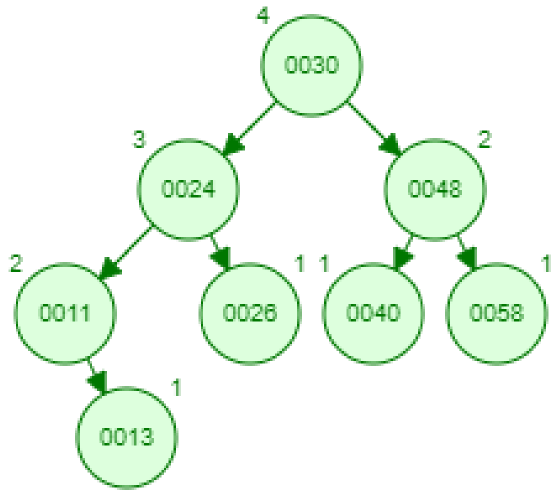
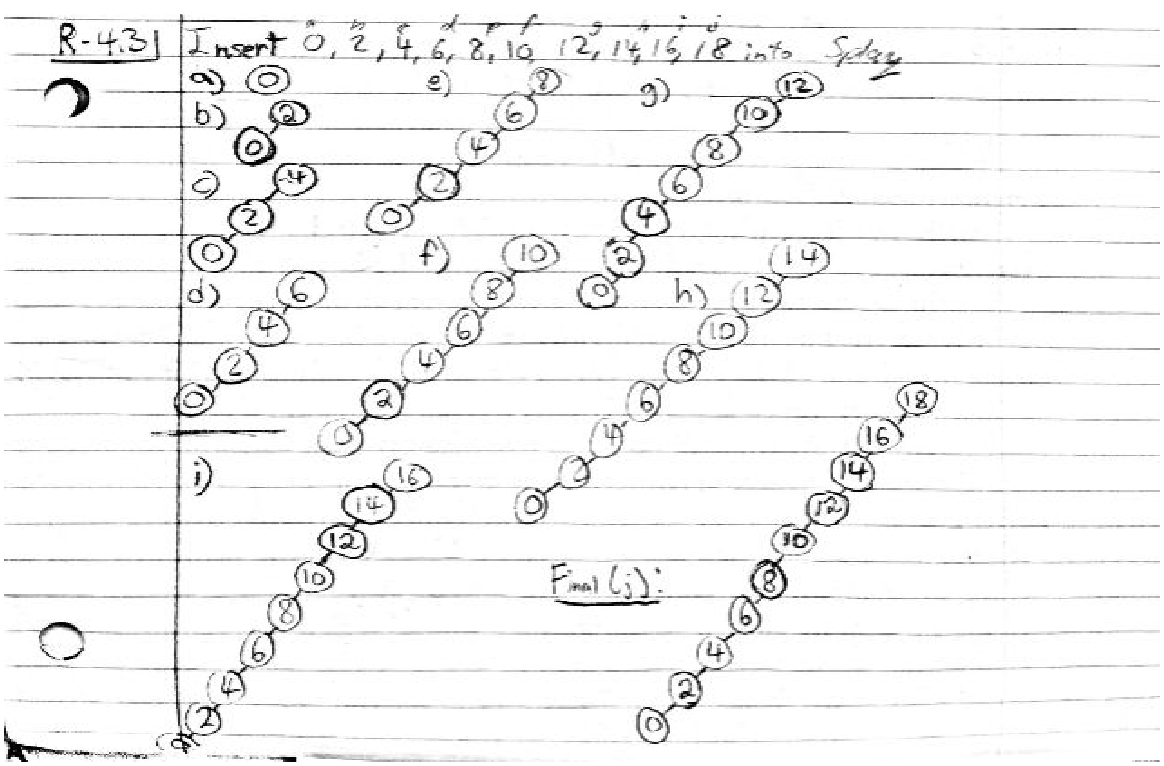
R-4.1 Consider the insertion of items with the following keys (in the given order) into an initially empty AVL tree: 30, 40, 24, 58, 48, 26, 11, 13. Draw the final tree that results.



R-4.3 Consider the insertion of items with the following keys (in the given order) into an initially empty splay tree: 0, 2, 4, 6, 8, 10, 12, 14, 16, 18. Draw the final tree that results.



C-4.1 Show that any n-node binary tree can be converted to any other n-node binary tree using O(n) rotations.
Hint: Show that O(n) rotations suffice to convert any binary tree into a left chain, where each internal node has an external right child.

The idea is to construct an algorithm that converts any n-nodes binary tree into a left chain in time $O(n)$ and then use the inverse of this algorithm to convert the obtained left chain into the desired binary tree.

