Test 1: CompSci 100

Name (print): ____________________________________________

Community Standard acknowledgment (signature): ____________________________

<table>
<thead>
<tr>
<th>Problem</th>
<th>Value</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem 1</td>
<td>9 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 2</td>
<td>9 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 3</td>
<td>6 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 4</td>
<td>14 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 5</td>
<td>12 pts.</td>
<td></td>
</tr>
<tr>
<td>TOTAL:</td>
<td>50 pts.</td>
<td></td>
</tr>
</tbody>
</table>

This test has 8 pages, be sure your test has them all. Do NOT spend too much time on one question — remember that this class lasts only 50 minutes and there are 50 points on the exam. That means you should spend no more than 1 minute per point.

You may consult your four (4) sheets and no other resources. You may not use any computers, calculators, or cell phones. You may refer to any program text supplied in lectures or assignments.

Don’t panic. Just read all the questions carefully to begin with, and first try to answer those parts about which you feel most confident. Do not be alarmed if some of the answers are obvious.

If you think there is a syntax error or an ambiguity in a problem’s specification, then please ask.

In writing code you do not need to worry about specifying the proper import statements. Assume that all libraries and packages we’ve discussed are imported in any code you write.
PROBLEM 1: (Big-Oh (9 points))

For each of the following, either indicate that it is true and briefly tell why or give a counterexample.

A. A $O(n)$ algorithm is always preferable to a $O(n^2)$ algorithm.

B. $n \log n + 4n^2 \in O(n^2)$

C. $3^n \in O(2^n)$
PROBLEM 2:  (*Sets (9 points)*)

A.  What is $\log_2 1000$ to the nearest integer?

B.  A set of values is maintained as a linked list, sorted in increasing order.  What are the tightest asymptotic bounds you can give for the best and worst-case times for performing $N$ insertions into this set, starting from an empty set?  By tight bounds, we mean that if the program runs in time $T(n)$, give $O(f(n))$ such that $T(n) \in O(f(n))$ and $T(n) \notin o(f(n))$.  We want bounds for the worst-case time and bounds for the best-case time.

C.  A set of values is maintained as an array, sorted in increasing order, plus an integer variable containing the current number of items in the set (which may differ from the size of the array).  What are the tightest asymptotic bounds you can give for the best and worst-case times for performing $N$ insertions into this set, starting from an empty set?  Justify your answer.  (Assume an ideal computer where $N$ can be arbitrarily large and is not known in advance, and at any given time, the array has some finite size.  Assume also that comparisons on the objects in the array require constant time.)
PROBLEM 3:  *(Stacks and Queues (6 points))*

A. Describe the contents of stack s after the method *convert* executes. That is, describe the contents in a general manner based on what’s in s before the code executes.

```java
public void convert(Stack<Object> s){
    ArrayList<Object> list = new ArrayList<Object>();
    while (s.size() > 0) {
        list.add(s.pop());
    }
    for(Object o : list) {
        s.push(o);
    }
}
```

B. What happens if a queue is used instead of a stack in the code above, e.g.,

```java
public void convert(Queue<Object> q){
    ArrayList<Object> list = new ArrayList<Object>();
    while (q.size() > 0) {
        list.add(q.remove());
    }
    for(Object o : list) {
        q.add(o);
    }
}
```
PROBLEM 4:  (Subsequence (14 points))

In this method, you will write methods to select an evenly spaced subsequence of a list. For example, if list is a linked list containing the strings

"aardvark", "batty", "lee", "collins", "gnu", "madsen", "moseley", "kde", "paulson"

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

"batty", "gnu", "kde"

A. Write subseq1 that returns the subsequence as described above but does not modify the original list.
You may create new Node objects.

```java
static Node subseq1(Node L, int i, int k)
{
    //FILL THIS IN (about 8 lines)
}
```
B. For this version, do **not** use the `new` operator or in any other way allocate new `Node` objects.

```java
/* A linked 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
* May modify List objects in the list pointed to by L.
*/
public static Node subseq2(Node L, int i, int k)
{
    //FILL THIS IN (about 16 lines)
```
PROBLEM 5 : (Boggle (12 points))

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 an ArrayList 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 vector of cubes {"etaoin", "shrdlu", "qwert"}, it is possible to spell the words "as" and law! but not "weld" or "toe".

A. 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) {
```

B. Give the recurrence relation for your solution.
Throughout this test, assume that the following classes and methods are available. These classes are taken directly from the material used in class. There should be no methods you have never seen before here.

Definitions

Some common recurrences and their solutions.

- \( T(n) = T(n/2) + O(1) \) \( \mathcal{O}(\log n) \)
- \( T(n) = T(n/2) + O(n) \) \( \mathcal{O}(n) \)
- \( T(n) = 2T(n/2) + O(1) \) \( \mathcal{O}(n) \)
- \( T(n) = 2T(n/2) + O(n) \) \( \mathcal{O}(n \log n) \)
- \( T(n) = T(n-1) + O(1) \) \( \mathcal{O}(n) \)
- \( T(n) = T(n-1) + O(n) \) \( \mathcal{O}(n^2) \)

List Node

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

String

public class String {
    // Compares this string to the specified object.
    // The result is true if and only if the argument
    // is not null and is a String object that
    // represents the same sequence of characters */
    public boolean equals(Object anObject)
    /* Returns the index within this string of the
    * first occurrence of the specified substring.
    * -1 if it does not exist */
    public int indexOf(String str)
    /* Returns the length of this string. */
    public int length()
    /* Searches for the first occurrence of the given
    * argument, returns -1 if not found */
    public int indexOf(Object elem)
    /* Returns a new string that is a substring of this
    * string. Begins at the specified beginIndex and
    * extends to the character at index endIndex - 1 */
    String substring(int beginIndex, int endIndex)
}

TreeSet/HashSet

public class TreeSet {
    // Constructs a new, empty set
    public TreeSet()
    // Returns an iterator over the elements in
    // this set. The elements are visited in
    // ascending order.
    public Iterator iterator()
    // Returns the number of elements in this set.
    public int size()
    // Returns true if this set contains o
    public boolean contains(Object o)
    // Adds the specified element to this set
    // if it is not already present.
    public boolean add(Object o)
}

Queue

public class Queue {
    /* A new element <code>o</code> is added to the queue */
    public boolean add(Object o)
    /* Retrieves and removes head of queue */
    public Object remove()
    // Retrieves, but does not remove, the head of this queue, or returns null.
    public Object peek()
}

Stack

public class Stack {
    /* Pushes an item onto the top of this stack. */
    public Object push(Object o)
    /* Removes the object at the top of this stack and returns that object. */
    public Object pop()
    /* Tests if this stack is empty */
    public boolean empty()
}

ArrayList

public class ArrayList {
    // Constructs an empty list
    public ArrayList()
    // Returns the number of elements in this list.
    public int size()
    // Searches for the first occurrence of the given
    // argument, returns -1 if not found */
    public int indexOf(Object elem)
    // Returns the element at the specified position
    // in this list.
    public Object get(int index)
    // Replaces the element at the specified position
    // in this list with the specified element. */
    public Object set(int index, Object element)
    // Inserts the specified element at the specified position
    public void add(int index, Object o)
    // Returns Iterator over elements in list
    public Iterator iterator()
}
