Test 2 PRACTICE: CompSci 100
Michael Hewner
4/14/2012

Name: _______________________________________________________________

Netid (please print clearly): ______________

Honor code acknowledgement (signature): ________________________________

<table>
<thead>
<tr>
<th>Question</th>
<th>value</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question 1: Trees (No Coding)</td>
<td>3</td>
<td>points</td>
</tr>
<tr>
<td>Question 2: Big O &amp; Recurrence Relations</td>
<td>4</td>
<td>points</td>
</tr>
<tr>
<td>Question 3: Very Hard Problems</td>
<td>1</td>
<td>points</td>
</tr>
<tr>
<td>Question 4: Linked List 1</td>
<td>5</td>
<td>points</td>
</tr>
<tr>
<td>Question 5: Linked List 2</td>
<td>5</td>
<td>points</td>
</tr>
<tr>
<td>Question 6: Recursive Backtracking 1</td>
<td>5</td>
<td>points</td>
</tr>
<tr>
<td>Question 7: Recursive Backtracking 2</td>
<td>5</td>
<td>points</td>
</tr>
<tr>
<td>Question 8: Trees (Coding)</td>
<td>5</td>
<td>points</td>
</tr>
<tr>
<td>Question 9: Stacks, Queues, Priority Queues 1</td>
<td>5</td>
<td>points</td>
</tr>
<tr>
<td>Question 10: Stacks, Queues, Priority Queues 2</td>
<td>5</td>
<td>points</td>
</tr>
<tr>
<td>TOTAL</td>
<td>43</td>
<td>points</td>
</tr>
</tbody>
</table>
**Question 1:** Trees (No Coding) [4 points]

Imagine you added the following integers to a binary search tree: 1 19 3 6 5. What does the resultant tree look like?

Imagine you added the following integers to a heap: 1 19 3 6 5. What does the resultant tree look like?

Given the following binary tree:

```
     10
    /  \
   3    20
  / \
/    \
4    7
\    /
 \   19
  \ 
   8
```

Write its postorder traversal:

Write its inorder traversal:
Question 2: Big O & Recurrence Relations [3 points]

What is the Big O of the runtime of the following functions:

1.

```java
public int computeData(String[] strings, int start) {
    if(start >= strings.length)
        return 0;
    for(int i=0; i< strings.length; i++) {
        if(strings.charAt(0) == 'A')
            return 7;
    }
    return computeData(strings, start + 1)
}
```

Big O (n is the length of strings): __________________________________________________________________________________

2.

```java
public boolean findSeven(LinkedList<Integer> intData) {
    if(intData.isEmpty())
        return false;
    int first = intData.remove(0);
    if(first == 7)
        return true;
    return findSeven(intData);
}
```

Big O (n is the length of intData): __________________________________________________________________________________

3.

```java
// height is a recursive function that runs in O(n) time where n is the number of tree nodes
public boolean isHB(TreeNode root) {
    if(root == null)
        return tree;
    int heightL = height(root.left);
    int heightR = height(root.right);
    boolean leftHB = isHB(root.left);
    boolean rightHB = isHB(root.right);
    if(Math.abs(heightL - height) > 1) return false;
    return leftHB && rightHB;
}
```

Assume the tree is height balanced

Big O (n is number of nodes in the tree): __________________________________________________________________________________

4.

Same code as #3, do NOT assume the tree is height balanced

Big O (n is number of nodes in the tree): __________________________________________________________________________________
Question 2: Big O & Recurrence Relations (continues)

5.
Finding to see if a particular node exists in a non-height balanced binary search tree:

Big O (n is number of nodes in the tree): ___________________________

6.
Removing a particular value (not the lowest value) from a Priority Queue implemented using a heap.

Big O (n is number of nodes in the heap): ___________________________
Question 3: Very Hard Problems [1 point]

A problem whose solution can be verified in polynomial time can be called a member of this complexity class:

1. P
2. NP
3. NP-Complete
4. EXP
5. Incomputable

Say while working on an APT, you happen to come up with a solution for an NP-Complete problem like satisfiability. What does that mean?

