CS314 Spring 2013 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 bad assumption.

1. Answer as shown or -1 unless question allows partial credit.
No points off for minor differences in spacing, capitalization, commas, and braces

Bonus Question: 32 bits + 1 point

A. $4N^2 + 4N + 4$, +/- 1 on each coefficient

B. $O(N^2)$

C. O(logN), base 2 okay

D. $O(N^2)$

E. $O(N^2)$

F. 20 seconds

G. $O(N^3)$

H. O(N)

I. 0.0022 seconds

J. [A, B, CC, C] (missing commas or brackets okay, quotes **not** okay)

K. runtime error **OR** A C D and then a runtime error

L. 1. invalid (or syntax error or compile error)
   2. valid

M. 1. valid
   2. invalid (or syntax error or compile error)

N. 1. invalid (or syntax error or compile error)
   2. valid

O. Lab: Amiga

P. true Lab: dual

Q. false Room: off

R. Compile Error

S. null 3 (runtime error **not** okay)

T. 4 0

2. Comments. Typical GenericList problem. Interesting with two GenricLists to deal with and keeping track of the two different indices. (where pulling from, where adding to)

Common problems:

- Not creating a GenericList or creating an array of Es, but not assigning it to GenericList
- Treating GenericList like an array, list[index]
- not updating size of GenericList as adding
- errors in indices in the two arrays
- go all the way to the end of the values array in the calling object, adding elements from extra capcity that are not part of the list
- calling methods or using classes not allowed (question did not allow anything besides given constructor with initial size and native array)
- writing a public add method that doesn't resize correctly. (AIOBE possible)

Suggested Solution:

```
public GenericList<E> sublist(int start) {

     int newSize = size - start;
     GenericList<E> result = new GenericList<E>((newSize + 1)* 2);

     for(int i = start; i < this.size; i++) {
          result.values[i - start] = this.values[i];
     }

     result.size = newSize;
     return result;
}
```

General Grading Criteria: 13 points

- create new Generic List correctly, call correct constructor: 2 points, lost this if just create array
- new list has sufficient capacity: 1 point (lose this if multiple resizes, efficiency), not necessary to have extra capacity
- Only add (size - start) elements to result: 2 points
- indices correct when adding [start, this.size) to [0, result.size) : 3 points [= inclusive,) = exclusive
- update size of result correctly: 2 points
- solution is O(N): 2 points
- return GenericList: 1 point (or stated method had void as return type), lose this if return array

Other common deductions:

- -3 if use result[i] instead of result.values[i] and result is a GenericList
- create public methods that don't work for clients (add methods that don't add), -2
- disallowed classes / methods -2 to -7 (in particular using ArrayList or calling add method was -7)

3. Comments: Probably the easiest question on the test. Saw lots of 2 or 3 branch if statements. easier I think to have the if, inside the nested loop.

Common problems:

- Not calling size method on the ArrayList vector
- treating ArrayList like an array

Suggested Solution:

```
public void multiplyByVector(ArrayList<Integer> vector) {

     if(vector.size() == coeffs.length
               || vector.size() == coeffs[0].length) {

          boolean matchesRows = vector.size() == coeffs.length;

          for(int r = 0; r < coeffs.length; r++)
               for(int c = 0; c < coeffs[0].length; c++)
                    if(matchesRows)
                         coeffs[r][c] *= vector.get(r);
                    else
                         coeffs[r][c] *= vector.get(c);
     }
}
```

```
Common alternate solution:

public void multiplyByVector(ArrayList<Integer> vector) {
     if(vector.size() == coeffs.length)
          for(int r = 0; r < coeffs.length; r++)
               for(int c = 0; c < coeffs[0].length; c++)
                    coeffs[r][c] *= vector.get(r);
     else if(vector.size() == coeffs[0].length)
          for(int r = 0; r < coeffs.length; r++)
               for(int c = 0; c < coeffs[0].length; c++)
                    coeffs[r][c] *= vector.get(c);
}
```

