Test 2: CompSci 100

Michael Hewner 4/11/2012

Name: _____Netid (please print clearly): _____

Honor code acknowledgement (signature): _____

	value	grade	grader
Question 1: Trees (No Coding)	4 points		
Question 2: Big O & Recurrence Relations	3 points		
Question 3: Very Hard Problems	1 points		
Question 4: Linked List 1	5 points		
Question 5: Linked List 2	5 points		
Question 6: Recursive Backtracking 1	5 points		
Question 7: Recursive Backtracking 2	5 points		
Question 8: Trees (Coding)	5 points		
Question 9: Stacks, Queues, Priority Queues 1	5 points		
Question 10: Stacks, Queues, Priority Queues 2	5 points		
TOTAL	43 points		

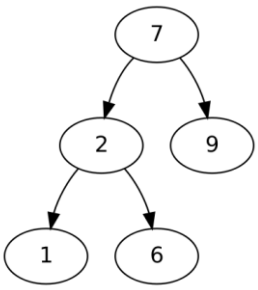
This exam contains 12 pages. Please check to ensure your copy has the right number of pages.

You have 75 minutes to complete this exam. Do not speed too much time on any one question. Do not start this exam until your professor gives you permission to do so. When time is called, stop immediately - otherwise your exam will get at 10% pentalty.

Please write as close to accurate java code as you can. You do not have to worry about import statements.

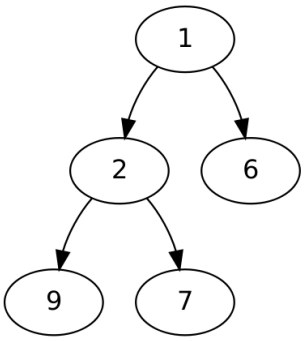
Question 1 [4 points]: Trees (No Coding)

Imagine you added the following integers to a binary search tree: 7 2 6 9 1. Assuming the tree uses the algorithm we discussed in class that that inserts a node at the appropriate place in the BST without moving any existing nodes, what does the resultant tree look like?



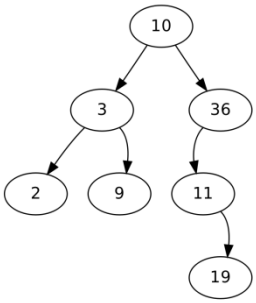
You would get half credit if you created a valid BST but not added in the appropriate order.

Imagine you added the following integers to a heap: 7 2 6 9 1. What does the resultant tree look like?



You would get half credit if you created a valid Heap but not added in the appropriate order.

Given the following binary tree:



Write its preorder traversal:

10 3 2 9 36 11 19

Write its inorder traversal:

2 3 9 10 11 19 36

Question 2 [3 points]: Big O and Recurrence Relations

Some useful recurrence relations:
$T(n) = T(n/2) + O(1) \rightarrow O(\log n)$
$T(n) = T(n-1) + O(1) \rightarrow O(n)$
$T(n) = 2T(n/2) + O(1) \rightarrow O(n)$
$T(n) = 2T(n/2) + O(n) \rightarrow O(n \log n)$
$T(n) = T(n-1) + O(n) \rightarrow O(n^2)$

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

1.

```
public int getValue(int value) {
	if(value <= 1) return 0;
	return value + getValue(value/2);
}
```

Big O (n is the size of value): **O(log n)**

2.

```
public void divide (LinkedList<String> bigList, LinkedList<LinkedList<String>> outputList) {
	if(bigList.size <= 1)
		return;
	LinkedList<String> list1 = new LinkedList<String>();
	LinkedList<String> list2 = new LinkedList<String>();
	for(int i = 0; i < bigList.size(); i++) {
		if(i % 2 == 0) {
			list1.add(bigList.get(i));
		} else {
			list2.add(bigList.get(i));
		}
	}
	outputList.add(list1);
	outputList.add(list2);
	divide(list1,outputList);
	divide(list2,outputList);
}
```

Big O (n is the length of intData): **O(n log n)**

Question2 [3 points] (continued): Big O and Recurrence Relations

3.

```
// height is a recursive function that runs in O(n) time where n is the number of tree nodes
public int countVals(TreeNode root, int val) {
	if(root == null)
		return 0;
	if(root < root.val)
		return countVals(root.left);
	if(root == root.val)
		return 1 + countVals(root.right);
	if(root > root.val)
		return countVals(root.right);
}
```

Assume the tree is height balanced

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

4.

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

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

5.

Removing a particular value from a height balanced binary search tree:

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

6.

