CS314 Fall 2019 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.

MCE - Major Conceptual Error. Answer is way off base, question not understood.

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. (Statements in parenthesis not required, only for explanation of answer to students.)

A. `1133`

B. `11113232`

C. `38`

D. `0.32 seconds`

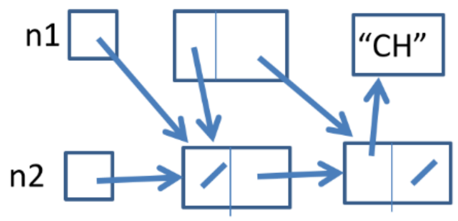
E. `16 (method simply sums up elements in array)`

F. `7`

G. `syntax error OR compile error (Cannot instantiate abstract classes)`

H. `-------------------------------------->`

`OKAY IF GARBAGE NODE NOT SHOWN`



I. `4 seconds (method is O(N))`

J. `45 seconds (method is O(N`^{2}`) under given conditions`

K. `{0=AI, 2=null, 4=CS} (minor differences in spacing and brackets okay)`

L. `O(N)`

M. `80 seconds`

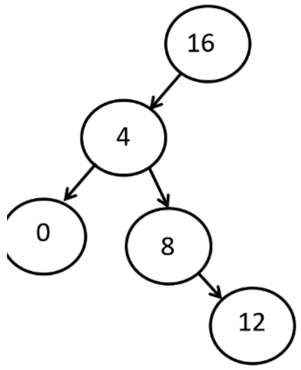
N. `ArrayList is faster due to O(1) for accessing elements compared to O(N) for accessing elements in LinkedList. (or words to that effect)`

O. `O(N`^{2}`logN) (base 2 okay)`

P. `RTUPO`

Q. `O(N)`

R. `--------------------------------------------------->`



S. `0 12 8 4 16`

T. `3`

2. Comments. O(N) approach: Determine the difference in size of the lists. Start at the first node in the larger list and traverse a number of nodes equal to the difference in sizes. Now move both temporary references. If the temp references are every the same, the lists converge. Nina pointed out (and many students saw) a simple O(N) solution. Just go to the last node in each list and check if they are the same node. This is always O(N), but does some unnecessary work in some cases.

```
    public boolean listsConverge(LinkedList314<E> other) {
        if (thisSize == 0 || otherSize == 0) {
            return false; // empty can't converge
        }
        int diff = this.size - other.size; // assume this is bigger
        Node<E> tempBig = this.first;
        Node<E> tempSmall = other.first;
        if (diff < 0) { // other bigger
            diff = -diff;
            tempBig = other.first;
            tempSmall = this.first;
        }

        // move the temp in the bigger list
        while (diff > 0) {
            tempBig = tempBig.next;
            diff--;
        }

        while (tempBig != null) {
            if (tempBig == tempSmall) {
                return true; // Same node! Lists converge.
            }
            tempBig = tempBig.next;
            tempSmall = tempSmall.next;
        }
        return false;
    }
```

22 points, Criteria:

- explanation is clear and concise, code matches explanation, 3 points (-1, -2 possible)
- return immediately if either list is size 0, 1 point
- temporary node references for this and other, 2 points
- move temp node in bigger list difference in size times correctly, 3 points
- main loop that goes until one or both temp nodes store null, 2 points
- correctly check if nodes equal and return true if so, 3 points
- move temp nodes correctly, 4 points
- stop when first shared node found, 3 points (-2 if all the way to end of each list)
- return false if do not converge and not empty, 1 point

Other deductions:

- using disallowed methods: size() -2, get() -4, others -> vary
- create new data structures, -7 alter either list, -7 create single new nodes, -2
- check data not nodes, -3 double work, check both ways, -1 start at front and just loop -9
- start at front and just loop in tandem, the MCE, tandem -9 NPE, -3

O(N^2) approach: Have a temporary node reference in each list. For each node in this list, traverse all nodes in the other list and see if any of them are equal to the node in this list. If so, stop, otherwise move the temporary reference in this list to the next node. This is O(N^2)

Many stduents were also concerned with finding the smaller list and make that the outer loop. This insured O(1) if either list is empty, but otherwise had little effect on the worst case.Still traversing N * M nodes either way in the worst case.

```
public boolean listsConverge2(LinkedList314<E> other) {
    if (this.size == 0 || other.size == 0) {
        return false;
    }
    Node<E> tempThis = this.first;
    while (tempThis != null) {
        Node<E> tempOther = other.first;
        while (tempOther != null) {
            if (tempOther == tempThis) {
                return true;
            }
            tempOther = tempOther.next;
        }
        tempThis = tempThis.next;
    }
    return false;
}
```

22 points, Criteria:

- explanation is clear and concise, code matches explanation, 3 points (-1, -2 possible)
- O(N^2) approach, -5 efficiency
- return immediately if either list is size 0, 1 point
- temp Node variables for each list, 1 point
- outer loop based on size or until outer temp is null, 3 points
- reset inner temp node to start of other list (could be this) for each iteration, 3 points
- inner loop based on size or until inner temp is null, 2 points
- correctly check if nodes equal and return true if so, 3 points
- move temp nodes correctly, 5 points
- return false if do not converge and not empty, 1 point

Other deductions:

- not resetting the node to the list traversed in the inner loop, -3
- using disallowed methods: size() -2, get() -4, others -> vary
- create new data structures, -7
- alter either list, -7
- create single new nodes, -2
- checking data instead of nodes, -3
- just loop in tandem from start of lists, MCE tandem error, -9
- check each way, double the necessary work, -1