General Grading Criteria: 14 points

- determine if vectors aligns with row or column correctly: 3 points
- nested loop with correct bounds to access all elements: 3 points
- access elements from vector correctly (get method, size of list): 2
- scale element in 2d array by correct value from vector (multiply, correct index from vector): 2 points
- MathMatrix unchanged if vector not equal to number of rows or columns: 1 point
- O(N^2), no new arrays created: 2 points

4. Comments: A very interesting problem. The key was to verify the ArrayList ranks started with UNRANKED and then to skip through the ranks until the first rank NOT equal to UNRANKE occurred. At that point, the method can determine the answer. Lots of good, different solutions. It was okay to assume there was at least one ranked decade. (Not all elements == UNRANKED). Okay to assume every NameRecord has at least one decade that is ranked.

Common problems:

- Using 0 instead of the constant UNRANKED
- using the other constants instead of ranks.size() (No guarantee on number of ranks)
- Not stopping when first rank found. A LOT of solutions would return true on a name that was
  UNRANKED 300 UNRANKED 200
  if the cutoff was 250
- using a hard coded cutoff (250) instead of the parameter cutoff
- assuming 11 ranks

Suggested Solution:

```
public boolean trendy(int cutoff) {
    boolean result = ranks.get(0) == UNRANKED;
    if(result) {
        int index = 1;
        while(index < ranks.size() && ranks.get(index) == UNRANKED)
                index++;
        if(index < ranks.size())
            result = ranks.get(index) <= cutoff;
    }
    return result;
}
```

General Grading Criteria: 12 points

- determine if name record starts with one or more unranked decades correctly: 2 points
- use UNRANKED, not 0: 2 points
- find first ranked decade after unranked decades: 4 points
- check that first ranked decade is within cutoff: 2 points (-1 if < instead of <=)
- return result: 1 point
- use instance variable (ArrayList) correctly: 1 point

Other common deductions:

- Treating UNRANKED as a String -3
- Using BASE_YEAR or YEARS_PER_DECADE instead of ranks.size() (no need to use those constants on this method)
- Assume 11 decades -2 (hard coded 11)
- using disallowed classes / methods -2 each
- using 250 (or other hard coded number) instead of cutoff -2

5. Comments: I thought this was an interesting problem. (How many list questions can I ask.) The description tried to emphasize the order of elements in the Bag could change. This meant when you find an instance the EASIEST thing to do is to replace the reference of what you are removing with the LAST reference in the Bag (1 operation), not shift all the elements down 1 (on average N/2) operations. Granted, this does not change the order, but the question specified to have a small T(N). Starting from the end instead of the start does not guarantee an improvement. (It is just as likely the first occurrence is in the first half of the list as it is in the second half.) noe = numberOfElements

Common problems:

- going to the length of the array instead of the numberOfElements (extra capcity possible)
- creating a new array which is inefficient in terms of time and space for this problem
- nulling out the removed element and not doing a swap. (Question specified the N elements of the Bag are stored in the FIRST n elements of the array.)
- shifting elements instead of swapping last reference
- calling a non-existent remove method on an array, elements.remove(3). Elements is an array. The only methods it has are those from Object.
- using == instead of .equals to check equality of objects
- removing ALL occurrences of the target object, not one
- not updating numberOfElements (noe)
- not nulling out last elements (possible memory leak because object should be garbage, but isn't)
- continue to loop even when an occurrence found

Suggested Solution:

```
public boolean removeSingleOccurrence(Object tgt) {
      for(int i = 0; i < numberOfElements; i++) {
            if(tgt.equals(elements[i]) {
                  // found one!
                  numberOfElements--;
                  elements[i] = elements[numberOfElements];
                  elements[numberOfElements] = null;
                  return true;
            }
      }
      // never found tgt
      return false;
}
```

General Grading criteria: 11 points

- search to find tgt, correct number of elements 2 points (loop)
- use equals method, not ==. 2 points
- stop when find the first occurrence, 2 points
- swap reference of last with found, 2 points
- null old last reference, 1 point
- decrement numberOfElements if found, 1 point
- return correct result, 1 point

Other common deductions:

- remove method (-6), infinite loop (-3 to -6), go to length not numberOfElements (-2), create new array (-2), remove all equal values not just 1 (-3)