- Algrithm that coverts any n-nodes binary tree into a left chain in $O(n)$
  1. perform left relations on the root node until its right child is empty
  2. If the left child is now empty we are done. If the left child is not empty we traverse down to the left and go back to 1 to perform rotations on this subtree.

Time: There are n nodes in the binary tree. A rotation takes time $O(1)$. Each time we do a rotation the height of the left chain that we are building grows with at least one. Therefore the algorithm will terminate after at most n rotations giving us time $O(n)$.

C-4.3 Show, by induction, that the minimum number, n_h, of internal nodes in an AVL tree of height h, as defined in the proof of Theorem 4.1, satisfies the following identity, for h ≥ 1:

$$n_h = F_{h+2} - 1,$$

where F_k denotes the Fibonacci number of order k, as defined in the previous exercise.

C-4.3 (cont'd)

$n_1 = 1 \qquad F_{1+2} - 1 = F_3 - 1 = 2 - 1 = 1 \qquad n_2 = 2 \qquad F_{2+2} = 3$

$1 = 1$ ✓ $\qquad 2 = 3 - 1 = 2$ ✓

Now assume $n_h = F_{h+2} - 1$ for $h = k$ and $h = k-1$

WTS $n_{k+1} = F_{k+3} - 1$

$n_{k-1} = F_{k+1} - 1 = F_{k-1} + F_k - 1$

Inductive hypothesis: $n_k = F_{k+2} - 1 = F_k + F_{k+1} - 1$

From Thm 4.1 proof: $n_h = 1 + n_{h-1} + n_{h-2}$

So $1 + n_{k-1} + n_{k-2} = F_k + F_{k+1} - 1$

$n_{k+1} = 1 + n_k + n_{k-1} = 1 + (F_k + F_{k+1} - 1) + (F_{k-1} + F_k - 1)$

$= F_{k+2} + F_{k+1} - 1$

$= F_{k+3} - 1$ ✓

So $\forall n$, $n_h = F_{n+2} - 1$

C-4.7 Draw an example of an AVL tree such that a single remove operation could require Θ(log n) trinode restructurings (or rotations) from a leaf to the root in order to restore the height-balance property. (Use triangles to represent subtrees that are not affected by this operation.)

let T_h be an arbitrary binary tree w/ n_h be minimum nodes of height h.

∴ T_0 : o

T_1 : (Zero Rotation $\log_2 1 = 0$)

T_2 deletion ⇔ T_1 T_0 → $= A_1$ (one Rotation $\log_2 2 = 1$)

T_3: T_2 T_1 ⇒ T_2 A_0 ⇒ A_1 (One Rotation $\log_2 3 = 1.9 = 1$)

T_4: T_3 T_2 ⇒ T_3 A_1 ⇒ A_2 (Two series of rotation $\log_2 4 = 2$)

⋮

T_h : T_{h-1} T_{h-2} ($\log_2 n$ rotation)

Here, h is bounded by logn.

∴ This is an informal proof to prove that for AVL tree remove() operation, it requires $\Theta(\log n)$

A-4.2 Suppose you are working for a fast-growing startup company, which we will call "FastCo," and it is your job to write a software package that can maintain the set, E, of all the employees working for FastCo. In particular, your software has to maintain, for each employee, x in E, the vital information about x, such as his or her name and address, as well as the number of shares of stock in FastCo that the CEO has promised to x. When an employee first joins FastCo they start out with 0 shares of stock. Every Friday afternoon, the CEO hosts a party for all the FastCo employees and kicks things off by promising every employee that they are getting y more shares of stock, where the value of y tends to be different every Friday. Describe how to implement this software so that it can simultaneously achieve the following goals:

- **The time to insert or remove an employee in E should be O(log n), where n is the number of employees in E.**
- **Your system must be able to list all the employees in E in alphabetical order in O(n) time, showing, for each x in E, the number of shares of FastCo the CEO has promised to x.**
- **Your software must be able to process each Friday promise from the CEO in O(1) time, to reflect the fact that everyone working for FastCo on that day is being promised y more shares of stock. (Your system needs to be this fast so that processing this update doesn't make you miss too much of the party.)**

A-4.2 : An AVL-tree type structure would be useful here. Let's get to Some detail with that:

(Each node Contains all required information)

- Insertion & Deletion: is an o(log n) runtime Operation, which is nice.
- Alphabetical Order: Our tree sorts alphabetically, with earlier letters as left children and later ones as right children. In-order traversal is o(n).
- Friday Bonus: Store a value "Promised" globally, and have each node point to it. Updating this value each Friday is very simple, and is o(1). It's not different for everyone.

An AVL tree was chosen for this job since accessing in o(log n) time every time was a requirement. It makes printing in-order Names fairly Straight-forward, and makes sorting straight-forward. Overall, a fairly easy pick.