Test 1: Compsci 100

Owen Astrachan

February 16, 2009

Name: _____

Login: _____

Honor code acknowledgment (signature) _____

	value	grade
Problem 1	20 pts.	
Problem 2	18 pts.	
Problem 3	16 pts.	
Problem 4	14 pts.	
Problem 5	12 pts.	
TOTAL:	80 pts.	

This test has 17 pages (there's an insert and two pages at the end that aren't numbered), be sure your test has them all. Do NOT spend too much time on one question — remember that this class lasts 75 minutes.

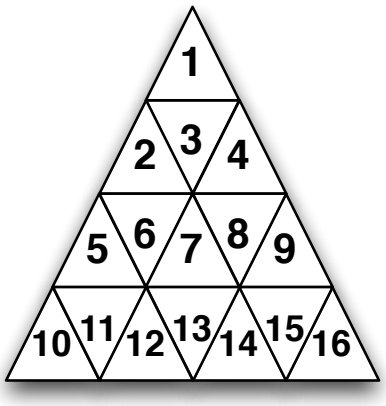
In writing code you do not need to worry about specifying the proper `import statements`. Assume that all libraries and packages we've discussed are imported in any code you write.

Unless indicated otherwise, here's the `Node` class for this test.

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

PROBLEM 1 : (*TriPyramid (20 points)*)

The picture below shows a four-rowed, two-dimensional pyramid constructed of triangles – this will be called a *4-pyramid* in this problem because it has four rows. The top triangle is number one, then the triangles are numbered left-to-right in a row and top-to-bottom as shown. In the fourth row there are seven triangles; in general in the N^{th} row there are $2N-1$ triangles. In answering questions below assume we're discussing an N-pyramid with N rows for a large value of N.



Part A (2 points)
What is the *exact* value or number of the right-most triangle in the seventh row?

Part B (2 points)
What is the *exact* value of the right-most triangle in the 40^{th} row?

Part C (2 points)
What is the *exact* value of the left-most triangle in the 61^{st} row?

Part D (2 points)

Using big-Oh what is the number of triangles in the bottom row of a pyramid with N rows. Justify your answer.

Part E (2 points)

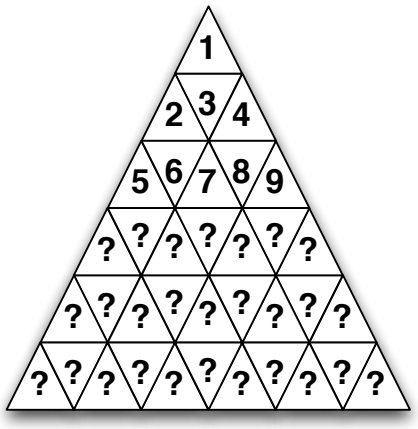
Using big-Oh what is the number of triangles in the bottom row of a pyramid with N^2 total triangles. Justify your answer.

Part F (2 points)

Using big-Oh what is the number of triangles in the bottom row of a pyramid with N^4 total triangles. Justify your answer.

Part G (8 points)

The diagram below shows four 3-pyramids combined to make an 11-pyramid. Assume there is a function or method *combine* that takes an N-pyramid as a parameter and returns a new pyramid created by combining four N-pyramids as shown. For example, the pyramid shown below would be returned by the call `combine(3)`.



G.1 (2 points)

What is the big-Oh number of triangles in the bottom row of the pyramid returned by the call `combine(N)`. Justify your answer.

G.2 (2 points)

What is the big-Oh value of the rightmost pyramid in the bottom row (the triangle with the largest number) in the pyramid returned by the call `combine(combine(N))`. Justify your answer.

G.3 (4 points)

Consider the pseudo-code below for a sequence of calls to create a pyramid. For example, when $N = 2$ the initial pyramid `p` is constructed before the loop with two rows and three triangles in the bottom row; the loop then executes twice. The first time through the loop results in `p` having four rows. The second time through the loop results in `p` having eight rows (with 15 triangles in the bottom row and the value printed is 64).

```
Pyramid p = new Pyramid(N); // create pyramid with N rows

int size = p.rows();  // set size to N
for(int k=0; k < size; k++){
    p = combine(p);
}
System.out.println("biggest number is "+p.lastPyramid());
```

If the value of $N = 10$ so that the initial pyramid has 10 rows in which the last pyramid is numbered 100 what is the value printed by the code above? Justify your answer. Your answer should be exact, but can be expressed using exponentiation and multiplication, e.g., $3^5 \times 100^2$ is acceptable as an answer.

