CS314 Spring 2014 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
Gacky or Gack - Code very hard to understand even though it works. (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.
RTQ - Read the question. Violated restrictions or made incorrect assumption.

1. Answer as shown or -1 unless question allows partial credit.
No points off for minor differences in spacing, capitalization, commas, and braces.
If """ included in output -1 first occurrence then error carried forward.

A. **6N + 7, 5 to 7N, 5 to 9**

B. **O(N^2)**

C. **O(N^2)**

D. **O(N)**

E. **88 seconds**

F. **100 seconds**

G. **any value between 10,000 seconds and 10,240 seconds inclusive**

H. **O(1)**

I. **O(NlogN), base 3 okay,**

J. **[12, 12, 17]** (missing commas or brackets okay, quotes **not** okay)

K. **C30 40** (extra space okay)

L. 1. **invalid**
   2. **valid**

M. 1. **valid**
   2. **invalid**

N. **BRED 50** (extra spaces okay)

O. **Compiler Error (syntax error okay. error NOT okay)**

P. **BRED RED 60** (extra spaces okay)

Q. **Lhelp** (extra spaces okay)

R. **Cannot add Strings to ar3 which can only hold Component objects. (or words to that effect)**

S. **The Switch class does not implement the compareTo method. (or words to that effect)**

T. **true C D**

2. Comments. Meant to be an easy question based on all the GenericList (array based list) examples we did in class, plus the number of GenericList questions on past exams. We saw a lot of different answers.

Common problems:

- using length instead of the array instead of size of list
- not dealing with the possible null values
- not finding and using min list size

Suggested Solution:

```
    public int lengthOfStartingMatch (GenericList<E> other) {
        int index = 0;
        boolean match = true;
        while(match && index < this.size && index < other.size) {
            E thisVal = this.values[index];
            E otherVal = other.values[index];
            if(thisVal == null)
                match = otherVal == null;
            else
                match = thisVal.equals(otherVal);
            // increment index only if still matching
            if(match)
                index++;
        }
        return index;
    }
```

17 points , Criteria:

- Check sizes, loop bounds correct. 3 points
- Access values correctly. 2 points. (other[i] instead of other.values[i] is -2)
- Stop checking on first mismatch: 2 points (return as soon as possible)
- Track number of matches correctly: 2 points
- Handles nulls correctly: 2 points
- .equals, not == on non-null objects: 2 points
- move index through list (or for loop): 3 points
- return value: 1 point

Disallowed methods -1 for Math.min, size(), or get()
-3 to -4 on other more severe problems
-2 if use iterator (no method available)
-2 if take out element at position 0 or assume list size > 0
-1 use this instead of E for data type of elements

Note, this does not work. Potential for Null Pointer Exception
if((values[i] == null && other.values[i] == null) || values[i].equals(other.values[i]))

Why not? if values[i] == null and other.values[i] != null first part is false so second part evaluates and NPE occurs.
3A. Comments: A very straight forward question. The only real issue was not handling UNRANKED. If a name is unranked it should be counted as above the cutoff. Most students did well on this question.

Suggested Solution:

```
public boolean anyRanksGreater(int cutoff) {
    for(int rank : ranks)
        if(rank > cutoff)
            return true;
    return false;
}

public boolean anyRanksGreaterAlt(int cutoff) {
    for(int i = 0; i < ranks.size(); i++)
        if(ranks.get(i) > cutoff)
            return true;
    return false;
}
```

6 points , Criteria:

- loop or for each loop, 2 point
- check if current rank (for each loop or get) is greater than cutoff, 1 point >= ok
- stop when found first rank greater than cutoff, 1 point
- access ArrayList element correctly with get or use for each loop, 1 point
- return answer, 1 point

DISALLOWED METHODS -1 PER
Not removing UNRANKED NAMES -1
-1 skipping any value in list
-1 if N^2 or worse
-1 if don't return true when NameRecord contains an Unranked
-1 if incorrect use of iterator.

