CS314 fall 2016 Exam 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.)

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.

A. $1.5N^3 + 5N^2 + 5N + 5$ +/- .5 on N^3 term, +/- 1 on each other coefficient and the constant

B. $O(N^3)$

C. O(NlogN) base 3 okay

D. $O(N^2)$

E. $O(N^2)$

F. $O(N^2)$

G. O(N)

H. 32 seconds

I. 250,000 seconds (assume 2^20 = 1,000,000) or 2^20 *.25 or 2^18

J. 42 seconds

K. $O(N^4)$

L. 20 seconds if assume 10,000. 2000 seconds if assume 1,000 or no assumption stated

M. {-5=3, 0=-2, 1=3, 3=9, 7=12} No brackets okay, order matters

N. 3 7 then runtime error occurs

O. 1. invalid (Student abstract) 2. invalid (Undegrad not subclass)

P. 1. valid 2. invalid (Object not subclass of GradStudent)

Q. 10-15 10-15

R. MWF3 5

S. 10-10 10-15

T. MWF 8-5 12-2

2. Comments. Simplest question on the exam.

Common problems:

- going through length of array instead of size. This is inefficient and may lead to incorrect result if val is null
- using == instead of .equals to check equality of objects
- not dealing with nulls correctly.
- checking for null via val.equals(null). If val is null this, of course, causes a null pointer exception

```
public int frequency (E val) {
     int matches = 0;
     for (int i = 0; i < size; i++) {
          if (val == null && container[i] == null) {
               matches++;
          } else if (val != null && val.equals(container[i]) {
               matches++;
          }
     return matches;
}
```

16 points , Criteria:

- loop through valid portion of array , 6 points
  - -4 if limit is not size instance variables
- handle null case correctly, 4 points
- call equals correctly, not == for non nulls, 4 points
- create and return correct result, 2 points

Usage errors:
using disallowed methods, -6

3. Comments: A question involving multiple Generic Lists. This method is in the GenericList class so we have access to private fields of any GenericLists in scope such as other and result.

Common problems:

- calling disallowed methods. Note, add was not allowed on this question per the instructions.
- creating public methods that grant access to instance variable such as container. (This is very poor design.)
- not using the getArray method
- not setting size variable of result
- not ensuring enough capacity in result
- not determining min size based on smaller of the two lists

```
public GenericList<E> getNonMatchingPairs(GenericList<E> other) {

      // create result large enough to hold all pairs
      GenericList<E> result = new GenericList<E>();
      result.container = getArray(size + other.size);

      int minSize = size < other.size ? size : other.size;

      for (int i = 0; i < minSize; i++) {
            E o1 = container[i];
            E o2 = other.container[i];
            if (!o1.equals(o2)) {
                  // non matching so add to result
                  result.container[result.size] = o1;
                  result.container[result.size + 1] = o2;
                  result.size += 2;
            }
      }
      return result;
}
```

20 points, Criteria:

- create result with enough space (or implement resize method) , 2 points
- determine min size, 3 points
- check pair does not match (equals) correctly, 3 points
- add elements to correct spots in result, 5 points
- update result size correctly, 2 points
- return result, 1 point

Other deductions:

- using add method without implementing (-8)
- O(N^2) -4
- incorrectly shadow this.size, -2
- accessing list variable as if it is an array, other[index] instead of other.container[index], -4

4. Comments: Meant to be an easy problem.

Common problems:

- not checking matrix square correctly
- not returning answer as soon as known

Suggested Solution:

```
public boolean isStrictlyDiagonallyDominant() {
      if (myCells.length != myCells[0].length) {
            return false; // not square
      }

      for (int r = 0; r < myCells.length; r++) {
            int rowSum = 0;
            for (int c = 0; c < myCells.length; c++) {
                  if (r != c) {
                        rowSum += myCells[r][c];
                  }
            }
            if (myCells[r][r] <= rowSum) {
                  return false;
            }
      }
      return true;
}
```

16 points, Criteria:

- check not square correctly and return false if not, 3 points
- nested loop to check cells in Matrix, 4 points
- correctly sum all elements on row except element on main diagonal, 4 points
- return false as soon as answer know, 4 points
- return correct answer if true, 1 point