PROBLEM 2 : (*Linkin (18 points)*)

The method `firstToLast` below moves the first element in a list so that it's the last element in the list.

```
public List<String> firstToLast(List<String> list){
    String first = list.remove(0);
    list.add(first);
    return list;
}
```

The timings below result from calling `firstToLast` with both a `LinkedList<String>` and an `ArrayList<String>` with the number of elements (size of the list) indicated in the left-most column. These timings are relatively consistent in their pattern across different versions of Java on different machines, e.g., running on a Mac, Linux, or Windows machine.

Times are in seconds for a `LinkedList<String>` and an `ArrayList<String>`.

size (10^3)	**link**	**array**
10	0.031	1.617
20	0.086	6.442
30	0.125	14.180
40	0.160	
50	0.199	
60	0.219	
70	0.232	
80	0.282	
90	0.358	
100	0.329	

The code for timing is below:

```
String first = list.get(0);
double start = System.currentTimeMillis();
for(int j=0; j < 20*list.size(); j++){
    list = firstToLast(list);
}
double end = System.currentTimeMillis();
System.out.printf("%d\t%1.3f\n",list.size(),(end-start)/1000.0);
if (! first.equals(list.get(0))){
    System.out.printf("first failure at %d\n",list.size());
}
```

Part A (3 points)

Expain in a sentence of two the timing results for the column labeled **link**. You should explain the pattern of the results, not the particular running times.

Part B (6 points)

Project a time for the column labeled **array** for a 40,000 element list and a 100,000 element list. Justify both answers.

(continued)

Part C (3 points)

Provide a plausible reason for the timing loop having a limit of `20*list.size()` instead of `list.size()`.

Part D (6 points)

Assume you're implementing a linked list using a `Node` class as shown below.

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

The method `last` below returns a pointer to the last node of a linked list, or `null` if there is no last node.

```
public Node last(Node list){
    if (list == null) return null;
    while (list.next != null){
        list = list.next;
    }
    return list;
}
```

Write the method `firstToLast` that moves the first node of a linked list so that it's the last node and returns a pointer to the new first node (what was originally the second node). You can call `last` on this page, you cannot call `new` in writing your code.

```
public Node firstToLast(Node list) {









}
```

PROBLEM 3 : (*APT analysis (16 points)*)

The write-up for the *Anagram* APT you were assigned is attached at the end of this test. The solution on the next page is *all-green*, it passes all test cases. You'll be asked a few questions about the code.

Part A (2 points)

Briefly, why is an `ArrayList` declared and used in the method `anagrams` when an array is ultimately returned from the method?

Part B (2 points)

In the constructor for the inner class !Ana! the String parameter `s` is converted to lower case, sorted, and leading and trailing spaces are removed. Why is it necessary to call `trim()` to remove spaces *after* the String is sorted, e.g., why won't these lines work:

```
public Ana(String s){
    word = s;
    char[] chars = s.toLowerCase().trim().toCharArray();
    Arrays.sort(chars);
    sortedWord = new String(chars));
}
```

Part C (4 points)

If the `TreeSet` created in `anagrams` is replaced by `HashSet` the code compiles but passes fewer than one-third of the test cases. There's a bug in the class `Ana` that causes this failure. What's the bug and how do you fix it? (You won't need to change must code if you find the bug.)

(continued)

```
import java.util.*;

public class Aaagmnrs {

    public class Ana implements Comparable<Ana>{
        public String word;
        public String sortedWord;
        public Ana(String s){
            word = s;
            char[] chars = s.toLowerCase().toCharArray();
            Arrays.sort(chars);
            sortedWord = new String(chars).trim();
        }
        public int compareTo(Ana o) {
            return sortedWord.compareTo(o.sortedWord);
        }
        public boolean equals(Object o){
            Ana other = (Ana) o;
            return word.equals(other.word);
        }
        public int hashCode(){
            return sortedWord.hashCode();
        }
    }

    public String[] anagrams(String[] phrases){
        ArrayList<String> unique = new ArrayList<String>();
        Set<Ana> set = new TreeSet<Ana>();
        for(String s : phrases){
            Ana a = new Ana(s);
            if (! set.contains(a)) {
                set.add(a);
                unique.add(s);
            }
        }
        return unique.toArray(new String[0]);
    }
}
```

Part D (8 points)

