We've learned about:

- **Turtle:**
  - functions, with parameters, returning values
  - recursion, single/multiple recursive calls, stack frames

- **Strings/Files:** operations

- **Lists:** operations

- **Loops:** for, while

- **Searching/sorting** →
  - search: linear, binary (list must be sorted)
  - sort: insertSort, bubbleSort

- **Trees**<sub> traversal</sub>

- **Graphs** - adj list, DFS

- **Time complexity**
Possible review problems

- (Strings/Searching) Write a function, uses_all, that takes a word and a string of letters and returns True if that string uses all the required letters. Write pseudocode, test cases and implement a solution.

- (Lists/Searching/Sorting) Write a boolean function, is_sorted, that takes a list of elements and returns whether they are in ascending order or not. Write pseudocode, test cases and implement a solution.

- (Trees/Recursion) Write a function print_tree, that displays a graphical representation of an indexed binary search tree (i.e. Homework 8).
The following `tree_size` pseudocode incorrectly tries to determine the number of nodes in a binary search tree. First, trace the code with the following indexed tree representation and show what it displays for output.

```python
function tree_size(tree, node):
    if node is not a valid index:
        return 0
    left_size <- call tree_size(tree, tree[node][1])
    right_size <- call tree_size(tree, tree[node][2])
    return 1 + left_size + right_size

tree <- [ (10,1,2), (5,-1,3), (15,-1,-1), (7,-1,-1) ]
print the result of calling tree_size(tree, 0)
```

Next, fix the pseudocode and implement the solution. Verify it works with several of your own test cases.
function tree_size(tree, node):
    if node is not a valid index:
        return -1
    left_size <- call tree_size(tree, tree[node][1])
    right_size <- call tree_size(tree, tree[node][1])
    return 1 + left_size + right_size

tree <- [ (10,1,2), (5,-1,3), (15,-1,-1), (7,-1,-1) ]
print the result of calling tree_size(tree, 0)
(Graphs) You are given the following input file from the Cowabomba lab. Show the resulting node list and adjacency list. Draw the resulting graph and trace which bomb paints the most cows.

- bomb ORANGE 5 0 1
- cow Daisy 6 0
- bomb YELLOW 0 5 1
- cow Fauntleroy 0 6
- bomb RED 0 0 5
- bomb GREEN -5 0 1
- cow Milka -6 0
- bomb BLUE 0 -5 1
- cow Hector 0 -6

(Graphs/Recursion) Compute the number of bombs "reachable" from a given bomb.
- (Graphs/Recursion) Compute the number of bombs “reachable” from a given bomb.

**Pseudo Code:**

```python
def num_bombs(adjlist, inode, exploded):
    sum = 1
    for elem in adjlist[inode]:
        if elem is a bomb:
            if elem has not been exploded:
                set elem's position in exploded as True
                x = num_bombs(adjlist, elem, exploded)
                add x to sum
    return sum
```

Alternatively:

use DFS from last week and then go through the visited list and count the number of bombs that have been visited.
For each of the algorithms below, state its (worst-case) time complexity:

- linear search: $O(n)$
- binary search: $O(\log n)$
- bubble sort: $O(n^2)$
- insert sort: $O(n^2)$
- tree traversal: $O(n)$ (in the worst case, we make $n$ recursive calls and about 5 steps per recursive call (except the recursive call steps) → about $5n$ steps: $O(n)$)
- DFS

For each of the following time complexities, sketch a typical pseudo code with that time complexity:

- $O(1)$
- $O(n)$
- $O(n^2)$
<table>
<thead>
<tr>
<th>Lab Number</th>
<th>Lab Name</th>
<th>Rating</th>
<th>Difficulty</th>
<th>Least Favorite</th>
<th>Most Favorite</th>
<th>Most Helpful</th>
<th>Least Helpful</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Hello world</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Staircase</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Vanishing squares</td>
<td>2</td>
<td>9</td>
<td>9</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Tiny Turtle</td>
<td>23</td>
<td>8</td>
<td>5</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Class schedule</td>
<td>12</td>
<td></td>
<td>1</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>SI session position</td>
<td>4</td>
<td>8</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Decision trees</td>
<td>1</td>
<td>5</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Cowabomba</td>
<td>2</td>
<td>10</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>