Others:
using an array to store sums, -2 (space)

5. Comments: A simple map problem.

Common problems:

- not accessing keys correctly in map
- creating lots of unnecessary ArrayLists (every word, instead of every new word)
- inefficient solutions.
- not creating TreeMap

Suggested Solution:

```
public static Map<String, ArrayList<Integer>> createIndexMap(String[] words) {
     Map<String, ArrayList<Integer>> result
                            = new TreeMap<String, ArrayList<Integer>>();

     for (int i = 0; i < words.length; i++) {
         ArrayList<Integer> value = result.get(words[i]);
         if (value == null) {
             value = new ArrayList<Integer>();
             value.add(i);
             result.put(words[i], value);
         } else {
             value.add(i); // have reference so don't need to put back
         }
     }
     return result;
 }
```

16 points, Criteria:

- create resulting TreeMap, 2 point
- loop through words, 2 point
- for first occurrence create and add ArrayList with first index, 5 points
- for later occurrences add index to existing key, 5 points
- return result 2 point

Other:

6. Comments: Hardest question on the exam. Lots of different cases to worry about. Lots and lots of different approaches to the problem.

Common problems:

- not updating positions of non default items correctly
- not shifting non default items correctly
- not updating size of list or number of elements stored

Suggested Solution:

```
public E remove(int pos) {
      final int NOT_FOUND = 0;
      final int DEFAULT = 1;
      final int NON_DEFAULT = 2;
      int status = NOT_FOUND;
      // determine if removing non default value
      int index = 0;
      while (index < elementsStored && status == NOT_FOUND) {
            int position = values[index].getPosition();
            if (position == pos) {
                  status = NON_DEFAULT; // found spot, it's a non default value
            } else if (position > pos) {
                  status = DEFAULT; // must be removing default value
            } else {
                  index++; // keep looking
      }
      if (status == NOT_FOUND)
            status = DEFAULT; // never found position
      E result = status == DEFAULT ? defaultValue : values[index].getData();

      // update positions of any elements after one being removed
      for (int i = index; i < elementsStored; i++)
            values[i].setPosition(values[i].getPosition() - 1);

      // shift elements if removing a non default item
      if (status == NON_DEFAULT) {
            elementsStored--;
            for (int i = index; i < elementsStored; i++)
                  values[i] = values[i + 1];
            values[elementsStored] = null; // prevent memory leak
      }

      // removed an element
      sizeOfList--;
      return result;
}
```

16 points, Criteria:

- find position of element to remove in values, 3 points
- determine if default or non default value, 2 points
- update position of non default items correctly, 4 points
- shift elements in values if necessary, 4 points
- update elementsStored if necessary, 1 point
- update size, 1 point
- return correct result, 1 point

For questions O - T, consider the following classes. You may detach this sheet from the test.

```
public abstract class UTPerson {
      private int hoursLimit;
      private int bookLimit;

      public UTPerson(int hours, int books) {
            hoursLimit = hours;
            bookLimit = books;
      }

      public abstract String getPrimeTime();

      public int getHours() {    return hoursLimit;    }

      public String toString() {
            return bookLimit + "-" + getHours();
      }
}

public class Staff extends UTPerson {

      public Staff() {    super(3, 5);      }

      public String getPrimeTime() {   return "8-5";    }
}

public abstract class Student extends UTPerson {

      public Student(int hours) {   super(hours, 10);  }

      public int getHours() {    return 15;       }
}

public class Undergrad extends Student {

      public Undergrad() {    super(20);    }

      public String getPrimeTime() {   return toString(); }
}

public class GradStudent extends Student {

      public GradStudent(int hours) {    super(hours); }

      public String getPrimeTime() {   return "12-2"; }

      public int getHours() {    return 10;       }
}


public class Faculty extends UTPerson {

      private int classes;
```

```
	public Faculty(int classes) {
		super(0, 30);
		this.classes = classes;
	}

	public String getPrimeTime() {	return "MWF";  }

	public int getHours() {	return classes * 5;	}

	public int officeHours() {	return classes * 3;	}

	public String toString() {
		return getPrimeTime() + officeHours();
	}
}
```