CS314 Fall 2017 Exam 2 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. 5

B. 741!-26810

C. 13

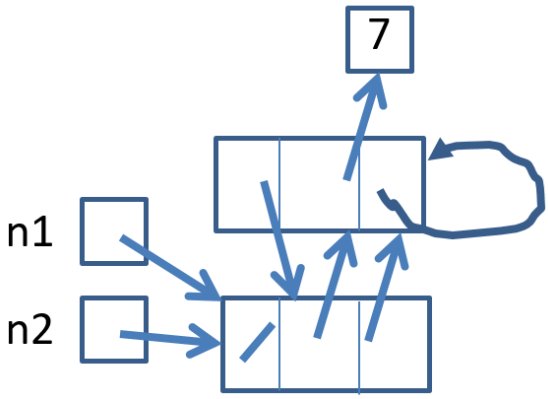
D. 42

E. 1000 or 1024 seconds

F. 12 seconds (O(N) with LinkedList)

G. 500 seconds (O(N^2) with ArrayList)

H. (syntax error or drawing) ------------->



I. O(N^3)

J. 110

K. 32 seconds (pop is O(N) based on implementation)

L. 44 seconds

M. 20 seconds

N. 4 2 2 6 2 4 8

O. 0 then a runtime error occurs (ClassCastException)

P. 4

Q. 12 0 -5 20 26 37 25 17

R. 3

S. .002 seconds (Method is O(N) due to small number of values)

T. 16 seconds (Method is O(N^2))

Commonly missed questions: D, H, K, O, and R

2. Comments. Meant to be an easy question involving linked lists. Question involves moving through linked lists, but not actually changing them.

Common Problems:

- not checking out of bounds at start and just returning false
- using nonexistent methods
- always O(N) instead of stopping at stop index

```
public boolean rangeIsEqual(LinkedList314<E> other, int start, int stop) {
    if (stop > this.size || stop > other.size)
        return false;
    Node<E> n1 = this.first;
    Node<E> n2 = other.first;
    for (int i = 0 ; i < start; i++) {
        n1 = n1.next;
        n2 = n2.next;
    }
    int numCheck = stop - start;
    for (int i = 0; i < numCheck; i++) {
       if (!n1.data.equals(n2.data))
           return false;
        n1 = n1.next;
        n2 = n2.next;
    }
   return true;
}
```

20 points , Criteria:

- if stop out of bounds, return false immediately, 1 point (efficiency)
- 2 temp nodes, 1 point
- move both temp nodes to start node, 5 points (off by one -2)
- loop with correct bounds for checking data in nodes equal, 3 points
- check data in nodes (not nodes themselves), 3 points
- use equals correctly, 2 points
- return false on first mismatch, 2 points
- move down list correctly when checking data, 2 points
- return correct answer if same, 1 point

Other penalties:

- using disallowed methods, -8 (can be less for severity)
- destroy either list, -8
- recursive solution with N stack frames -5 (space efficiency)
- Null Pointer Exceptions -4
- use getData / getNext -2 (just once)
- off by one errors, -2

3. Comments: Having to change a linked list.

Common problems:

- not checking if start == stop and if so doing nothing, NO-OP (no operation)
- not stopping at node before node at position start
- not dealing with moving first if start == 0
- not updating size. I thought these were easy points and it was mentioned in the post condition

```
public void removeRange(int start, int stop) {
    if (start < stop) {
        int numRemoved = stop - start;
        if (numRemoved == size)
            // clear whole list.
            first = null;
        else if (start == 0) {
            // special case, removing first node
            for (int i = 0; i < numRemoved; i++) {
                first = first.next;
            }
        } else {
            // general case, move to node before start
            Node<E> n = first;
            for (int i = 1; i < start; i++)
                n = n.next;
            // now remove the right number of nodes
            for (int i = 0; i < numRemoved; i++)
                n.next = n.next.next;
            // ?? Special case for stop == size ??
        }
        size -= numRemoved;
    }
}
```

20 points, Criteria:

- do nothing if start == stop, 2 points
- special case handled correctly when start == 0, moving first, 4 points
- general case, move to node before start correctly, 4 points
- general case, remove correct number of nodes, 6 points
- update size correctly, 4 points

Other penalties:

- using disallowed methods, -8 (can be less for severity)
- recursive solution with N stack frames -5 (space efficiency)
- NullPointerException, -6
- infinite loop due to not moving, -6

4. Comments: Meant to be a simple question involving moving through a binary tree without changing it.

Common problems:

- returning early
- not handling case when tree is empty
- summing 1 or 3 elements instead of parent and child

```
public int parentChildPairsWhoseSumEquals(int tgt) {
    return help(root, tgt);
}

private int help(BNode n, int tgt) {
    if (n == null || (n.left == null && n.right == null))
        return 0; // leaf or empty
    int result = 0;
    if (n.left != null && (n.data + n.left.data == tgt))
        result++;
    if (n.right != null && (n.data + n.right.data == tgt)){
        result++;
    }
    result += help(n.left, tgt) + help(n.right, tgt);
    return result;
}
```

20 points, Criteria:

- create helper method and start at root node, 2 points
- correct base case for null or leaf (can handle elsewhere), 5 points
- non base case, check if left child exists and if sum of parent and child equals target, 2 points (Can handle this possible with parameter)
- non base case, check if right child exists and if sum of parent and child equals target, 2 points
- correct recursive calls to left and right child (can skip if don't exist), 5 points
- return correct value, 4 points (lose if try to store in a parameter only)

Others:

- altering tree, -8
- new data structure, -8
- early return -5
- not handling case when root == null, empty tree, -4
- not summing two correct elements, -4

5. Comments: An interesting recursive problem. Note, the choices were what truck to put an item on. The question said no other classes, and this included creating new arrays. Very easy to do this in place.

Common problems:

- not making success base case all capacity used
- not returning as soon as answer known (after undoing changes)

- not checking option of leaving item off of all trucks
- no base case for all items tried
- nested loop in recursive method. Just one loop for truck options
- not filling in arguments for method call (question said to complete arguments, but not to add any other code)

```
public static boolean canShip(int[] trucks, int[] items) {
    return canShipHelp(trucks, items, 0);

private static boolean canShipHelp(int[] trucks, int[] items, int index) {
    if (allZeros(trucks))
        return true;
    else if (index == items.length)
        return false;  // never found a solution
    int item = items[index];
    for (int i = 0; i < trucks.length; i++) {
        if (trucks[i] >= item) {
            // item will fit on truck i
            trucks[i] -= item;
            // did we do it?
            boolean solved = canShipHelp(trucks, items, index + 1);
            // take item off truck
            trucks[i] += item;
            // if we solved it, stop!!!
            if (solved)
                return true;
        }
    }
    // one last chance, don't put item on any truck!
    return canShipHelp(trucks, items, index + 1);
}

// saw a very nice solution that determined total capacity and
// sent and altered that instead of contsantly summing
private static boolean allZeros(int[] trucks) {
    for (int capacityLeft : trucks)
        if (capacityLeft > 0)
            return false;
    return true;
}
```

20 points, Criteria:

- call helper with two arrays and index of current item, 2 points
- base case when solved problem (no capacity left), 3 points
- base case when no items left, 2 points
- recursive case, loop through all the trucks, 3 points
- only put item on truck if capacity exists, 2 points
- recursive call if item fits, 3 points
- remove item from truck after result of recursive call, 2 points (necessary as array must be unchanged)
- return true when solution found, 2 point
- return false after trying all choices, 1 points

Other:

- early return, -7, other objects including arrays, -6, not stopping when answer known, -4, stack overflow -7