1. P = NP
2. P does not equal NP
3. EXP equals NP
4. None of these things
Write a function `copyToDoublyLinkedList` that takes a parameter which is the head of a singly linked list and returns the head of a doubly linked list which is a copy of the original list. Use the following node classes:

```java
public class ListNode {
    public int value;
    public ListNode next;
    public ListNode(int v, ListNode n) {
        value = v;
        next = n;
    }
}

public class DListNode {
    public int value;
    public DListNode next;
    public DListNode prev;
    public DListNode(int v, DListNode n, DListNode p) {
        value = v;
        next = n;
        prev = p;
    }
}

public DListNode copyToDoublyLinkedList(ListNode head) {
    // Implementation goes here
}
```
Question 5: Linked List 2

Write a function removeDuplicates that takes as a parameter a node that is the head of a linked list. removeDuplicates removes all but the first occurrence of each value from the linked list. The list should remain in the same order otherwise. Use the following ListNode class:

```java
public class ListNode {
    public int value;
    public ListNode next;

    public ListNode(int v, ListNode n) {
        value = v;
        next = n;
    }
}
```

Examples:
- [1,2,3] becomes [1,2,3]
- [1,2,2,3,1,3] becomes [1,2,3]
- null (empty list) becomes null

```java
public void removeDuplicates(ListNode head) {
```
Question 6: Recursive Backtracking 1

Don't Go Over is a game played with a deck of numbered cards. The first player selects one card and adds its value to his/her score, then passes the remaining cards to the second player. The second player selects a card and adds its value to his/her score then passes it back to the first player. This continues until the deck runs out of cards. Once all the cards are distributed, any player that has their score higher than scoreLimit has their score reduced to 0. Then the player with the highest score wins.

Write a function whoWins that given an ArrayList representing a deck of cards and a scoreLimit, determines if player 1 or 2 wins the game (assuming both sides play perfectly...they try to win, and if they can't win try to tie). If player 1 wins, whoWins should return 1. If player 2 wins, whoWins should return 2. If they will tie, return 0.

*Hint:* You might want to write a helper function public void whoWins(ArrayList<Integer> cards, int scoreLimit, int p1score, int p2score)

*Examples:*
cards [10,10,3,2] scorelimit 19 returns 1
cards [10,11,12,13] scorelimit 19 returns 0
cards [10,9,8] scorelimit 17 returns 2

public void whoWins(ArrayList<Integer> cards, int scoreLimit) {

Question 7: Recursive Backtracking 2

The function `allDotsReachable` takes a 2 dimensional array of the characters '.' and 'X'. You start in the upper left hand corner (0,0) and are allowed to move north south east or west - you can't move through Xs. `allDotsReachable` should return true if every dot on the board is reachable from the starting position, false otherwise. You can modify the input array if you wish.

*Hint:* you might benefit from a helper function for this one.

*Examples:*

```
. . . . .
. . . . .
. . . . .
. . . . .
false (lower right not reachable)
```

```
. . . . .
. . . . .
. . X . .
. . . . .
false (right half not reachable)
```

```
. . . . .
. . . . .
. . . X X
. . . X X
true
```

```java
public boolean allDotsReachable(char[][] map) {
```
**Question 8: Trees (Coding)**

Write a function `evenPathToLeaf` that takes the root of a binary tree as a parameter, and returns `true` if there is a path from the root to a leaf that only consists of even elements. Use the following `TreeNode` class:

```java
public class TreeNode {
    public int value;
    public TreeNode left;
    public TreeNode right;

    public TreeNode(int v){
        value = v;
    }
}
```

**Examples:**

```
null  true  false  true  false  false
```

```java
public boolean evenPathToLeaf(TreeNode root) {
```

Question 9: Stacks, Queues and Priority Queues

A word chain is a list of words such that the last letter of the previous word becomes the first letter of the next. So the string [dog, goat, time, eat] is a word chain. Write a function shortestChain that takes a start word and end word of a word chain and a list of allowed intermediate words. shortestChain should return the last intermediate word in the shortest word chain that connects the two words. You can assume that a chain will always be possible, given the words you're provided.

Examples:
shortestChain(dog, bark, [tab,groan,goat,normal,lab] returns tab (shortest chain is dog->goat->tab->bark)
shortestChain(good,luck,[dog,lake,earl]) returns earl

public String shortestChain(String start, String end, ArrayList<String> words) {
Imagine you have a binary tree that contains characters instead of integers. There are 3 kinds of nodes: 'E' nodes that are easy to pass through, 'D' nodes that you can pass through but are difficult, and 'G' nodes that represent your goal. Write a function findEasyPath to find the "easiest" path to a 'G' node, where easiest means passing through the minimum number of 'D' nodes. Your function should return the 'G' node that is destination of the easiest path. Return null if the tree has no G nodes. Use the following TreeNode class - note that it implements Comparable in a way that should make your life easier:

```java
class TreeNode implements Comparable<TreeNode> {
    char value;
    TreeNode left;
    TreeNode right;

    // Sorts G nodes first, then E nodes, then D nodes
    public int compareTo(TreeNode o) {
        return o.value - value;
    }
}
```

Example:

```
the highlighted node is the one that should be returned
returns null - no G nodes
```

```java
public TreeNode findEasyPath(TreeNode root) {
    // Implementation goes here
}
```