CS314 Fall 2012 Midterm 1 Solution and Grading Criteria.

Grading acronyms:
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.
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

A. T(N) = 25N + 2, +/- 1 on each coefficient

B. O(N)

C. O(N^2)

D. O(N^3)

E. O(NlogN) base 3 okay

F. 12 seconds

G. 22 seconds

H. 320 seconds

I. 6 [BB, EE, C, A, D, C]

J. 1. syntax error (1 point)
2. syntax error (1 point)

K. true trueGoal (ignore spacing differences)

L. true Time true
false Time false

M. 326

N. 12 19 17 31 42 31

O. line 7, (1 point) the return type of get is Object for the raw ArrayList and Object does not have a length method. (1 point)

2. This was the hardest coding question on the test. Very limited on what you could use and necessary to keep track of indices in three different arrays. I liked this question a lot because there were so many valid, different solution to the problem.

Common mistakes:

- Using classes or methods not allowed. The restrictions were strict. You could not use ArrayList. You could not use GenericList (constructor, add, insert methods) unless you wrote those methods yourself.
- Not keeping track of the index correctly in each array
- Not putting elements in correct position in resulting array. (Saw lots of i, i + 1 instead of i * 2, i * 2 + 1)
- not updating the size instance variable of the object that invoked the method
- trying to create new E[newSize]. Must create array of objects and cast. (no points were taken off for this problem, but you must be aware of it.)

Suggested Solution:

```
public void interleave(GenericList<E> other) {
        int newCap = (this.size + other.size + 1) * 2;
        E[] temp = (E[]) new Object[newCap];
        int minSize = Math.min(this.size, other.size);

        // copy elements from each
        for(int i = 0; i < minSize; i++) {
            temp[i * 2] = this.values[i];
            temp[i * 2 + 1] = other.values[i];
        }

        // One of the two lists may have left over values.
        // Copy the values onto end of the result.
        int indexResult = minSize * 2;
        for(int i = minSize; i < this.size; i++) {
            temp[indexResult] = this.values[i];
            indexResult++;
        }
        for(int i = minSize; i < other.size; i++) {
            temp[indexResult] = other.values[i];
            indexResult++;
        }
        size += other.size;
        this.values = temp;
    }
```

General Grading Criteria (22 points) - using disallowed methods 1 - 14 based on severity

- ensure enough space / create new array - 3 points
- interleave equal number of elements from each array
    - bounds of iteration correct - 2 points
    - iteration correct - 3 points
    - add elements from this at correct spot - 3 points
    - add elements from other at correct spot - 3 points
- copy rest of elements from this - 2 points
- copy rest of elements from other - 2 points
- this.values assigned correctly - 2 points
- adjust this.size correctly - 2 points

3. A relatively easy problem. The most common problem was not ensuring the other MathMatrix was an integer multiple of the calling MathMatrix. Many different ways of checking this, although a bunch of students came up with a very clever solution that didn't require checking if result was integer. (You have to be aware of how integer division works. This causes all kinds of logic errors in your code if you do not.)

Common mistakes:

- Not verifying int division is same as floating point division. ways to do this:
  - use / and then use % to ensure no remainder
  - compare result of floating point division and integer division and ensure they are the same
- Not ensuring all elements are the same multiple

Suggested Solution:

```
    public boolean isIntegerMultiple(MathMatrixWithExtras other) {
        int magicMultiple = other.elements[0][0] / this.elements[0][0];
        int rem = other.elements[0][0] % this.elements[0][0];
        boolean intMultiple = rem == 0;
        int r = 0;
        while(r < this.elements.length && intMultiple) {
            int c = 0;
            while(c < this.elements[0].length && intMultiple) {
                int mult = other.elements[r][c] / this.elements[r][c];
                rem = other.elements[r][c] % this.elements[r][c];
                intMultiple = mult == magicMultiple && rem == 0;
                c++;
            }
            r++;
        }
        return intMultiple;
    }
```

alternate solution that used multiplication

```
    public boolean isIntegerMultiple(MathMatrixWithExtras other) {
      int scale = other.elements[0][0] / this.elements[0][0];
      for(int r = 0; r < elements.length; r++)
          for(int c = 0; c < elements[0].length; c++)
              if(this.elements[r][c] * scale
                      != other.elements[r][c])
                  return false;
      return true;
    }
```

General Grading Criteria (16 points)

- find scale factor correctly - 2 points
- check scale is int (or use multiplication technique correctly) - 4 points
- iterate through cells of 2d array of ints - 5 points
- check elements have same scale factor - 4 points
- return correct result - 1 point

4. Another relatively easy problem. Most students did very well on this problem.

Mistakes: (no common errors)

- hard coding a range of 30 instead of using parameter
- not find min and max correctly

```
public ArrayList<String> inRange(int rangeSize) {
    ArrayList<String> result = new ArrayList<String>();
    for(NameRecord name : namesList) {
        if(name.alwaysPresent()) {
            int max = 0;
            int min = MAX_RANK + 1;;
            for(int decade = 0; decade < NUM_DECADES; decade++)
                int rank = name.getRank(decade);
                if(rank > max)
                    max = rank;
                if(rank < min)
                    min = rank;
            }
            if(max - min <= rangeSize)
                result.add(name.getName());
        }
    }
    return result;
}
```

General Grading Criteria (16 points)

- create resulting ArrayList - 1 points
- iterator (or loop) through namesList - 3 points
- check alwaysPresent or check no zeros present - 2 points
- loop through ranks - 3 points
- find min and max correctly - 3 points
- check range within bounds and add to result - 3 points
- return correct result - 1 point

5. This was a hard test to write, because really all we have studied is implementing lists in an object based way and algorithm analysis. I really liked this question, because the answer was relatively simple, once you understood what the abstraction was. And that is a skill you must have to be a programmer. Certainly not the only skill, but a necessary skill. There were no restrictions. I meant to forbid any other classes or objects, but forgot to put that in the instructions. Some people tried to rebuild the dense list, but this is not trivial, and most students that tried, did not get it right. Also, the most efficient search would be a binary search, but for this test a linear search was perfectly acceptable.

Common mistakes:

- using values.length instead of elementsStored. The array may have had extra capacity. (specified in description of class.)
- not accessing into array, treating values as a list instead of an array

Suggested Solution

```
public E get(int pos) {
    // linear search for element at given
    // position. Binary search would be faster
    // but not expected (and no restriction on
    // effeciency
    for(int i = 0; i < elementsStored; i++) {
        // non default value?
        if(values[i].getPosition() == pos)
            return values[i].getData();
        // did I pass the position?
        else if(values[i].getPosition() > pos)
            return defaultValue;
    }
    return defaultValue;
}
```

General Grading Criteria (16 points)

- iteration through values attempted - 4 points
- iteration through values correct (bounds) - 3 points
- check to see if value at pos is no default - 4 points
- check to see if value at pos is default (prefer stopping early as shown in solution, but okay to search all the way through away on test, because no efficiency requirements.) - 4 points
- return correct value - 1 point