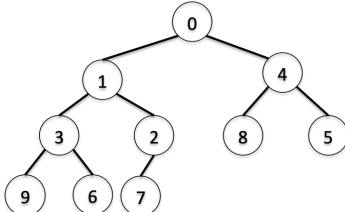


## Lab 8

- Download `LabBinaryTree.java` and `LabPriorityQueue.java` from `~dxu/handouts/labs/08`. These are the interfaces that we will use in lab. They are similar to those you saw in class, but simplified.
- Design and implement `ArrayBinaryTree` that implements `LabBinaryTree`. Note that different from lab7(or A6), this is now an array-based binary tree. Also note that you only need to implement a binary tree, *not* a binary search tree (in the sense that each node can have max two children, but no ordering of any kind is enforced, between the parent/children, or among the siblings). Think about what instance variables you will need.
- Start with the methods `size`, `isEmpty`. Implement the suggested helper methods below before moving onto `insert` and `toStringBreadthFirst`, which prints out the elements of the binary tree in breadth first traversal order (layer-by-layer). Breadth-first traversal is easy when you have an array-based binary tree.
  - `int parent(int i); // returns the index of the parent of child stored at i`
  - `int left(int i); // returns index of left child of parent stored at i`
  - `int right(int i); // returns index of right child of parent stored at i`
  - `void swap(int i, int j); // swaps the two nodes stored at indices i and j`
  - `int containsIdx(E element); // returns the index of the node containing element`
- Test your methods by creating a `ArrayBinaryTree<Integer>` object in a driver class, and insert the integers 1-20 into the tree.
- Proceed with implementing and testing `getRootElement` and `remove`. Note that an arbitrary binary tree is not guaranteed to be complete (the user may execute `remove` on any element at any time) and you need to decide how handle the case where positions may become `null` through out your tree. Hint: use `swap`
- Implement `ArrayHeap` that extends `ArrayBinaryTree` and implements `LabPriorityQueue`. Start with overriding `insert` so that elements can be inserted in heap order.
- Test by inserting the integers 9 down to 0 into the heap. If all goes well, your heap should look like this:
 

```

graph TD
    0((0)) --- 1((1))
    0 --- 4((4))
    1 --- 3((3))
    1 --- 2((2))
    3 --- 9((9))
    3 --- 6((6))
    2 --- 7((7))
    4 --- 8((8))
    4 --- 5((5))
      
```
- Proceed with implementing and testing `peek()`
- This is not required for lab, but think about how `remove` should be overridden to maintain heap order, and how you will implement `poll`.