The APT *ClientsList* is attached to the end of this test. The code below could be *all-green* if the methods `Inner.compareTo` and `Inner.toString` are implemented. Implement the methods so that the solution below is all-green.

```
public class ClientsList {
    public class Inner implements Comparable<Inner>{
        String first,last;
        public Inner(String s){
            String[] sp = s.split(" ");
            if (sp[0].endsWith(",")){
                first = sp[1];
                last = sp[0].substring(0,sp[0].length()-1);
            }
            else {
                first = sp[0];
                last = sp[1];
            }
        }
        public int compareTo(Inner o) {
            // *TODO* you implement this method









        }
        public String toString(){
            // *TODO* you implement this method




        }
    }
    public String[] dataCleanup(String[] names){
        Inner[] list = new Inner[names.length];
        for(int k=0; k < names.length; k++){
            list[k] = new Inner(names[k]);
        }
        Arrays.sort(list);
        for(int k=0; k < names.length; k++){
            names[k] = list[k].toString();
        }
        return names;
    }
}
```

PROBLEM 4 : (*You really, really like me (14 points)*)

Suppose Facebook changes the *liking* relationship so that it is asymmetrical/one-way: it's possible for Fred to like Jim, but for Jim not to like Fred. To keep track of friend relationships we use an instance variable that's a map in which the key is the name of a person and the corresponding value is a list of the person's friends. In this problem all instance variables and methods are in a class `FriendStuff`.

```
public class FriendStuff {
    private Map<String,ArrayList<String>> myFriends;
    ...
```

Given a map representing friendship data we can reason about the relationships represented in the map. For example, the code below returns the name of the person with the most friends (assuming there are no ties).

```
public String friendliest(){
    String most = null;
    int max = 0;
    for(String s : myFriends.keySet()){
        int size = myFriends.get(s).size();
        if (size > max){
            max = size;
            most = s;
        }
    }
    return most;
}
```

For the next problems we'll use the sample input data below to help explain the questions; this data is stored in `myFriends`. Each line represents the name of a person followed by a friend of that person; assume this information is loaded correctly into `myFriends`. In the list below *owen* has two friends: *jeff* and *robert*; four other people have *jeff* as a friend, but jeff's only friend is *landon*.

```
"owen jeff"
"owen robert"
"susan jeff"
"susan robert"
"susan owen"
"robert jeff"
"robert susan"
"vince jeff"
"jeff landon"
"landon jeff"
"alvy landon"
"landon alvy"
```

In writing the two methods that follow you may find it useful to call the `ArrayList.indexOf` method. Given a value, e.g., a String when called on an ArrayList of Strings, `indexOf` returns the index at which the String occurs in the ArrayList, or -1 if the String doesn't occur. For example, if `list.indexOf("george")` evaluates to 2 then `"george"` occurs at index 2 in `list`; if `list.indexOf("fred")` evaluates to -1 then `"fred"` does **not** occur in `list`.

Part A (6 points)

Suppose we define the *hate* relationship as the situation when A likes B, but B doesn't like A; we say that this means B *hates* A. So, if B appears in A's friend list, but A does not appear in B's friend list,then B hates A. Complete method `hateCount` so that it returns the number of people that hate `name`. For the data on the previous page `hateCount("owen")` should return 2 since according to the definition *robert* and *susan* hate *owen*.

```
public int hateCount(String name){
      int total = 0;
      for(String friend : myFriends.get(name)){









      }
      return total;
}
```

Part B (8 points)

Write the method `mostLikedBy` to return the name of the person who is liked by more people than anyone else, i.e., who appears in more friend-lists than any other person (don't worry about ties). For the data above your code should return *jeff*. The method you write should access the data stored in `myFriends` in calculated who is liked the most.

```
public String mostLikedBy(){

}
```

PROBLEM 5 : (***Top 10 (12 points)***)

The top three songs played on in iPod during a week are stored in comma-separated string so `"Heartless,Just Dance,Let It Rock"` indicates the most listened to song is *Heartless*, the second most listened to song is *Just Dance* and the third is *Let It Rock*.

Given the top-three lists from several iPods as input, write a method that returns the most popular song. In determining the most popular song, give three points to the first song in each top-three list; give two points to the second song; and give one point to the third song. For example, given the array of four top-three lists below on the left, points would be awarded as shown on the right.

```
"Heartless,Just Dance,Let it Rock"
"Just Dance,Dead and Gone,Viva la Vida",
"Viva la Vida,Heartless,Poker Face",
"Let it Rock,Poker Face,Just Dance"
```

song	points	total
Heartless	3+2	5
Just Dance	2+3+1	6
Let It Rock	1+3	4
Dead and Gone	2	2
Viva la Vida	1+3	4
Poker Face	1+2	3

So that the method you write should write `Just Dance`. If two songs are tied with the most points return the one that occurs first alphabetically.

```
public String mostPopular(String[] playlists) {




}
```

(nothing on this page)

Anagrams

Problem Statement

Two phrases are anagrams if they are permutations of each other, ignoring spaces and capitalization. For example, "Aaagmnrs" is an anagram of "anagrams", and "TopCoder" is an anagram of "Drop Cote". Given a String[] phrases, remove each phrase that is an anagram of an earlier phrase, and return the remaining phrases in their original order.

