1. Describe an efficient way to implement the select method for BSTs. $T.select(k)$ should return the $k^{th}$ smallest value stored in tree $T$: for example, $T.select(1)$ will return the smallest element, $T.select(n)$ the largest, $T.select(n/2)$ the median, and so on. Suppose that $T$ has $n$ nodes and height $h$. One approach would be to perform an inorder traversal and capture the $k^{th}$ element that is visited during the traversal. This would take time $O(n)$. You should aim for $O(h)$. Augment the tree by adding to each node a field which stores the size of the subtree rooted at that node, and use that information during the select process. [8 points]

2. Insert into an initially empty AVL tree, in the order given, the following values: 15, 12, 5, 4, 2, 9, 10, 11, 7, 6. Show the intermediate steps. [6 points]

3. Insert the values above into an initially empty (2,4)-tree. [6 points]

4. Let $T$ and $U$ be two (2,4)-trees storing $n$ and $m$ items, respectively, such that any item in $T$ has a key less than the keys of all items in $U$. Describe an $O(\lg n + \lg m)$ method for joining the trees into a single tree that stores all the items in $T$ and $U$. The original $T$ and $U$ may be be destroyed in the process. [8 points]

5. Suppose that an AVL implementation stores the height of the subtree in each node to calculate the balance factors. Also suppose that 3 bits are allocated for this purpose. What is the fewest number of nodes in an AVL tree that could cause one of these fields to overflow? What is the largest number of nodes that could be accommodated without overflow? (Use the $g_k$, related to the Fibonacci numbers, from class.) [6 points]

Total: 34 points

Notes:

- We had defined $g_k$ as the minimum number of nodes in an AVL tree of height $k$. We then saw that $g_k = F_{k+3} - 1$, where $F_k$ is the $k^{th}$ Fibonacci number.