Due Friday, October 5 in class

Big-Oh

1. Suppose $T_1(n) \in O(f(n))$ and $T_2(n) \in O(f(n))$. Answer whether the following are true or false and give justification. A justification for being true is a simple mathematical proof, while you can prove something to be false by giving a specific counterexample. You should use the definition of big-Oh in both cases.

(a) $T_1(n) + T_2(n) \in O(f(n))$

(b) $T_1(n) - T_2(n) \in O(f(n))$

(c) $T_1(n)/T_2(n) \in O(1)$

(d) $T_1(n) \in O(T_2(n))$
2. By doubling the size of an array used to store an ArrayList, we pay constant amortized time for each add operation. Suppose that allocating a ArrayList containing M elements takes $M/2 + 10$ time units, copying M elements from one ArrayList to another takes M time units, and a push takes 1 time unit plus the amount of time (if any) required to increase the size of the ArrayList.

(a) If the size of the ArrayList increases by 100 (that is, 100 more elements, not 100 times as many), how long will N add operations take?

(b) If the size of the ArrayList doubles each time the stack fills up, how long will N add operations take?

(c) If the size of the ArrayList increases by a factor of 1.5 each time, how long will N adds take?

3. Extra Credit: Show $(\log n)^K \in O(n)$ for any $K$. 
Linked Lists

In all these problems assume that the inner class Node exists as shown below. You’ll write static methods in a class LinkProblems for your solutions. You cannot use instance variables, all variables must be local to the static methods you write.

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

    // your methods here
}

4. Write the method fruitCounter whose header follows. Method fruitCounter returns a count of the number of nodes whose info field has a value for which a static method isFruit (see below) returns true.

    public static boolean isFruit(String s){
        // code not shown
    }

For example, assuming that "apple", "orange", and "pear" are fruits (isFruit returns true) and "bear", "coyote", and "fox" are not fruits, and list is represented by:

("apple", "bear", "apple", "orange", "coyote", "fox", "orange", "pear")

the call fruitCounter(list) should evaluate to 5 since there are five fruits.
Write two versions, one iterative and one recursive.

/**
 * @return the number of nodes whose info field is a fruit
 * as determined by method isFruit
 */
public static int fruitCounter(Node list)
5. Occurs Check

(a) Write a method `hasDuplicates` whose header follows. The method returns true if parameter list has any duplicates (words occurring more than once) and false otherwise. For example, for the list

( "apple", "guava", "cherry"

`hasDuplicates` should return `false`, but it would return `true` for the list below since "guava" appears twice.

( "apple", "guava", "cherry", "guava"

In writing `hasDuplicates` you **must** call the method `countOccurrences` shown below and your method must be either a recursive function with no loop or a function with one loop. Either version can include calls to `countOccurrences`. You cannot use any ArrayList, Set, etc. objects in your code.

```java
/**
 * @return the number of occurrences of s in list
 */
public static int countOccurrences(Node list, String s) {
    if (list == null) return 0;
    int count = 0;
    if (list.info.equals(s)) count = 1;
    return count + countOcurrences(list.next,s);
}
```

```java
/**
 * @return true if and only if list has duplicates
 */
public static boolean hasDuplicates(Node list) {
```
(b) What is the complexity (using big-Oh) of the solution you wrote to hasDuplicates and why?

(c) Describe how to write a more efficient solution to the hasDuplicates method using, for example, a TreeSet or a HashSet instead of calling countOccurrences. Be sure to indicate what the complexity of your solution is and why.
6. **Extra credit**: Write a method that determines whether a list has any cycles in it using \textit{constant} extra space.

The prototype for the function is:

```java
/**
 * Returns true if and only if list has no cycles, that is no
 * node appears multiple time in the list
 * Amount of space allocatd is constant, i.e. not proportional to the
 * size of the list
 * @param list that should not be changed
 * @return true if and only if list is circular
 */
public static boolean isCircular(Node list)
```

7. The following problems use the class `TermNode` below to represent a single term of a polynomial. For example $3x^{100}$ can be represented by `TermNode(3,100,null)`; and $7x^{50} + 2x^5 + 8$ is represented by

```
TermNode poly = new TermNode(7,50, new TermNode(2,5, new TermNode(8,0,null)));
```

Here's the class definition.

```java
public static class TermNode
{
    int coeff;
    int power;
    TermNode next;
    TermNode(int co, int po, TermNode follow) {
        coeff = co;
        power = po;
        next = follow;
    }
}
```
(a) Write a method `makePolyNomial` that takes a polynomial expressed in array form in which every term is explicitly stored (including those with zero-coefficients) and returns a polynomial expressed in linked list form (list of TermNodes) in which just terms with non-zero coefficients are stored. For example, given a array of \([3, 0, 0, 2, 5]\), your function should return the list \((3, 4), (2, 1), (5,0)\) since both represent \(3x^4 + 2x + 5\).

Here we assume the 5 in the array has index 0 and the 3 has index 4, so the array is shown with the zero-index on the right (but your method doesn’t have a right or a left, just an array with a length and int coefficients).

```java
/**
   * Returns a list of TermNodes representing y poly,
   * the elements in poly are in sorted order by power/exponent
   * with largest exponent first, no zero-coefficient terms in list returned.
   */
public static TermNode makePolyNomial(int[] poly)
```

(b) Write a method `addPolyNomial` that takes two polynomials expressed as lists of TermNodes, and returns a new polynomial representing the sum of the two parameters. The parameters should not be altered as a result of calling `addPolynomial`, a new polynomial is created and returned.

For example, \((3x^4 + 2x + 4) + (2x^2 - 4x - 9) = (3x^4 + 2x^2 - 2x - 5)\)

\[
((3,4),(2,1),(4,0)) + ((2,2),(-4,1),(-9,0)) = \\
((3,4),(2,2),(-2,1),(-5,0))
\]

```java
// pre: elements in p1 and p2 are sorted by power, largest to smallest
//      (standard form for this problem)
// post: returns polynomial/list of TermNodes
//       representing the sum of p1 and p2
/**
   * Return polynomial in standard form of two standard form polynomials.
   * @param p1 is sorted by power, large to small, standard form
```
public static TermNode addPolyNomial(TermNode p1, TermNode p2)
(c) Extra credit: Write a method `multPolyNomial` that returns the simplified result of multiplying this TermNode with another polynomial.

For example \((x - 2)(x + 2) = x^2 + 2x - 2x - 4\), but your function should actually return \(x^2 - 4\) or `multPolyNomial( ((1,1),(2,0)), ((1,1),(-2,0)) )` → `((1,2),(-4,0))`

```java
/**
 * Return polynomial in standard form of product of two standard form polynomials.
 * @param p2 is sorted by power, large to small, standard form
 * @return polynomial consisting of new TermNodes representing
 * product of this and p2. The polynomial should have no redundant terms
 */
public static TermNode multPolyNomial(TermNode p2)
```