear box testing you have access to (can examine) the code.

1. (3 pts) What is flow path?
   1. The path the waterfall model defines for system design.
   2. Unique execution sequence through a program.
   3. The architecture style used by the iterative model.
   4. Common execution sequence through a program.
   5. None of the above.

**Answer:** b.

1. (3 pts) A program has 89% code coverage. That means:
   1. The program has 11% of code that is incorrect.
   2. The program has 11% of code that is correct.
   3. The program has 11% of code that was tested only once.
   4. The program has 89% of code that might have been tested and 11% that has been tested.
   5. None of the above.

**Answer:** e.

1. (15 pts) Modify (feel free to cross out and edit) the following **Schedule** class so it becomes a Generic class where the **String** class can be replaced with any class.

**public class Schedule {**

**String data[] = new String[4];**

**String compare(String x) {**

**if (x.equals(data[0])) {**

**return null;**

**}**

**return x;**

**}**

**}**

**Answer:**

**public class Schedule<T> {**

**T data[] = (T[]) new Object[4];**

**T compare(T x) {**

**if (x.equals(data[0])) {**

**return null;**

**}**

**return x;**

**}**

**}**

1. (3 pts) Is the following assignment allowed? If it is allowed, can we perform any kind of operations using c?

**ArrayList<?> c = new ArrayList<String>();**

**Answer:**

Assignment is allowed, but we cannot perform any kind of operation (e.g., add is not allowed).

1. (3 pts) Draw a tree representation for the following heap (specified as an array): 10 24 51 72

**Answer:**

**10**

**24 51**

**72**

1. (3 pts) What kind of exception require us to declare a catch block or to use throws?
   1. An unchecked exception
   2. A checked exception
   3. All exceptions
   4. None of the above

**Answer: b.**

1. (3 pts) A finally block is executed:
   1. Only when the exception occurs
   2. When no exception takes place
   3. Always
   4. When the exception is an illegal argument exception
   5. None of the above.

**Answer: c.**

1. (3 pts) The following assignment:

ArrayList<Object> m = new ArrayList<String>();

* 1. Will not compile.
  2. It will compile, but we cannot add any elements to m.
  3. It will compile and we can add String objects to m.
  4. None of the above.

**Answer: a.**

**Problem #4 Linear Data Structures**

Use the following classes to implement the method 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 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;**

**public LinkedList() { head = null; }**

**public boolean deleteEvenNumberedEntries(HashMap<Integer, T> map) { /\* IMPLEMENT \*/ }**

**}**

Implement a **NON-RECURSIVE** method called **deleteEvenNumberedEntries** that removes even numbered entries (nodes) from the list. In addition, the method will place the **data** element of each node removed into a HashMap where the key will be the position of the node in the list and the value the actual data. The method will return true if any node was removed and false otherwise. Notice the list can be empty. Here is an example:

**Original List**

Mike

Luisa

Peter

Mary

John

**List after calling deleteEvenNumberedEntries**

Mike

Peter

John

**Entries in the map**

{2=Luisa, 4=Mary}

Information about Map methods is included at the end. You can assume the first node is an odd node (it is associated with number 1). **Do not use recursion. If you do, you will lose credit.**

**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.
* int **size**() - Returns the number of key-value mappings in this map.

**IMPLEMENT YOUR METHOD ON THE NEXT PAGE**

**One Possible Answer:**

**public boolean deleteEvenNumberedEntries(HashMap<Integer, T> map) {**

**if (head == null || head.next == null) {**

**return false;**

**}**

**Node prev = null, curr = head;**

**int position = 1;**

**while (curr != null) {**

**if (position % 2 == 0) {**

**prev.next = curr.next;**

**map.put(position, curr.data);**

**} else {**

**prev = curr;**

**}**

**curr = curr.next;**

**position++;**

**}**

**return true;**

**}**

**Problem #5 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. **The solution to this problem must be recursive. If you use any iteration statement (e.g., while, do while, for loop) you will get 0 credit.**

**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 int getPathKeys(ArrayList<K> list, K target) { /\* IMPLEMENT \*/ }**

}

Write a **RECURSIVE** method called **getPathKeys** that adds to an ArrayList the key values of nodes on the path that leads to the node with a key that corresponds to **target** in the tree. The ArrayList will include the target key. If the key does not exist in the tree, the path will include all the keys that were visited (until you determine the target key was not in the tree). The method returns the number of keys added to the list. You can use the ArrayList size() method to return / compute the number of keys added. The following is an example (feel free to ignore if you know what to implement). Notice the example relies on the add method that you do need to implement.

|  |  |
| --- | --- |
| **Driver**  **public static void main(String[] args) {**  **BinarySearchTree<Integer, String> tree = new BinarySearchTree<Integer, String>();**  **tree.add(40, "Forty");**  **tree.add(20, "Twenty");**  **tree.add(60, "Sixty");**  **tree.add(10, "Ten");**  **tree.add(30, "Thirty");**  **tree.add(50, "Fifty");**  **tree.add(70, "Seventy");**    **ArrayList<Integer> list = new ArrayList<Integer>();**  **System.out.println("Keys added: " + tree.getPathKeys(list, 30));**  **System.out.println(list);**  **}** | **Output**  **Keys added: 3**  **[40, 20, 30]** |

**IMPLEMENT YOUR METHOD ON THE NEXT PAGE**

**Answer:**

**public int getPathKeys(ArrayList<K> list, K target) {**

**getPathKeysAux(root, list, target);**

**return list.size();**

**}**

**public void getPathKeysAux(Node rootAux, ArrayList<K> list, K target) {**

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

**list.add(rootAux.key);**

**int comparison = target.compareTo(rootAux.key);**

**if (comparison == 0) {**

**return;**

**} else if (comparison < 0) {**

**getPathKeysAux(rootAux.left, list, target);**

**} else {**

**getPathKeysAux(rootAux.right, list, target);**

**}**

**}**

**}**