Sample Test 2

Definitions
Some common recurrences and their solutions.

\[
\begin{align*}
T(n) &= T(n/2) + O(1) \quad O(\log n) \\
T(n) &= T(n/2) + O(n) \quad O(n) \\
T(n) &= 2T(n/2) + O(1) \quad O(n) \\
T(n) &= 2T(n/2) + O(n) \quad O(n \log n) \\
T(n) &= T(n-1) + O(1) \quad O(n) \\
T(n) &= T(n-1) + O(n) \quad O(n^2)
\end{align*}
\]

List Node

```java
public class Node {
    String info;
    Node next;
    Node(String s, Node link) {
        info = s;
        next = link;
    }
}
```

Node for Binary Trees

```java
public class TreeNode {
    String info;
    TreeNode left;
    TreeNode right;
    TreeNode parent;
    TreeNode (String s, TreeNode lt, TreeNode rt, TreeNode p)
    
    { 
        info = s;
        left = lt;
        right = rt;
        parent = p;
    }
}
```
PROBLEM 1:  (Short Ones)

A. Why should you use a HashMap rather than a TreeMap? Choose the best answer.
   I. To conserve memory.
   II. Because the performance of get and put is better
   III. Because the Java library implementation is optimized.
   IV. To guarantee that lookups (get) will be faster than inserts (put).
   V. Because it supports a fast sort implementation.

B. Give a primary reason to use a balanced BST instead of a:
   I. Sorted linked list
   II. Trie

C. Suppose that I take a sorted linked list of $N$ integers, and break it into a list of $M$ equal-sized sorted lists of integers (that is, put the first $N/M$ integers into the first list, the next $N/M$ into the second, etc.). What is the worst-case time for finding whether an integer $x$ is in anywhere in this list of lists? If for some fixed $N$, you can choose $M$, what $M$ should you choose for maximum lookup speed?

D. You are creating a boggle word search program and you want to find all valid words where the letters are adjacent. What data structure would be best suited to hold the dictionary (i.e. lexicon)?
   A. Queue
   B. Heap
   C. Trie
   D. Balanced Binary Tree
   E. Hash Table

F. In order to use the class Point containing fields $x$ and $y$ in a HashSet, you are considering multiple hash functions. Of these hash functions, which one would give the best performance in a HashSet? Assume that your points are likely to be between (0, 0) and (1280, 1024) (the size of the average computer monitor).
   A. public int hashCode () { return super.hashCode(); }  
   B. public int hashCode () { return 42; }
C. public int hashCode () { return x; }
D. public int hashCode () { return x + y; }
E. public int hashCode () { return x * 3 + y; }
F. public int hashCode () { return x * 1000 + y; }

G. Given the following recursive method mystery:

```java
int mystery(int n)
{
    if (n < 0)
        return -mystery(-n);
    else if (n < 10)
        return n;
    else
        return mystery(n/10 + n % 10);
}
```

What are would following calls evaluate to?

I. mystery(7)

II. mystery(-512)

H. Draw the list (i.e. that is a box-and-pointer diagram) resulting from the following code.

```java
Node m = l.next.next;
Node n = new Node("5", m);
```

Your answer should clearly note what l, m, and n point to.
PROBLEM 2: (Reverse (9 points))
Each of the Java functions on the left take a string s as input, and returns its reverse. For each of the following, state the recurrence (if applicable) and give the big-Oh complexity bound.
Recall that concatenating two strings in Java takes time proportional to the sum of their lengths, and extracting a substring takes constant time.

A. public static String reverse1(String s) {
    int N = s.length();
    String reverse = "
    for (int i = 0; i < N; i++)
        reverse = s.charAt(i) + reverse;
    return reverse;
}

B. public static String reverse2(String s) {
    int N = s.length();
    if (N <= 1) return s;
    String left = s.substring(0, N/2);
    String right = s.substring(N/2, N);
    return reverse2(right) + reverse2(left);
}

C. public static String reverse3(String s) {
    int N = s.length();
    char[] a = new char[N];
    for (int i = 0; i < N; i++)
        a[i] = s.charAt(N-i-1);
    return new String(a);
}
PROBLEM 3: (Boggle)
The game of Boggle is usually played using sixteen letter cubes. A letter cube can be represented as a string of length 6, one character for each face on the cube. Given a list of letter cubes and a target word, you want to determine whether it is possible to spell that word using those letter cubes, where each cube can be used at most once in spelling the word.