Removing the lowest value from a Priority Queue implemented using a heap.

Big O (n is number of nodes in the heap): **O(log n)…assuming that you also pay the cost of rebalancing the heap. We also accepted O(1) if you don't count the cost of rebalancing.**

Question 3 [1 points]: Very Hard Problems

The "halting problem" is the famous problem of writing a java program that can determine if another java program will enter an infinite loop. The halting problem is:

1. In P

2. In NP

3. NP-Complete

4. In EXP

5. Incomputable

Say while working on an APT problem, you notice that there's a way to convert a known NP complete problem (like Satisfiably) to your problem in polynomial time. What does this mean?

1. P = NP

2. P does not equal NP

3. Your problem is NP-Complete

4. You must be mistaken; such a conversion could not exist if the problem is NP-Complete

5. None of these things

Question 4 [5 points]: Linked List 1

Write a function insertAtEnd which takes as a parameter the head of a non-empty doubly linked list and an element to add. insertAtEnd adds the element at the end of the doubly linked list. Use the following list node class:

```
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;
   }
}
```

Examples:
[1,2,3] and 4 becomes [1,2,3,4]

```
public void insertAtEnd(DListNode head, int elementToAdd) {
```

```
     public void insertAtEnd(DListNode head, int elementToAdd) {
         DListNode cur = head;
         while(cur.next != null) {
             cur = cur.next;
         }
         cur.next = new DListNode(elementToAdd, null, cur);
     }
```

Question 5 [5 points]: Linked List 2

Write a function removeEveryOtherElement that takes as a parameter a node that is the head of a linked list. removeEveryOtherElement keeps the head, removes the 2nd element, keeps the 3rd element, removes the 4th element, etc..
Use the following ListNode class:

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

Examples:
[1,2,3,4,5,6,7] becomes [1,3,5,7]
[2,20,200,2000] becomes [2,200]
null (empty list) does nothing

```
public void removeEveryOtherElement(ListNode head) {
```

```
     public void removeEveryOtherElement(ListNode head) {
         ListNode cur = head;
         while(cur != null) {
             if(cur.next != null) {
                 cur.next = cur.next.next;
             } else {
                 cur.next = null;
             }
             cur = cur.next;
         }
     }
```

Question 6 [5 points]: Recursive Backtracking 1

PilesList is a game played with 2 players. The game begins with several piles of tokens. Each turn, a player must remove 1 or 2 tokens from 1 pile. If a pile contains 0 elements, it can no longer be used. A player loses if they can't move (because all the piles are 0). Write a function whoWinsPilesList that takes an arrayList of piles and determines if player 1 will win or player 2 will win (assuming both sizes play perfectly).

Examples:
[2,0,0] returns 1 because player 1 can just grab the 2
[1,1] returns 2 because player 1 must remove one pile and player 2 will remove the other
[] returns 2 (player one can't move)
[2,1] returns 1 because player 1 can subtract 1 from the 1st pile

```
public int whoWinsPilesList(ArrayList<Integer> piles) {
```

```
public int whoWinsPilesList(ArrayList<Integer> piles) {
    for(int i = 0; i < piles.size(); i++) {
        int val = piles.get(i);
        if(val >= 1) {
            piles.set(i, val - 1);
            int result = whoWinsPilesList(piles);
            piles.set(i, val);
            if(result == 2) {
                return 1;
            }
        }
        if(val >= 2) {
            piles.set(i, val - 2);
            int result = whoWinsPilesList(piles);
            piles.set(i, val);
            if(result == 2) {
                return 1;
            }
        }
    }
    return 2;

}
```

Question 7 [5 points]: Recursive Backtracking 2

The function escapeInDistance takes a 2 dimensional array of the characters '.' and 'X' representing a map. You start at a given row and column and are allowed to move north south east or west - you can't move through Xs. Your function should return true if it is possible to escape (that is, move off any edge of the map) moving through exactly the given number of squares. You are not allowed to visit any square twice.

