CS 314
Discussion
Problems

- Main Problem
  - GenericList
- Extra Problem
  - Extra Data Structure

Exams
fr this time though
Exam Stuff

● Exam Structure
  ○ 25 Short Answer
  ○ 3 Coding Questions

● No questions during exams. Write down your assumptions if confused
  ○ Doesn’t mean we’ll grade based on your assumption

● Keep note of time. A fair # of students don’t attempt some coding questions
public int trimEqualBacks(GenericList<E> other) {
    int indexThis = size - 1;
    int indexOther = other.size - 1;
    int result = 0;
    while (indexThis >= 0 && indexOther >= 0 &&
           container[indexThis].equals(other.container[indexOther])) {
        result++;
        container[indexThis] = null;
        indexThis--;
        other.container[indexOther] = null;
        indexOther--;
    }
    size -= result;
    other.size -= result;
    return result;
}
public int trimEqualBacks(GenericList<E> other) {
    int oldSize = size;
    while (size > 0 && other.size > 0 &&
        container[size - 1].equals(other.container[other.size - 1])) {
        container[indexThis] = null;
        other.container[indexOther] = null;
        size--;
        other.size--;
    }
    return oldSize - size;
}
public boolean removeSingleOccurrence(Object tgt) {
    for (int i = 0; i < numberOfElements; i++) {
        if (tgt.equals(elements[i])) {
            numberOfElements--;
            elements[i] = elements[numberOfElements];
            elements[numberOfElements] = null;
            return true;
        }
    }
    return false;
}
1. Two Sum

Given an array of integers `nums` and an integer `target`, return indices of the two numbers such that they add up to `target`.

You may assume that each input would have **exactly one solution**, and you may not use the same element twice.

You can return the answer in any order.

**Example 1:**

Output: `[0,1]`  
Explanation: Because `nums[0] + nums[1] == 9`, we return `[0, 1]`.

**Example 2:**

Input: `nums = [3,2,4]`, `target = 6`  
Output: `[1,2]`
<table>
<thead>
<tr>
<th>Category</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arrays &amp; Hashing</td>
<td>(0 / 10)</td>
</tr>
<tr>
<td>Two Pointers</td>
<td>(0 / 5)</td>
</tr>
<tr>
<td>Sliding Window</td>
<td>(0 / 6)</td>
</tr>
<tr>
<td>Stack</td>
<td>(0 / 7)</td>
</tr>
<tr>
<td>Binary Search</td>
<td>(0 / 7)</td>
</tr>
<tr>
<td>Linked List</td>
<td>(0 / 11)</td>
</tr>
<tr>
<td>Trees</td>
<td>(0 / 15)</td>
</tr>
<tr>
<td>Tries</td>
<td>(0 / 3)</td>
</tr>
<tr>
<td>Heap / Priority Queue</td>
<td>(0 / 7)</td>
</tr>
<tr>
<td>Backtracking</td>
<td>(0 / 9)</td>
</tr>
<tr>
<td>Graphs</td>
<td>(0 / 13)</td>
</tr>
<tr>
<td>Advanced Graphs</td>
<td>(0 / 6)</td>
</tr>
<tr>
<td>1-D Dynamic Programming</td>
<td>(0 / 12)</td>
</tr>
<tr>
<td>2-D Dynamic Programming</td>
<td>(0 / 11)</td>
</tr>
<tr>
<td>Greedy</td>
<td>(0 / 8)</td>
</tr>
<tr>
<td>Intervals</td>
<td>(0 / 6)</td>
</tr>
<tr>
<td>Math &amp; Geometry</td>
<td>(0 / 8)</td>
</tr>
</tbody>
</table>
QueensAreSafe

- Relevant Directions
- Parameterized Row/Col Solutions
- Slope Method
Directions

- Only need to check 4 directions
Directions

- You can use an array to store the different changes in rows and columns

rows = {0, 1, 1, 1}
cols = {1, 1, 0, -1}