Examples: given the list of cubes \{"etaoin", "shrdlu", "qwertxy"\}, it is possible to spell the words "as" and "law!" but not "weld" or "toe".

Write a recursive, backtracking method canSpell that takes a target word along with an ArrayList of letter cubes. The function should return true if it is possible to spell the word using the cubes in the list and false if otherwise. Assume that the word and letter cubes will only contain lowercase letters.

```java
public static boolean canSpell(String word, ArrayList<String> cubes)
{
```
PROBLEM 4:  (Puzzle Hunt)

You are given a matrix of positive integers to represent a game board, where the \((0, 0)\) entry is the upper left corner. The number in each location is the number of squares you can advance in any of the four primary compass directions, provided that move does not take you off the board. You are interested in the total number of distinct ways one could travel from the upper left corner to the lower right corner, given the constraint that no single path should ever visit the same location twice.

Consider the initial game board to the left, and notice that the upper left corner is occupied by a 2. That means you can take either two steps to the right, or two steps down (but not two steps to the left or above, because that would carry you off the board). Suppose you opt to go right so that you find yourself in the configuration to the right.

![Game Board Example]

After that, you could continue along as follows:

![Continued Game Board Example]

This series of moves illustrates just one of potentially several paths you could take from upper left to lower right. Your task is to write a method called `numPaths`, which takes a 2-d array of integers and computes the total number of ways to travel to the lower right corner of the board. Note that you never want to count the same path twice, but two paths are considered to be distinct even if they share a common sub-path. And because you want to prevent cycles, you should change the value at any given location to a zero as a way of marking that you’ve been there. Just be sure to restore the original value as you exit the recursive call. You may want to write a helper function to handle the recursion and a utility function to decide it you are on the board or not.
A. Write `numPaths` below.

```java
/**
 * Calculates total number of distinct ways one could travel from the
 * upper left corner of grid to the lower right corner, given the
 * constraint that no single path should ever visit the same location twice.
 * @param board square matrix board[i][j] is the number of squares
 * one can advance vertically or horizontally from (i,j)
 * @return the number of possible paths from (0,0) to the lower
 * right corner of board (board.length-1, board[0].length - 1)
 */
public static int numPaths(int[][] board)
{

    // HELPER FUNCTIONS
    /**
     * @return true if (row,col) is within the bounds of the board
     * (i.e. 0 <= row < board.length and 0 <= col < board[0].length)
     * false otherwise
     */
    public static boolean onBoard(int[][] board, int row, int col)
    {

    } // HELPER FUNCTIONS

    /**
     * @return the number of possible paths from (row,col) to the lower
     * right corner of board (board.length-1, board[0].length - 1)
     */
    public static int numPaths(int[][] board, int row, int col)
    {
```
B. Give a recurrence for your solution. You do not need to solve the recurrence.
Write the method, subseq to select an evenly spaced subsequence of a list. For example, if list is a linked list containing the strings

"aardvark", "basilisk", "axolotl", "gibbon", "gnu", "kumquat", "grapefruit", "kiwi", "poplar"

then both subseq1(L, 1, 3) and subseq2(L, 1, 3) return

"basilisk", "gnu", "kiwi"

A.  /** A List consisting of elements I, I+K, I+2K, ... of L, numbering
    * from 0. If I is greater than or equal to the length of L, the
    * result will be the empty list. Requires I>= 0, K>0
    * Does NOT modify any of the Node objects in the original list
    * referred to by list
    * @param list refers to a list of words
    * @param i starting index >= 0
    * @param k spacing > 0
    * @return beginning of list of elements i, i+k, i+2k,... of list
    */

    static Node subseq1(Node L, int i, int k)
    {
        //FILL THIS IN (about 8 lines)
    }

