CS314 Spring 2015 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.)
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.

A. **$3N^2 + 6N + 4$, $\pm$ 1 on each coefficient and the constant**
B. **$O(N^2)$**
C. **O(NlogN)** // base 2 okay
D. **$O(N^2)$** // even though starts at end what if we remove first half?
E. **2000 seconds**
F. **10 seconds**
G. **$O(N^3)$** // when data double time goes up by factor of 8
H. **[K, P, P, X]** // missing commas or brackets okay
I. **5 8 -2 [8, 0, 2, 4]** // missing commas or brackets okay
J. **CC CC A C**
K. **M 11**
L. 1. **valid**
   2. **invalid** // LabClass not a sub class of CrossListed
M. 1. **invalid** // cannot instantiate interface types
   2. **invalid** // AcademicClass not a sub class of CrossListed
N. **I 21**
O. **GOV60** // extra spaces okay
P. **ECO 1** // calls getStudents() based on dynamic type
Q. **true**
R. **false true**
S. **runtime error** // ClassCastException or Exception okay
T. **true C25** // extra spaces okay

2A. Comments. Meant to be an easy problem using ArrayLists.

Common problems:

- base decade does not play any role in the this solution
- not handling the stored 0 case (convert to 1001)
- minor logic errors

Suggested Solution:

```
public int[] getBestAndWorst() {
    // assume at least one rank
    int min = ranks.get(0);
    if(min == 0)
            min = 1001;
    int max = min;
    for(int i = 1; i < ranks.size(); i++) {
        int rank = ranks.get(i);
        if(rank == 0)
            rank = 1001;
        if(rank < min)
            min = rank;
        if(rank > max)
            max = rank;
    }
    return new int[]{min, max};
}
```

10 points , Criteria:

- initialize min and max correctly, okay for max = 0, min = 1001, // 1 point
    - okay to use array of length 2 for this
- loop through array list correctly 3 points
- adjust 0 to 1001 correctly, 2 points
- check for new min and max correctly, 2 points
- create array of length 2 and place min rank (best) in first pos, max rank (worst) in second pos
- return result correctly 1,point (must be an array)

2B. Comments: Also meant to be an easy problem. One tricky item to make solution more efficient in most cases.

Common problems:

- not using getRank to stop as soon as one rank out of range. (Calling get bestAndWorst does unnecessary work in many cases.
- not accessing nameList correctly

Suggested Solution:

```
public ArrayList<NameRecord> getConstrained(NameRecord source) {
    int[] bestAndWorst = source.getBestAndWorst();
    int bestSource = bestAndWorst[0];
    int worstSource = bestAndWorst[1];
    ArrayList<NameRecord> result = new ArrayList<NameRecord>();
    for(int i = 0; i < nameList.size(); i++) {
        NameRecord nr = nameList.get(i);
        boolean inRange = true;
        int decade = 0;
        while(inRange && decade < NUM_DECADES) {
            int rank = nr.getRank(decade);
            if(rank == 0)
                rank = 1001;
            inRange = bestSource< rank && rank < worstSource;
            decade++;
        }
        if(inRange)
            result.add(nr);
    }
    return result;
}
```

10 points , Criteria:

- get minMax for source, 1 point
- create result, 1 point
- loop though nameList, 1 point
- check current rank not outside range correctly, 5 points (-1 if call method)
- if constrained add to result, 1 point
- return correct result, 1 point

3. Comments: This was a tough problem. There were very few methods in the GenericList class you were allowed to use and you had to keep track of three different GenericLists

Common problems:

- going through entire container array instead of active (size) portion
- assuming GenericList was Iterable (given header did not implement Iterable)
- assuming add, get, and size methods available
- ineffecient solutions that did not stop as soon as watch found
- using == instead of .equals
- not ensuring enough capacity in result.conatainer
- not updating result.size correctly
- adding too many items. Often occurred when add was inside inner loop
- no inner loop, in correct logic
- confusing GenericLists for arrays and vice versa

```
public GenericList<E> inOtherListOnly(GenericList<E> other) {
    GenericList<E> result = new GenericList<E>();
    result.container = getArray(other.size);
    for(int i = 0; i < other.size; i++) {
        E current = other.container[i];
        int indexThis = 0;
        boolean distinct = true;
        while(distinct && indexThis < this.size) {
            distinct = !current.equals(this.container[indexThis]);
            indexThis++;
        }
        if(distinct) {
            result.container[result.size] = current;
            result.size++;
        }
    }
    return result;
}
```

20 points, Criteria:

- create result: 1
- make resulting container large enough, 2
- loop through correct elements of other, 2
- check current from other not in this.container, 3
- equals called correctly, 3
- if not in this.container add to result correctly, 3
- does not confuse list with array and vice versa, 3
- update result.size correctly, 2
- return correct result, 1

Other deductions:

- iterable: -5
- add method: -5
- length of array not size: 3
- add too many: -4

4. Comments: A tough problem. Lots of abstraction going on. Had to keep track of current location in the outer container.

Common problems:

- slow check in hasNext() instead of updating current position
- not decrementing outer bag size when remove
- using size of outer Bag instead of length of array in hasNext check. (index vs. number of items returned.)
- not searching for next non null in array

Suggested Solution:

```
    private class BagIterator implements Iterator<E> {
        private int numToReturn;
        private int index;
        private boolean removeOk;

        private BagIterator() {
            numToReturn = size;
        }

        public boolean hasNext() {
            return numToReturn > 0;
        }

        public E next() {
            if(!hasNext()) throw new NoSuchElementException();
            // find the next spot;
            while(container[index] == null) {
                index++;
            }
            removeOk = true;
            numToReturn--;
            E result = container[index];
            index++; // move past current item;
            return result;
        }

        public void remove() {
            if(!removeOk) throw new IllegalStateException();
            removeOk = false;
            container[index - 1] = null; // remove previous item
            size--;
        }
    }
```

15 points, Criteria:

- instance var for position, 2
- instance var for removeOK, 1
- constructor if necessary, 1
- hasNext() correct, 3
- next() correct, 4
- remove(), 3 (must be O(1) or -1), decrements size
- one of hasNext() and next() O(N), other O(1), 1