# 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></td>
<td>3 points</td>
</tr>
<tr>
<td>Question 2: Big O &amp; Recurrence Relations</td>
<td></td>
<td>4 points</td>
</tr>
<tr>
<td>Question 3: Very Hard Problems</td>
<td></td>
<td>1 point</td>
</tr>
<tr>
<td>Question 4: Linked List 1</td>
<td></td>
<td>5 points</td>
</tr>
<tr>
<td>Question 5: Linked List 2</td>
<td></td>
<td>5 points</td>
</tr>
<tr>
<td>Question 6: Recursive Backtracking 1</td>
<td></td>
<td>5 points</td>
</tr>
<tr>
<td>Question 7: Recursive Backtracking 2</td>
<td></td>
<td>5 points</td>
</tr>
<tr>
<td>Question 8: Trees (Coding)</td>
<td></td>
<td>5 points</td>
</tr>
<tr>
<td>Question 9: Stacks, Queues, Priority Queues 1</td>
<td></td>
<td>5 points</td>
</tr>
<tr>
<td>Question 10: Stacks, Queues, Priority Queues 2</td>
<td></td>
<td>5 points</td>
</tr>
<tr>
<td>TOTAL</td>
<td></td>
<td>43 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?

![Binary Search Tree Diagram]

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

![Heap Diagram]

Given the following binary tree:

![Binary Tree Diagram]

Write its postorder traversal:

4 8 7 3 19 20 10

Write its inorder traversal:

4 3 7 8 10 19 20 *corrected*
**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): O(n^2) ...because T(n-1) + O(n)
```

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): O(n) ... because T(n-1) + O(1)
```

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 true;
       int heightL = height(root.left);
       int heightR = height(root.right);
       boolean leftHB = isHB(root.left);
       boolean rightHB = isHB(root.right);
       if(Math.abs(heightL - heightR) > 1) return false;
       return leftHB && rightHB;
   }

   Assume the tree is height balanced

   Big O (n is number of nodes in the tree): O(n log n) ...because 2T(n/2) + O(n)
```

4. ```java
   Same code as #3, do NOT assume the tree is height balanced

   Big O (n is number of nodes in the tree): O(n^2) ...because T(n-1) + O(n)
```
**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): O(n)

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): O(n)
**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

This last one is maybe a little trickier than I intended it. What I had intended to say was that you found "a polynomial time solution" for an NP-Complete problem, in which case the answer would be 1. But I left off the "polynomial time" part of it, so the solution you found could be exponential which would not change anything.
Question 4: Linked List 1

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) {
    if (head == null) return null;
    ListNode current = head;
    DListNode doubleCur = new DListNode(head.value, null, null);
    DListNode doubleHead = doubleCur;
    while (current.next != null) {
        current = current.next;
        doubleCur.next = new DListNode(current.value, null, douleCur);
        doubleCur = doubleCur.next;
    }
    return doubleHead;
}
```
**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
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) {
    HashSet<Integer> seen = new HashSet<Integer>();
    if(head == null) return;
    seen.add(head.value);
    ListNode current = head;
    while(current.next != null) {
        if(seen.contains(current.next.value)) {
            current.next = current.next.next;
        } else {
            seen.add(current.next.value);
            current = current.next;
        }
    }
}
```
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 a 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 int 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 int whoWins(ArrayList<Integer> cards, int scoreLimit) {
    return whoWins(cards,scoreLimit,0,0);
}

public int whoWins(ArrayList<Integer> cards, int scoreLimit, int p1score, int p2score) {
    if(cards.isEmpty()) {
        if(p1score > scoreLimit) p1score = 0;
        if(p2score > scoreLimit) p2score = 0;
        System.out.printf("Final score: %d %d\n", p1score, p2score);
        if(p1score > p2score)
            return 1;
        if(p1score < p2score)
            return 2;
        return 0;
    }
    boolean foundTie = false;
    for(int i = 0; i < cards.size(); i++) {
        int cardValue = cards.remove(i);
        int winner = whoWins(cards,scoreLimit,p2score,p1score + cardValue);
        cards.add(i,cardValue);
        if(winner == 2) return 1;
        if(winner == 0) foundTie = true;
    }
    if(foundTie) return 0;
    return 2;
}

