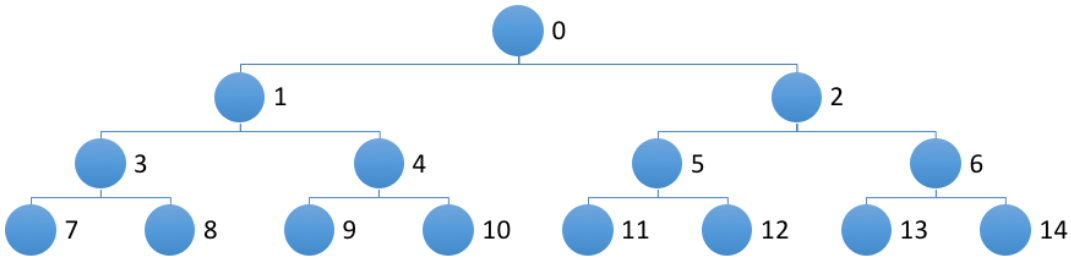
Tree Balancing, Reading

In all of our studies in this module we make one *hugely important* oversight! In all of our big oh considerations we assumed that our tress would be flat, like this:



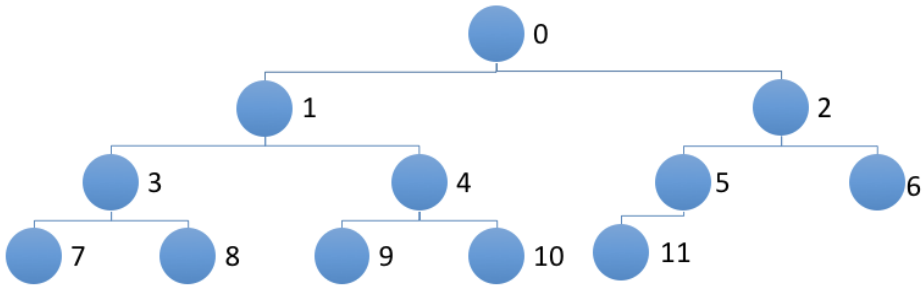
In our heap application, we did indeed maintain this flatness by not having another level until the level above was full.

Full Trees

Ideally, our trees would look like the above, with no holes in any level, and the last level full. That configuration is called a "full tree". It's idealized and rare -- rare because the number of values in the tree has to be exactly 1, 3, 7, 15, 31, etc. Anything else requires at least an incomplete bottom level.

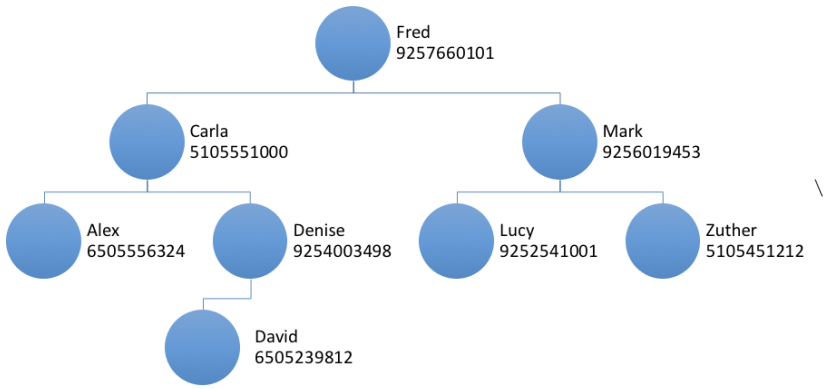
Complete Trees

Realizing that full trees are just not going to happen very often, the next best thing we can do is keep all the holes in the bottom level, and let the rest of the levels be "full". If the bottom level's nodes are all fully to the left, and all the unused locations to the right of them, that's a "complete tree". Heaps use complete trees.



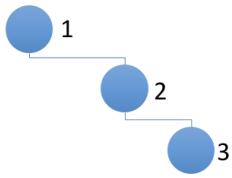
Balanced Trees

It's hard to have complete binary *search* trees, because of the requirements that the left child be <= the parent, and the right >= the parent. We'd be constantly rearranging the tree to left-justify the bottom level. So the best we can do is a "balanced" tree, with all levels above the lowest full, and holes in the bottom where they need to be:



Uncontrolled Growth

Unfortunately, we do not enforce any requirement for keeping our trees balanced in our push and pop operations. We allow uncontrolled growth, which leads to holes all over the place, in levels other than the bottom. Consider this example: in a BST of int values, push a 1, a 2, and a 3. Already we've lost balance:



Rebalancing

The solution to the above imbalance after the addition of the 3 would be to raise the 2 to the top level, and make 1 *its* child. That process is called a "rotation" because that's what it looks like when animated. But it's really like a set of elevators that raise and lower and relink nodes to maintain full levels above the bottom level. Remember that the left-to-right order of nodes still has to be maintained.

There are several different methods that have been developed in computer science for maintaining balance. We'll not be getting into the algorithms or code for them in this course, but here are some references that explain how they are done:

- **AVL Trees** (https://en.wikipedia.org/wiki/AVL_tree)
  - **A USF animation** (https://www.cs.usfca.edu/~galles/visualization/AVLtree.html)
- **Red-Black Trees** (https://en.wikipedia.org/wiki/Red%E2%80%93black_tree)
  - **A USF animation** (https://www.cs.usfca.edu/~galles/visualization/RedBlack.html)
- **Splay Trees** (https://en.wikipedia.org/wiki/Splay_tree)
  - **A USF animation** (https://www.cs.usfca.edu/~galles/visualization/SplayTree.html)
- **2-3 Trees** (https://en.wikipedia.org/wiki/2%E2%80%933_tree)