CS307 Spring 2011 Midterm 1 Solution and Grading Criteria.

Grading acronyms:
ABA - Answer by Accident
AIOBE - Array Index out of Bounds Exception may occur
BOD - Benefit of the Doubt. Not certain code works, but, can't prove otherwise
ECF - Error carried forward.
Gacky or Gack - Code very hard to understand even though it works or solution is not elegant. (Generally no points off for this.)
GCE - Gross Conceptual Error. Did not answer the question asked or showed fundamental misunderstanding
LE - Logic error in code.
NAP - No answer provided. No answer given on test
NN - Not necessary. Code is unneeded. Generally no points off
NPE - Null Pointer Exception may occur
OBOE - Off by one error. Calculation is off by one.

1. Answer as shown or -2 unless question allows partial credit.
No points off for differences in spacing, capitalization, commas, and braces
1 point for each part on G - J.

```
A.    3 10
B.    3 11 8 7
C.    9 7 4
D.    -2
E.    0 [5, 12, -5, 7]
F.    BCCABC
G.    1. Invalid
      2. Valid
H.    1. Valid
      2. Valid
I.    1. Valid
      2. Invalid
J.    1. Invalid
      2. Invalid
K.    h: 6
L.    4
M.    4
N.    Runtime error or Exception or ClassCastException
O.    10
```

2. Comments. I thought this would be an easy, problem, but many people had problems. The efficiency requirement was a hint to use a single loop. A double loop was possible if the inner loop updated the outer loop counter.

Common problems:
- confusion between array and list
- using container.length instead of size instance variable. (The array is not the list!)
- not keeping track of longest run, just returning length of last run
- not handling size 0 or 1 case correctly
- restarting length of current run at 0 instead of 1
- this(i) or this[i]. this is not an array
- not checking last run

Suggested Solution:

```
public int maxSortedLength() {
    if(size == 0)
        return 0; // special case
    max = 1;
    int current = 1;
    for(int i = 1; i < size; i++) {
        if(container[i] >= container[i - 1]){
            current++;
        }
        else {
            // run ended, check if best and reset
            max = Math.max(max, current);
            current = 1;
        }
    }
     // check last run
    max = Math.max(max, current);
    return max;
}
```

General Grading Criteria: 20 points

handle case of size 0 and 1 correctly: 2 points
track max length so far: 1 point
loop through elements of list: 3 points
use size not container.length: 3 points
check if current element part of ascending section: 2 points
decide if current run is best so far: 3 points
return answer: 1 point
loop O(N), only one loop. 5 points (lose this if major errors on rest of problem, ~8 points off on rest of problem)

3. Comments: I thought this was a hard problem, but most people did well. There are special cases with zeros, but we didn't take off for missing those. (The sample solution ignores them.) There is an interesting solution that calculates the constant for the first value and then checks as the other values in the resulting vector are calculating. This has a better best case Big O, but was not required.

Common problems:
- not ensuring floating point division. double d = int / int is still int division
- not storing values correctly
- not checking all values are the SAME constant. (A number of people just checked if each pair of values from xVec divided evenly into the result. That is not correct.)

Suggested Solution:

```
public boolean isEigenVector(int[] xVec) {
    assert (numRows() > 0) && (numRows() == numCols()) && (numRows() == xVec.length);

    int[] multResult = new int[xVec.length];
    for(int r = 0; r < numRows(); r++) {
        int sum = 0;
        for(int c = 0; c < numCols(); c++)
            sum += myCells[r][c] * xVec[c];

    }

    double magicNum = 1.0 * multResult[0] / xVec[0];
    boolean result = true;
    int index = 1;
    while(result && index < multResult.length) {
        double curNum = 1.0 *  multResult[index] / xVec[index];
        result = curNum == magicNum;
        index++;
    }

    return result;
}
```

General Grading Criteria: 30 points

Create array of proper length to hold result (or do calculation as you go): 3 points
Multiply xVec by myCells in MathMatrix:
- nested loop: 6
- correct calculation for resulting vector: 6

Find potential constant based on first value: 4 (is if not floating point div)
Check other values (may be completed as part of multiply)
- loop: 4 (no loop needed if completed along with multiply)
- calculation: 3
- check equals first constant: 2

return boolean: 2

4. Comments. This was a hard problem because of all the data abstraction going on. The method was in the Names class. The ArrayList of NameRecords called nameList had the data. A nested loop was necessary to create all combinations. (Easier to just start each loop control variable at zero.) After creating the compound name the easy approach was to call getName and check the result. I regret requiring a clone of the original name record being. It would have been a more straightforward question to just add the NameRecord when found. There is a very nice solution that uses the foreach loop, but it was not required. Also a lot of people called getName in the if statement and then the add. That would be very inefficient, but we didn't take off for it.

Common problems:
- not use get method on nameList
- confusion between nameList and the Names object
- using methods that were not allowed on the questions such as charAt, substring, length of String, or index of
- adding String instead of NameRecord to resulting ArrayList
- using == on Strings instead of .equals
- misusing for each loop

Suggested Solution

```
public ArrayList<NameRecord> getCompoundNames() {
    ArrayList<NameRecord> result = new ArrayList<NameRecord>();

    for(int i = 0; i < nameList.size(); i++) {
        for(int j = 0; j < nameList.size(); j++ ) {
            String compoundName = nameList.get(i).getName() +
                    nameList.get(j).getName();
            NameRecord temp = getName(compoundName);

            if(temp != null)
                result.add((NameRecord) temp.clone());
        }
    }
    return result;
}
```

General Grading criteria: 20 points
Create resulting ArrayList: 2 point
Nested loop to consider ALL combinations of names: 6 points (attempt 2, correct 4)
Create compound name: 4 points
Determine if compound name is present (another loop or simply getName): 3 points
If compound name is present, add to result: 4 points
return result: 1 point