Heirachical Structures, Reading

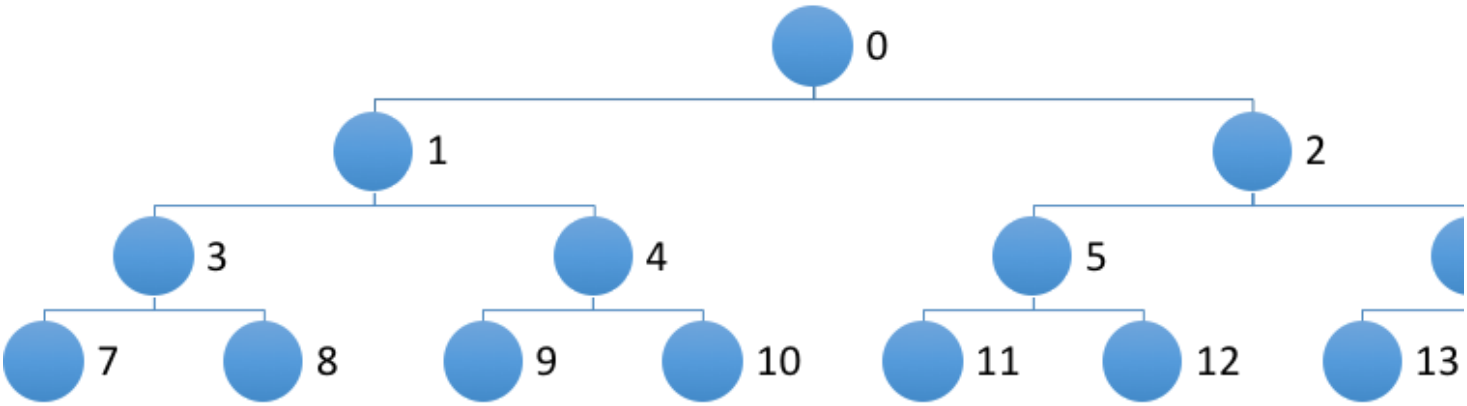
Conceptually this is an new structure -- it's not sequential like an array or linked list. At least we don't *traverse* it sequentially. If we use an array for the values, we'll need something other than **i++** to move through it. There would have to be a different formula for finding the value above and below. A value needs to be able to locate the ones *below* it so that it can compare them and promote the highest-qualified to take its place when *it* gets promoted. And a value needs to be able to locate the one *above* it for possible promotion in a push operation.

Let's look at our options.

Arrayed Solution

It can get really complicated to track where is a value's "parent" and "children" (to move away from military terms in favor of generic terms). The problem is that we don't know how many children any given node has. But we can fix that problem by allowing only *two* children. That loses our military example, but generically, all we really need in the structure is that each child's value be less than or equal to its parent's value. If that's maintained, then one of the two children of any value will be guaranteed to be greater than (or equal to) all values below it in the hierarchy.

That means the top level of the hierarchy will (of course) have one value. The next level, two. Then four, eight, sixteen, and so on, like this:



Putting this into an array with the indexes matching the labels above, we get this:

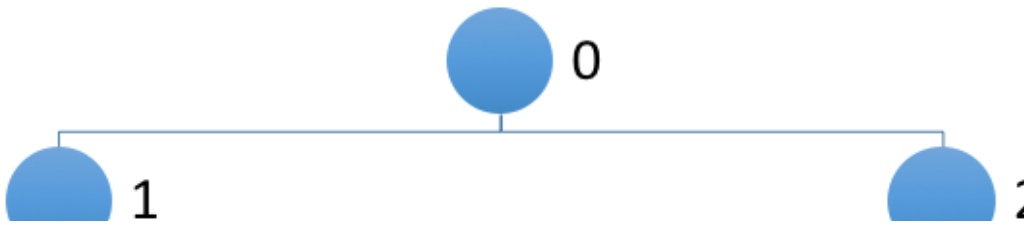
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

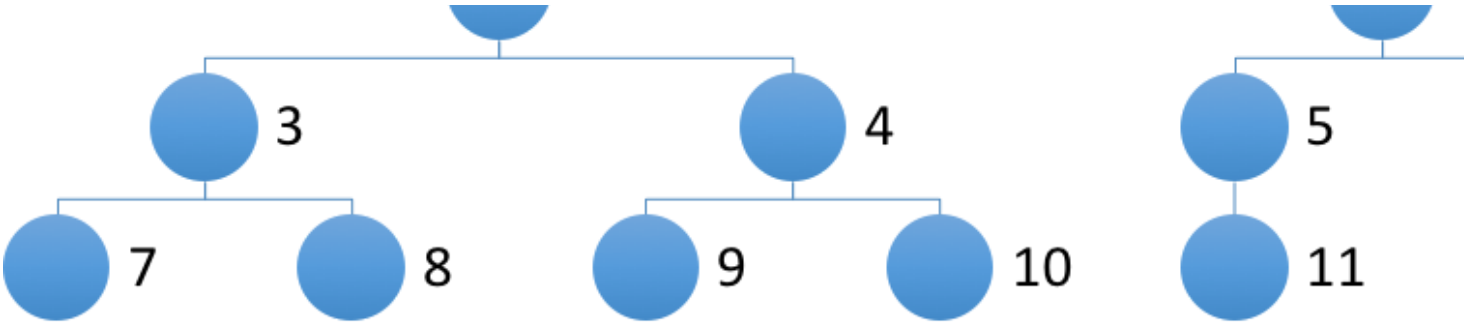
Per the above, using the indexes as values, the children of 0 are 1 and 2. 1's are 3 and 4. 2's are 5 and 6. 3's are 7 and 8, etc. Do you see formulae emerging? How about:

```
left-child index = 2 * index + 1
right-child index = 2 * index + 2
parent index = (index + 1) / 2 - 1 // using integer division with truncation
```

So an arrayed option could work. But what if we're not always so lucky as to have all positions on the bottom level filled? What if instead of 15 values (indexes 0 - 14), we have only 12? That could leave "holes" in the array and complicate things with an "inUse" flag and searching for the first unused location for a push.

But if we keep the array *organized*, we can overcome this issue. Allow for there only to be holes in the *bottom* row, and require that positions be filled *from the left to the right*. When full, add another row. Something like this, for n=12:





Now, no in-use tracking is necessary. When a value with no children is found, it's detectable because its left-child index will be greater-than or equal-to the "siz". (When there's just *one* child, it's a *left*-child).

Where to push new values? The obvious place now is the array index position *equal to siz* (before incrementing it).

Linked Solution

You my have been expecting this to be the solution, and not an array. Who knew the array would work out so well? Again, a value needs to know its parent and its children. We could design a node like this:

```
struct Node
{
  V value;
  Node* parent;
  Node* leftChild;
  Node* rightChild;
};
```

...and proceed from there. But there will be some challenges -- one is tracking the Node that would be the parent of any new push. It's certainly doable, but just thinking about the logic and all the coding is enough to say: let's go arrayed!