B.  /** Return the array resulting from inserting each of the Strings
    * in the List STRINGS into the array A. STRINGS is assumed to
    * contain only non-null Strings. The Strings from STRINGS are
    * processed in sequence, and each is inserted just before the
    * first String in the resulting array that is larger
    * alphabetically.
    * Assume there are no duplicated Strings. For example, if A is
    * the array
    *   { "mess", "here’s" }
    * and STRINGS contains the Strings
    *   [ "pretty", "in", "less", "a" ]
    * then the resulting array contains
    *   { "a", "in", "less", "mess", "here’s", "pretty"
    */

    static String[] insertAll(String[] A, List strings) {
        // FILL IN (about 10 lines)
PROBLEM 6:  (Strands)

We would like to keep track of long strands of characters, (e.g. DNA), using a class, so we create the `IStrand` interface.

```java
interface IStrand {
    public void prepend(String s);
    public int length();
}
```

Below are two classes, `StringStrand` and `LinkStrand` that implement this interface.

```java
class StringStrand implements IStrand {
    private String myString;
    public StringStrand(){
        myString = "";
    }

    public void prepend(String s){
        myString = s + myString;
    }

    public int length(){
        return myString.length();
    }
}

class LinkStrand implements IStrand {
    private Node myFront;
    private int myCount;

    public LinkStrand(){
        myFront = null;
        myCount = 0;
    }

    public void prepend(String s){
        myFront = new Node(s,myFront);
        myCount += s.length();
    }

    public int length(){
        return myCount;
    }
}
```
A. Which class do you think would be faster when performing \textit{n prepend} operations? Briefly explain why.

B. Instead of just prepending to a Strand, we would like to splice in a String and insert it anywhere. That is, we would like to have \texttt{s.insert(k,str)} to add str at the \textit{k}th position. Prepending could now also be performed as \texttt{s.insert(0,str)}. We would like this behavior in both \texttt{LinkStrand} and \texttt{StringStrand}. What do we need to modify in \texttt{I Strand}?

C. Write the \texttt{insert} method for \texttt{StringStrand} below.
D. Write the `insert` method for `LinkStrand` below.

E. Which `insert` method will be faster for large strands? Briefly explain why.
PROBLEM 7:  (Trees (27 points))

The questions in this problem will use the tree below.

A. [3pts] The inorder traversal of the tree below is:
   badger, bison, crayfish, eagle, hedgehog, hyena, iguana, octopus, wallaby.
   What is the post-order traversal?

B. [4pts] Show where the values fox, koala, and zebra would appear if inserted into the search tree above by adding the nodes in the diagram.

C. [4pts] Complete the method below that returns a copy of the tree shown above (in general a copy of any tree).

   ```java
   public TreeNode copy(TreeNode root){
     if (root == null) // add code
       return new TreeNode(root.info,
               copy(root.left),
               copy(root.right));  // add code
   }
   ```
D. [6pts] Write method oneChildCount that returns the number of nodes in a tree that have one child (e.g., not zero children and not two children). In the tree at the beginning of this problem the value returned would be two since the nodes labeled octopus and hedgehog in the original tree each have one child.

```java
public int oneChildCount(TreeNode root) {
}
```

E. [6pts] The two methods print and printItems below generate the following output:

```
iguana eagle bison badger
iguana eagle bison crayfish
iguana eagle hedgehog hyena
iguana octopus wallaby
```
when invoked with the call `print(tree,new LinkedList<String>());` where `tree` is the root of the tree on the previous page.

```java
public static void printItems(LinkedList<String> list){
    Queue<String> copy = (Queue<String>) list.clone();
    while (copy.size() != 0){
        System.out.print(copy.remove()+" ");
    }
    System.out.println();
}
```

```java
public static void print(TreeNode root, LinkedList<String> q){
    if (root == null) return;
    if (root.left == null && root.right == null){
        q.add(root.info);
        printItems(q);
        q.removeLast();
        return;
    }
    q.add(root.info);
    print(root.left,q);
    print(root.right,q);
    q.removeLast();
}
```
If `LinkedList` is replaced everywhere with `Stack` – and `add` replaced by `push`, `removeLast` by `pop` and so forth in both methods above then what will be printed by the call `print(tree, new Stack<String>())` – you should show four lines of output in your answer.

You should also briefly explain why the queue is cloned in the method `printItems`. 