3. Comments: The map question! Biggest issues were: using an iterator and then remonving via the map. This results in a ConcurrentModificationError. (create a new map or use the iterator remove, which does remove from the map even though the iterator is going over the keyset), not using a TreeMap for the final result. (names had to be in sorted order), not dealing with pairs of elements in the original array correctly, trying to increment or decrement Integer objects. This is syntactically allowed due to unboxing, but Integers are immutable, so the new value must be put in the map or it is discarded.

```
public static Map<String, Integer> getTeams(String[] games, int cutoff) {
    HashMap<String, Integer> allTeams = new HashMap<>();
    for (int i = 0; i < games.length; i += 2) {
        String winner = games[i];
        update(allTeams, winner, 1);
        String loser = games[i + 1];
        update(allTeams, loser, -1);
    }
    TreeMap<String, Integer> result = new TreeMap<>();
    for (String key : allTeams.keySet()) {
        int diff = allTeams.get(key);
        if (diff >= cutoff) {
            result.put(key, diff);
        }
    }
    return result;
}

private static void update(Map<String, Integer> map,
        String team, int add) {
    Integer diff = map.get(team);
    if (diff == null) {
        diff = new Integer(add);
    } else {
        diff = new Integer(diff + add);
    }
    map.put(team, diff);
}
```

22 points, Criteria:

- correctly loop through array of results, 1 point
- correctly increase winner's value in map by +1, 4 points (-4 if NPE possible)
- correctly decrease loser's value in map by -1, 4 points
- after processing array: loop through all results keyset correctly, 3 points
- determine if team's win difference is above cutoff correctly, 3 points
- add to result correctly, 4 points (lose if use map.remove during iteration. CME) Explicit iterator remove from keySet okay or Iterator.remove())
- return TreeMap, 2 points (must be in order)
- return result, 1 point

Others:

- using disallowed methods, -4
- worse than O(NlogN), -6
- altering games array, -5
- off by one errors, -2
- CME -> ConcurrenModificationError, -4

if first loop increments by 1 instead of 2, and no math, -4
Integer immutable logic error, -4
int == null check (syntax error), -2

4. Comments: A relatively simple tree problem. The restriction on the number of parameters for the helper was meant to prevent students from adding an unnecessary accumulator variable which typically leads to a logic error. Several students misunderstood the question and though we were looking for a path of nodes equal to the target. This was a past test question, but here we are looking at groups of nodes, parent value + children's values if they exist. The target value should never change in recursive calls.

```
public int numNodeAndChildrenEqualValue(int tgt) {
     return helper(root, tgt);
}

private int helper(BNode n, int tgt) {
     int result = 0;
     if (n != null) {
          int temp = n.data;
          if (n.left != null)
               temp += n.left.data;
          if (n.right != null)
               temp += n.right.data;
          if (temp == tgt)
               result++;
          // could incorporate recursive calls with null checks
          result += helper(n.left, tgt) + helper(n.right, rgt);
     }
     return result;
}
```

14 points, Criteria:

- helper method with current node and tgt, 1 point
- correctly handle case of empty tree (can be base case null check or special case in original method), 1 point
- correctly check if children exist, 2 points
- correctly check if current node's value and child nodes' values (if they exist) equal target, 3 points
- correctly make recursive calls, 4 points
- add result of recursive calls to local variable, 2 points
- return correct result, 1 point

Other:

- assuming we are looking for paths of nodes values that equal target, -7 MCE
- hard coding target value, -4
- more than 2 parameters to helper, -4
- referring to left or right without n. ,-2
- null pointer exceptions possible, NPE, -4
- .equals on int (doesn't autobox), -2
- -2 not matching parameters correctly, -2
- altering tree, -6
- early return, -7
- any other data structures -4
- disallowed methods, -4
- using disallowed classes and /or objects (arrays, ArrayLists), -5

5. Comments: Interestingly, we DON'T need a helper method. This is actually a graph problem. This was a fairly straight forward recursive backtracking problem. The biggest issue was not setting the current start building to visited right away as this can lead to paths that go through the start building twice, once at the start and later going back through.

```
public static boolean pathExists(String start, String end, int links,
        BuildingMap map) {
    if (start.equals(end)) {
        return links == 0;
    } else if (links > 0) {
        // now visiting start
        map.setVisited(start, true);
        int nextLinks = links - 1;
        String[] connections = map.connected(start);
        for (int i = 0; i < connections.length; i++) {
            String nextBuilding = connections[i];
            if (!map.visitedStatus(nextBuilding)) {
                boolean solved = pathExists(nextBuilding, end,
                                        nextLinks, map);
                if (solved) {
                    // undo change
                    map.setVisited(start, false); // GACK
                    return true;
                }
            }
        }
        // maybe another through this building?
        map.setVisited(start, false); // GACK
    }
    return false;
}
```

22 points, Criteria:

- success base case, 2 points
- failure base case of links is < 0, 2 points (Efficiency)
- recursive case items:
- set current node (start to visited), 2 points
- loop through choices, 2 points
- check if current choice visited or not, 2 points
- if current choice not visited already (could be alternate base case at top of method), make correct recursive call, 6 points
- if call succeeds, undo visited and return true, 3 points
- if all choices fail, undo visited, 2 points
- if all choices fail, correctly return false, 1 point

Other:

- not setting start to visited right away, -2
- early return -6
- creating other data structures, besides the String[] returned from the building map
- links-- in the parameter list, argument is just links in this case (-4)
- not using .equals for Strings, -2
- never undoing visitied, -4
- multiple links decrements in method, (logic error), -4