1. [15 points] Start with an empty Binary Search Tree.
   a. Draw a picture of the tree that will result from adding the following values to
   your tree, in the order they are given: 20 30 10 24 18 4 12

   ![Binary Search Tree Diagram]

   b. How will your answer change if the BST is replaced by an AVL tree?

   It won’t change at all; the tree in (1a) is balanced.
2. [5 points] What is the Balance Condition for AVL trees?

At every node the left and right subtrees have heights that differ by no more than 1.

3. [15 points] Now add data 19 and 11 (in that order) to your AVL tree from (1 b).
   Draw the resulting AVL tree.

   ![AVL tree diagram]

4. [15 points] Here is an array of 8 integer values:

   2 9 4 1 7 6 8 3

   We can think of this as a tree. Give the array that will result from turning this array into a heap. You may find it easier to think in terms of trees, but put your final answer back into an array.

   ![Heap tree diagram]

   1 2 4 3 7 6 8 9
5. [20 points] I want to have a map where the keys are names of movies and the values are lists of the actors in the movie. I could implement this as a hashmap with separate haining (linked lists) or a treemap with an AVL tree. I will put n movies into this structure.

a) What are Big-Oh estimates of the worst-case and average case insertion time for a new movie for both the hashmap and treemap versions?

<table>
<thead>
<tr>
<th></th>
<th>Average Case</th>
<th>Worst Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hashmap</td>
<td>O( loadfactor)</td>
<td>O(n)</td>
</tr>
<tr>
<td>Treemap</td>
<td>O( log(n) )</td>
<td>O( log(n) )</td>
</tr>
</tbody>
</table>

b) Estimate the worst-case and average-case lookup times for a movie that is in the map, for both the hashmap and treemap.

<table>
<thead>
<tr>
<th></th>
<th>Average Case</th>
<th>Worst Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hashmap</td>
<td>O(loadfactor)</td>
<td>O(n)</td>
</tr>
<tr>
<td>Treemap</td>
<td>O( log(n) )</td>
<td>O( log(n) )</td>
</tr>
</tbody>
</table>

c) Suppose I want to output the movies in alphabetical order; which structure makes that easier?

An inorder traversal of an AVL tree (or any other BST) gives a complete sort of the data, so the treemap makes this easy. The data in a Hashmap is not sorted.

d) Suppose I want to output the actors in alphabetical order; which structure makes that easier?

Neither structure gives much help for this. You need to iterate through all of the keys, gather the actors, and sort them.
6. [15 points] Suppose we have a directed graph with positive weights and want to find the LONGEST path through the graph from a given source to each other node.
   a) When will there not be a longest path?

      There is no longest path when there is a cycle (since we are told the weights are positive, any cycle will have positive weights).

   b) Give an algorithm to find the weight of the longest (i.e., most weight) path from the source to each other node. (I’m not looking for pseudo-code; just an English description of the algorithm and whatever data structures are needed to implement it.)

      The most efficient way to do this is to use our algorithm for acyclic graphs – maintain a working set (using a queue, a list or a stack) starting with any nodes with no incoming edges. At each step remove some element X of the working set, mark it as done, and “remove” all of its outgoing edges (by reducing a counter) For each of the outgoing edges e, compute the cost of a path to the endpoint Y of e as X.weight+e.weight. If that is greater than Y.weight, use it to update Y.weight. If Y has no more incoming edges, add it to the working set. Continue this until the queue empties. If it never empties, the graph was not acyclic.

      If you want to do this without a topological sort, you can’t use Prim’s algorithm. You should use a queue (not a priority queue) and a variant of the Bellman-Ford algorithm that keeps adding enodes back to the queue until no longer path can be found.
7. [15 points] Here is a game. Start by building a structure, which can take a while. After you build the structure I will give you pairs of words w1 and w2. Your job is to find the shortest sequence of changes that turns w1 into w2, changing only one letter at a time, and always using actual English words. For example, we can turn cat into dog with 4 changes: “cat” “bat” “bag”, “bog” “dog”. Here is how bob becomes a spy: “bob” “bomb” “comb” “come” “comet” “covet” “covert” You should count the following as legal changes:
   a. Replacing any one instance of one letter with a different letter.
   b. Removing any one letter
   c. Adding any one letter

Give an algorithm for playing this game.

Make a graph where the nodes are words. Edges connect two words that differ by one letter. Given any two words, it is easy to tell if they should be connected. There will be an edge joining them if either

- They are the same length and a loop through their letters shows that they differ in exactly one position, or
- Their lengths differ by 1, and we can turn the longer one into the shorter one by deleting one letter.

Then, given words w1 and w2, compute the shortest paths from w1. This will be relatively quick since the edges are unweighted. The path from w1 to w21 has the sequence of words we seek.