My opinion after fully solving this problem is that it's maybe a little too tricky to appear on an exam. Make sure you understand how the solution works though! There will be a similar (though easier) one on the actual midterm.
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:**

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

```
public boolean allDotsReachable(char[][] map) {
    markDotsReachableStartingAt(map, 0, 0);
    for (int r = 0; r < map.length; r++) {
        for (int c = 0; c < map[r].length; c++) {
            if (map[r][c] == '.') return false;
        }
    }
    return true;
}

public void markDotsReachableStartingAt(char[][] map, int r, int c) {
    if (r < 0 || r >= map.length) return;
    if (c < 0 || c >= map[0].length) return;
    if (map[r][c] == 'X') return;
    map[r][c] = 'X';
    markDotsReachableStartingAt(map, r - 1, c);
    markDotsReachableStartingAt(map, r + 1, c);
    markDotsReachableStartingAt(map, r, c - 1);
    markDotsReachableStartingAt(map, r, c + 1);
}
```
**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
class TreeNode {
    int value;
    TreeNode left, right;

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

**Examples:**

```
null  true  false  false  true  false  false
22    3     4      22    3     99     4
6     8     6      8     16    16
```

```java
public boolean evenPathToLeaf(TreeNode root) {
    if (root == null) return false;
    if (root.value % 2 == 1) return false;
    if (root.left == null && root.right == null) return true;
    if (evenPathToLeaf(root.left)) return true;
    if (evenPathToLeaf(root.right)) return true;
    return false;
}
```
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) {
    LinkedList<String> q = new LinkedList<String>();
    q.add(start);
    while(!q.isEmpty()) {
        String current = q.remove();
        char lastChar = current.charAt(current.length - 1);
        if(lastChar == end.charAt(0))
            return current;
        for(String s : words) {
            if(lastChar == s.charAt(0))
                q.add(s);
        }
    }
    //should never get here
    return null;
}
**Question 10:** Stacks, Queues and Priority Queues

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

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

**Example:**

```
// sorts G nodes first, then E nodes, then D nodes
```

```
the highlighted node is the one that should be returned
```

```
returns null - no G nodes
```

```java
public TreeNode findEasyPath(TreeNode root) {
    PriorityQueue<TreeNode> queue = new PriorityQueue<TreeNode>();
    LinkedList<TreeNode> dQueue = new LinkedList<TreeNode>();
    queue.add(root);
    while (!queue.isEmpty() || !dQueue.isEmpty()) {
        if (!queue.isEmpty()) {
            TreeNode current = queue.remove();
            if (current.value == 'G')
                return current;
            if (current.value == 'D') {
                dQueue.add(current);
            }
            if (current.value == 'E') {
                if (current.left != null)
                    queue.add(current.left);
                if (current.right != null)
                    queue.add(current.right);
            }
        } else {
            TreeNode current = dQueue.remove();
            if (current.left != null)
                queue.add(current.left);
            if (current.right != null)
                queue.add(current.right);
        }
    }
}
```
The above problem turned out to be more complex than I anticipated. I had a originally posted a much shorter priority queue solution, but Jeff Day correctly pointed out it was not correct. If you wrote a solution that looks like this:

```java
public TreeNode findEasyPath(TreeNode root) {
    PriorityQueue q = new PriorityQueue<TreeNode>();
    q.add(root);
    while(!q.isEmpty()) {
        TreeNode current = q.remove();
        if(current.value == 'G')
            return current;
        if(current.left != null)
            q.add(current.left);
        if(current.right != null)
            q.add(current.right);
    }
    return null;
}
```

You have the consolation of being as correct as your professor was. I wouldn't actually you to generate as much code as the true solution on the exam. Thanks Jeff for pointing out my mistake.