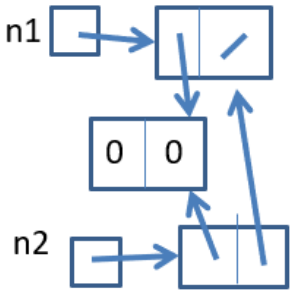
CS314 Spring 2023 Exam 2 Solution and Grading Criteria.

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 or GCE - 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.
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. 6
B. O(logN)
C. 137+468
D. 22
E. 4 seconds
F. 20 seconds
G. O(N^2)
H. O(N)
I. 10 seconds
J. 18 seconds
K. Unknown until code is run. OWTTE, (HashMaps store their keys in a seemingly random order.)
L. 11 seconds (code would be O(NlogN))
M. null



N. We were VERY strict on this one.
O. Radix sort
P. selection sort or quick sort (sort is unstable), 1 point each, -1 for any besides.
Q. -22
R. 2 seconds
S. 3 5 9 17
T. 9 8 7 6
U. 8
V. 4
W. 3 -7 7 9 5
X. 4 seconds
Y. 21 seconds

Extra Credit: 4 times (2 points)

2. Comments: A very easy problem as it was the second section problem the day before. Most students did well, especially if they attended section. Biggest issue was not making the temporary Node reference refer to the next node in the structure. **There is a decent chance (+50%) there will be another linked list question on exam 3.**

```
public int lastIndexOf(E tgt) {
    int index = 0;
    int result = -1;
    Node<E> temp = first;
    while (temp != null) {
        if (temp.data.equals(tgt)) {
            result = index;
        }
        index++;
        temp = temp.next;
    }
    // never found an element equal to tgt
    return result;
}
```

13 points, Criteria:

- track current index with variables initialized to 0 and incremented in loop, 1 point
- temp node that starts with first, 1 point
- loop that checks every node in the structure, 3 points (-2 if OBOE, misses one node, -3 if return early from loop)
- in loop, check if current node data equals tgt, 2 points (lose if don't call equals)
- if matches, record the current index for later use, 1 point
- make temp node refer to next node in structure, 4 points
- return result, 1 point (lose if don't return -1 if not present OR index incorrect)

Other deductions:

- Worse than O(N), -3
- disallowed methods, varied based on severity
- alter list, -5

3. Comments: A map question. Quite a lot to do. Required a triply nested loop. It was acceptable to have the nested outer loop that repeated work, comparing book A to book B and then later comparing book B to book A. Solutions did not have to avoid this. We did see a fair number of clever solutions that tried (and some worked) to avoid this inefficiency. Lots of issues in calculating the similarity score correctly. Many solutions did not simply use the contains method on the other set to see if a character was present or not.

```
public static double highestSimilarity(Map<String, Set<String>> books) {
    double max = 0;
    for (String book1 : books.keySet()) {
        for (String book2 : books.keySet()) {
            if (book1 != book2) {
                int sharedCharacters = 0;
                Set<String> myChars = books.get(book1);
                Set<String> otherChars = books.get(book2);
                for (String aCharacter : myChars) {
                    if (otherChars.contains(aCharacter)) {
                        sharedCharacters++;
                    }
                }
                double simScore = 1.0 * sharedCharacters
                / (myChars.size() + otherChars.size() - sharedCharacters);
                if (simScore > max) {
                    max = simScore;
                }
            }
        }
    }
    return max;
}
```

20 points, Criteria:

- local variable for max, 1 point
- nested loop using iterator for keyset to check all pairs of keys, 4 points (partial credit possible)
- check to make sure we do not compare a book to itself, (equals okay), 2 points
- get the sets of characters for both books, 3 points (must call get method correctly)
- inner most loop that iterates through one of the sets. (Either is acceptable, although a slight improvement could be made by looping through the smaller. This was not requred), 2 points
- call contains on other set of characters, 3 points (lose if nested loop here)
- increment shared characters correctly, 1 point
- determine similarity score correctly, 2 points (-1 if int division) (Lose if don't use size() method. For example, a loop to count characters.)
- check for new max and assign correctly, 1 point
- return answer, 1 point

Other:

- alter map or sets, -5
- new data structures, -3
- Math.max, -1
- double call on methods, efficiency, -2

4. Comments: A combination of the fair teams and flow off map problems from assignment 6. I thought this would make the question easy, but to be honest, many, many students struggled with this question. Like fair teams a local variable to track the best answer is needed and we need to try all 4 direction if they are possible. We are not simply looking for **A** solution. We are looking for the **BEST** solution and thus must try all directions that we can go. T0 avoid an infinite loop, bouncing back and forth between two cells that have a difference in of 10 or less, we need to alter the current cell. Easiest approach was to set it to -11 (all elevations were >= 0) or 1,000,010 (all elevations are < 1,000,000). That way we won't ever go back to a cell we have already visited. Changing a cell by +11 could lead to a logic error where we create a path that did not exist before. Also, a lot of folks missed the base case. Same as flow off map, we are on the edge of the matrix, not off the edge. And if we are on the edge (the current row and / or column is an edge cell) it takes ZERO steps to reach the edge, not 1. Also saw a lot of nest loops to go through the row and column deltas. Just a single loop. Didn't need to check in bounds as the edge cells are the base case and the initial row and column were guaranteed to ve in bounds. Just like the assignment 6 flow off the map problem.

```
public static int minSteps(int[][] area, int row, int col) {
    if (row == 0 || col == 0 || row == area.length - 1
            || col == area[0].length - 1) {
        return 0; // We are here!
    }
    int min = Integer.MAX_VALUE / 2;
    int thisElevation = area[row][col];
    area[row][col] = -100; // So we don't come back here.
    for (int i = 0; i < rc_deltas[0].length; i++) {
        int newR = row + rc_deltas[0][i];
        int newC = col + rc_deltas[1][i];
        int diff = area[newR][newC] - thisElevation;
        if (-10 <= diff && diff <= 10) {
            int steps = 1 + minSteps(area, newR, newC);
            if (steps < min) {
                min = steps;
            }
        }
    }
    area[row][col] = thisElevation; // undo so we can try other ways
    return min;
}
```

17 points, Criteria:

- base case for success first, 1 point
- base case check correct, 1 point
- return 0 steps if base met, 1 point
- recursive case, local variable for minimum number of steps, 1 point
- recursive case, alter current elevation so that we do not end up in an infinite loop, 3 points
- loop through choices of up, down, left, right, 1 point (lenient on 2d array)
- determine new row and new column, 1 point (lenient on 2d array)
- check if we can move the cell or not, 1 point (Math.abs allowed)
- if we can move to new cell, make recursive call correctly, 2 points
- calculate number of steps correctly, 1 point
- check if new min, 1 point
- undo change that prevented infinite loops, 2 points
- return best answer, 1 point

Other:

- add helper or alter parameters (likely -5)
- early return, not trying all options -4 additional
- Added helper, -4
- No compare of answers, -3

$O(N^2)$ at base case, -3
No difference check, -2
New data structure, -4