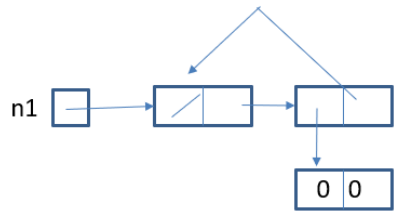
CS314 Spring 2024 Exam 2 Solution and Grading Criteria.

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. Lack of Zen.)
LE - Logic Error in code.
MCE - Major Conceptual Error. Answer is way off base, question not understood based on answer provided.
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.
EFF - Efficiency. Order is worse than expected or unnecessary computations done.

1. Answer as shown or -2 unless question allows partial credit.
First use of quotes in output is wrong, then error carried forward.
No points off for minor differences in spacing, capitalization, commas, and braces.
Text in parenthesis not required. It is simply grading guidance and / or a brief explanation for answer.

A. 8
B. 10
C. 9
D. 31
E. A
F. {A=9, E=7, G=9, N=5, S=6}
(Minor differences in spacing and separators okay)
G. Given the abstract method the class must also be declared abstract.
H. 8 seconds (code is O(N^3))



I.
J. B
K. 20 seconds (Code is O(N^2) due to removing first element and then shifting all remaining elements.)
L. 10 seconds (Code is O(N) with given resize.)
M. 0
N. B
O. 20 seconds
P. insertion sort (1 point)
quicksort (1 point)
Q. 3 and 7, 1 point per, -1 per any other,
(3, cannot create object of interface type. 7, declared type is Object)
R. mergesort (to maintain stability of objects)
S. 4 seconds
T. 8 seconds (removing the last element of a singly linked list is O(N) even with reference to last node.)
U. 4
V. K D M J O P
W. P M D K O J
X. 3 (Add left child to M, right child to D, and right child to O)
Y. 2

2. Comments: A straightforward question. Just had to move through the list of header nodes and then each row. A helper method reduced redundancy, but no points for repeated code on exam. No need to treat a single row matrix as a special case.

```
private boolean isRectangular() {
    if (firstRow == null) {
        return true;
    }
    int targetColumns = numColumns(firstRow);
    RowHeader<E> temp = firstRow.nextRow;
    while (temp != null) {
        int columns = numColumns(temp);
        if (columns != targetColumns) {
            return false;
        }
        temp = temp.nextRow;
    }
    return true;
}

private int numColumns(RowHeader header) {
    int count = 0;
    DataNode<E> temp = header.first;
    while (temp != null) {
        count++;
        temp = temp.next;
    }
    return count;
}
```

18 points, Criteria:

- empty case, firstRow == null, 2 points
- determine target number of columns (typically first row), 1 point
- when determining number of columns in a row:
    - move to first data node, 1 point
    - correctly check all nodes in row, Lose if OBOE 2 points
    - counter correct, 1 point
    - "move" temp correctly, 3 points
- check later rows have same number of columns as first:
    - move to next rowHeader node: 1 point
    - correctly check all row header nodes, Lose if OBOE 2 points
    - if mismatch, return false ASAP, 2 points
    - "move" temp for header to next row header, 2 points
- return true correctly, 1 point

Other deductions:

- doesn't access first correctly. It is an instance variable in RowHeader nodes, -2
- adding inappropriate public methods that violate encapsulation, -3
- Worse than O(N^2). O(N^3) or worse. -4 create new data structures, -4

- NPE not covered by other criteria, -3
- alter structure in any way, -5

3. Comments: Similar to example in class, creating and using a frequency map.

```
public static int getNum(Map<String, Set<String>> map, int people) {
    HashMap<String, Integer> freqs = new HashMap<>();
    // build the frequency map
    for (String name : map.keySet()) {
        for (String activity : map.get(name)) {
            if (freqs.containsKey(activity)) {
                freqs.put(activity, freqs.get(activity) + 1);
            } else {
                freqs.put(activity, 1);
            }
        }
    }
    // Check the number of activities that the target number of people
    // or more enjoy.
    int result = 0;
    for (String activity : freqs.keySet()) {
        if (freqs.get(activity) >= people) {
            result++;
        }
    }
    return result;
}
```

18 points, Criteria:

- Create local map, 1 point
- local map is a HashMap, 2 points (efficiency)
- loop through keys of provided map, 1 point
- get value for key correctly, 2 points
- loop through set, 1 point
- determine if key present or not, 1 point
- if activity **not** present, add key and freq of 1 to local map, 2 points
- if activity is present, get, increment, and put new frequency correctly (lose if ++ on Integer from get), 2 points
- counter for result, 1 point
- loop through local map's keys, 1 point
- correctly get and check if frequency of activity meets criteria, 2 points
- increment counter, 1 point
- return result, 1 point

Other:

- worse than O(keys * activities), -4 - new Iterator(): -2 - alter map: -5
- Comodification error: -4 - other inefficiencies: - 3

4. Comments: In actuality this is a ***depth first search in an undirected graph.*** Follows the recursive backtracking template closely. No need to undo / remove elements from Set.

Many solutions called the helper method but ignored / did nothing with the return value.
Lots of potential infinite recursion / stack overflow errors when adding the current airline to the set after the recursive calls. Bounces back and forth A -> B -> A -> B -> A -> B ….
A better name for the set would have been visited instead of tried.

```
private static boolean helper(Airline org, Airline dest,
                              Set<Airline> visited) {
    if (org.equals(dest))
        return true; // found a path
    else if (visited.contains(org)) {
        // been here before, going roundy, roundy
        return false;
    }
    // Now we are here at the org Airline.
    visited.add(org);
    Airline[] partners = org.getPartners();
    for (int i = 0; i < partners.length; i++) {
        if (helper(partners[i], dest, visited)) {
            return true;
        }
    }
    return false;
}
```

14 points, Criteria:

- check success base case and return's true, 3 points
- failure base case, already tried, return false, 2 points
- recursive case:
    - add current (org) to set of tried, 2 points (Can be in loop, but must be before the recursive call)
    - get partners and loop through, 1 point
    - recursive call correct (new org), 2 points (if don't use return value, lose)
    - if recursive call returns true, return true, 2 points
    - OKAY If remove after recursive call or after for loop, 0
- return false if never found path, 2 points

Other:

- early return, -5
- infinite recursion, -3 (Can be on top of -2 for not adding to set of tried / visited.)
- Nested loop for org and destination partners, -4
- Creating new data structures, -4