Examples:

(0,0) is upper left corner

```
. .
. .
```

for (0,0) and 1 - true (you visit the start and move off the map)
for (0,0) and 4 - true (visit every square then move off the map)
for (0,0) and 5 - false (there are not 5 squares to visit)

```
X X . X X
X . . . .
X X X X X
```

for (1,1) and 3 - true
for (1,1) and 4 - true
for (1,1) and anything else - false

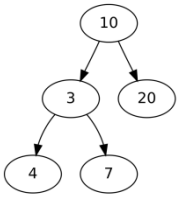
```
//to access a particular square do map[rowNum][columnNum].
public boolean escapeInDistance(char[][] map, int row, int col, int squares) {

     public boolean escapeInDistance(char[][] map, int row, int col, int squares) {

         if(row < 0 || row >= map.length)
                 return squares == 0;
         if(col < 0 || col >= map[0].length)
                 return squares == 0;
         if(squares == 0) return false;

         char current = map[row][col];
         if(current == 'X') return false;

         map[row][col] = 'X';

         if(escapeInDistance(map,row + 1,col,squares-1)) return true;
         if(escapeInDistance(map,row - 1,col,squares-1)) return true;
         if(escapeInDistance(map,row,col + 1,squares-1)) return true;
         if(escapeInDistance(map,row,col - 1,squares-1)) return true;

         map[row][col] = '.';
         return false;
     }
```

Question 8 [5 points]: Trees (Coding)

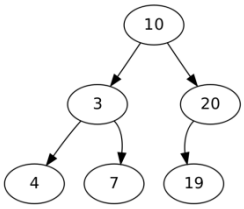
Write a function hasPartialLeaf that takes the root of a binary search tree as a parameter and returns true if there is node in the tree that is a partial leaf. A partial leaf is node that has a left subtree but no right subtree (or vice versa). Use the following TreeNode class:

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

Examples:



false



true

false

null

false

```
public boolean hasPartialLeaf(TreeNode root) {
```

```
     public boolean hasPartialLeaf(TreeNode root) {
         if(root == null) return false;
         if(root.left == null && root.right != null) return true;
         if(root.right == null && root.left != null) return true;
         return hasPartialLeaf(root.left) || hasPartialLeaf(root.right);
     }
```

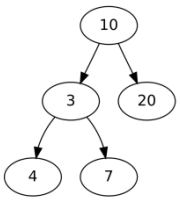
Question 9 [5 points]: Stacks, Queues, and Priority Queues 1

Binary trees have another kind of traversal called a "level order traversal". Basically, a level order traversal visits the elements of a tree in order of their distance from the root (i.e. first the root, then the second "level" of the tree, then the third "level" of the tree, etc). Write a function that takes the root of a tree and returns a level order traversal of that tree as an ArrayList of Strings. *Hint*: Although you're allowed to use any approach you like, it is simpler to use a stack/queue/priority queue approach to solve this problem than to use a recursive functions on a binary tree.

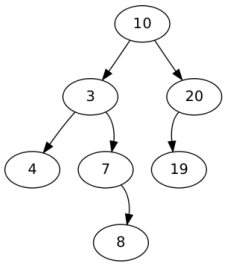
Use the following TreeNode class:

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

Examples:



[10,3,20,4,7]



[10,3,20,4,7,19,8]

[7]

null

[]

```
public ArrayList<Integer> levelOrderTraversal(TreeNode root) {
```

```
    public ArrayList<Integer> levelOrderTraversal(TreeNode root) {
        LinkedList<TreeNode> queue = new LinkedList<TreeNode>();
        queue.add(root);
        ArrayList<Integer> result = new ArrayList<Integer>();
        while(!queue.isEmpty()) {
            TreeNode current = queue.remove();
            result.add(current.value);
            if(current.left != null) queue.add(current.left);
            if(current.right != null) queue.add(current.right);
        }
        return result;
    }
```

Question 10 [5 points]: Stacks, Queues, and Priority Queues 2

The function getMissingParens takes a String that contains only the following 4 characters: '(' ')' '[' ']'. It returns a String that is whatever characters are necessary to "finish" the given string such that all the parens match. Note that '(' only matches with ')' and '[' only matches with ']'. If it is impossible to finish the string such that all the parens match, the function should return null.

Examples:
"(" should return ")"
"([(" should return ")])"
"()([]" should return ")"
"()()()[]" should return ""
"([)" should return null

```
//note that we've built two useful functions for you
public boolean isOpening(char a) { return (a == '(') || (a == '['); }
public char matchingClose(char a) {
   if(a == '(') { return ')';  } else { return ']';  }
}
public String getMissingParens(String input) {
```

```
public  String getMissingParens(String input) {
    Stack<Character> s = new Stack<Character>();
    for(int i = 0; i < input.length(); i++) {
        if(isOpening(input.charAt(i))) {
            s.push(input.charAt(i));
        } else {
            if(s.isEmpty()) return null;
            if(matchingClose(s.pop()) != input.charAt(i))
                return null;
        }
    }
    String result = "";
    while(!s.isEmpty()) {
        result += matchingClose(s.pop());
    }
    return result;
}
```