3B. Comments: Meant to be an easy question based on the Baby Names assignment. Testing use of ArrayList and another class. The most efficient solution was to create two new ArrayLists. One to return and one to hold the values retained. This results in an O(N) solution. Most students implemented an O(N^2) solution. (removing from ArrayList is slow.) Even starting at the back will result in O(N^2) although it avoids skipping elements.

Common problems:

- skipping names in ArrayList logic error. (remove at position i, increment i)
- assuming an equals method in NameRecord (which remove based on object requires)
- assuming the iterator remove method returns a value. (the return type is void for the iterator)

Suggested Solution:

```
public ArrayList<NameRecord> remove(int cutoff) {
    ArrayList<NameRecord> result = new ArrayList<NameRecord>();
    ArrayList<NameRecord> retainedRecords
               = new ArrayList<NameRecord>();

    for(int i = 0; i < records.size(); i++) {
        NameRecord temp = records.get(i);
        if(temp.anyRanksGreater(cutoff))
            result.add(temp);
        else
            retainedRecords.add(temp);
    }
    records = retainedRecords;
    return result;
}
```

18 points, Criteria:

- create resulting ArrayList, 2 points
- loop through all NameRecords (for each okay), 3 points
- access NameRecord and call anyRanksGreater correctly, 3 points
- check result greater than cutoff, 2 points
- if result of method call is true add to result, 3 points
- efficiently handle remaining NameRecords (false from method, 4 points)
    - -2 if call remove method
- return result 1 point

Other deductions:

assume NameRecord has correct equals - 1

code that would result in commodification error with iterator -2

no attempt to remove values -6

new Iteartor() -2

no new list for removed NameRecords -2

No attempt to generate result -6

assuming iterator.remove() returns a value -2

3B. Comments: A tough problem. Had to deal with a new data structure and abstraction. Had to deal with an ArrayList of objects. Required doing something not seen in class or programming assignments. Efficiency was not graded so a lot of student just looped all the way through the ArrayList for each element. That was fine. The question mistakenly said the values were listed right to left. Per the example they are left to right. Most students either didn't notice this or went with the example. For the few students that moved right to left no points were lost.

Common problems:

- using a 2d array. Not allowed peer question
- Having outer loop go through the ArrayList of SMEntries. It is possible to make this work, but many students who tried this approach appended far too many zeros to the result
- Doing output. The method was toString and returned a String. toString methods do not return values. The question showed the result of displaying the result of toString not the result of calling toString. (toString would not produce any output itself.)
- Removing values from the ArrayList. toString is an assessor method. It would never alter the underlying object or data structure
- Treating the ArrayList like an array instead of a list. (Using the [] instead of get.)
- Not checking the current index into the ArrayList < list.size() -> leads to ArrayIndexOutOfBounds exception. AIOBE

Suggested Solution:

```
 public String toString()  {
    StringBuilder result = new StringBuilder();
    String forZeros = "0_";
    int indexInList= 0;
    for(int r = 0; r < numRows; r++) {
        for(int c = 0; c < numCols; c++) {
            if(indexInList < nonZeroValues.size()) {
                SMEntry temp = nonZeroValues.get(indexInList);
                if(temp.getRow() == r && temp.getCol() == c) {
                    result.append(temp.getVal());
                    result.append("_");
                    indexInList++;
                }
                else
                    result.append(forZeros);
            }
            else
                result.append(forZeros);
        }
        result.append("\n");
    }
    return result.toString();
}
```

21 points, Criteria:

- Correct nested loop structure, 3 points
- Correctly access ArrayList: 4 points

- Track index of next SMEntry in list (or correct loop to search): 4 points
- check current value in ArrayList: 3 points
- append default value correctly: 1 point
- append non default value correctly: 1 point
- avoid AIBOE: 3 points
- newlines added to StringBuilder correctly: 1 point
- return sb.toString(), 1 point

Common Deductions:
Rebuild entire array: -5
Add too many zeroes: -6
Destroys data structure: -5
Using iterator (not allowed per question): -1