Definition

- Class: Aaagmnrs
- Method: anagrams
- Parameters: `String[]`
- Returns: `String[]`
- Method signature:

```
String[] anagrams(String[] phrases)
```

  (be sure your method is public)

Class

```
public class Aaagmnrs
{
  public String[] anagrams(String[] phrases)
  {
    // fill in code here
  }
}
```

Constraints

- `phrases` contains between 2 and 50 elements, inclusive.
- Each element of `phrases` contains between 1 and 50 characters, inclusive.
- Each element of `phrases` contains letters ('a'-'z' and 'A'-'Z') and spaces (' ') only.
- Each element of `phrases` contains at least one letter.

Examples

1. ```
   { "Aaagmnrs", "TopCoder", "anagrams", "Drop Cote" }

   Returns: { "Aaagmnrs",  "TopCoder" }
   ```

   The example above.

2. ```
   { "SnapDragon vs tomek", "savants groped monk", "Adam vents prongs ok" }

   Returns: { "SnapDragon vs tomek" }
   ```

3. ```
   { "Radar ghost jilts Kim", "patched hers first",
     "DEPTH FIRST SEARCH", "DIJKSTRAS ALGORITHM" }

   Returns: { "Radar ghost jilts Kim",  "patched hers first" }
   ```

Owen L. Astrachan
Last modified: Wed Jan 21 15:32:06 EST 2009
   ```

ClientsList

Problem Statement

Your company has just undergone some software upgrades, and you will now be storing all of the names of your clients in a new database. Unfortunately, your existing data is inconsistent, and cannot be imported as it is. You have been tasked with writing a program to cleanse the data.

You are given a String[], names, which is a list of the names of all of your existing clients. Some of the names are in "First Last" format, while the rest are in "Last, First" format. You are to return a String[] with all of the names in "First Last" format, sorted by last name, then by first name.

Definition

- Class: ClientsList
- Method: dataCleanup
- Parameters: `String[]`
- Returns: `String[]`
- Method signature:

```
public String[] dataCleanup(String[] names)
```

  (be sure your method is public)

Class

```
public class ClientsList
{
   public String[] dataCleanup(String[] names)
   {
            // fill in code here
   }
}
```

Constraints

- `names` will contain between 1 and 50 elements, inclusive.
- Each element of `names` will be of the form "First Last" or "Last, First" (quotes added for clarity).
- Each first and last name will begin with a single capital letter 'A'-'Z', and the remaining letters

will be lower case 'a'-'z'.

- Each element of `names` will contain between 3 and 50 characters, inclusive.

Examples

1.
```
{"Joe Smith", "Brown, Sam", "Miller, Judi"}
Returns: { "Sam Brown",  "Judi Miller",  "Joe Smith" }
```

The last `names`, in order, are Brown, Miller, Smith. We rearrange each name to be in the proper format also.

2.
```
{"Campbell, Phil", "John Campbell", "Young, Warren"}
Returns: { "John Campbell",  "Phil Campbell",  "Warren Young" }
```

Notice here how we sort by first name when the last `names` are the same.

3.
```
{"Kelly, Anthony", "Kelly Anthony", "Thompson, Jack"}
Returns: { "Kelly Anthony",  "Anthony Kelly",  "Jack Thompson" }
```

Be careful to properly identify first name versus last name!

4.
```
{"Trevor Alvarez", "Jackson, Walter", "Mandi Stuart",
 "Martin, Michael", "Peters, Tammy", "Richard Belmont",
 "Carl Thomas", "Ashton, Roger", "Jamie Martin"}
Returns:
{ "Trevor Alvarez",
 "Roger Ashton",
 "Richard Belmont",
 "Walter Jackson",
 "Jamie Martin",
 "Michael Martin",
 "Tammy Peters",
 "Mandi Stuart",
 "Carl Thomas" }
```

5.
```
{"Banks, Cody", "Cody Banks", "Tod Wilson"}
Returns: { "Cody Banks",  "Cody Banks",  "Tod Wilson" }
```

Notice that two identical `names` can appear in the list.

6.
```
{"Mill, Steve", "Miller, Jane"}
Returns: { "Steve Mill",  "Jane Miller" }
```

Notice that shorter `names` precede longer `names` alphabetically when the shorter name is a substring of the longer.