5.3\(b\) When \(h\) is 0, there is only a single node, and \(2^0 = 2^1 - 1 = 1\).

Assume that in general, a complete binary tree of height \(h\) has between \(2^h\) and \(2^{h+1} - 1\) nodes.

There are two cases for complete binary trees of height \(h + 1\):

1. The left sub-tree has between \(2^h\) and \(2^{h+1} - 1\) nodes and the right sub-tree has \(2^h - 1\) nodes, or
2. The left sub-tree has \(2^{h+1} - 1\) nodes and the right sub-tree has between \(2^h\) and \(2^{h+1} - 1\) nodes.

Taking into account the root node, the first case has between 
\(1 + 2^h + 2^h - 1 = 2^{h+1}\) and \(1 + 2^{h+1} - 1 + 2^h - 1 = 3\cdot 2^h - 1\) nodes, and
the second case has between \(1 + 2^{h+1} - 1 + 2^h = 3\cdot 2^h\) nodes and \(1 + 2^{h+1} - 1 + 2^{h+1} - 1 = 2^{h+2} - 1\) nodes.

Thus, the number of nodes runs between \(2^{h+1}\) and \(2^{h+2} - 1\), which is the expected result.

5.3\(d\) \(\left\lceil \frac{n}{2} \right\rceil\)

5.3\(f\) The actual tree is

![Diagram of a binary tree with nodes 84, 57, 42, 54, 73, 60, 31, 25, 14, 81]
### 5.3g Some implementations are:

```cpp
// template <typename Type, int N>
Type Complete_binary_tree::parent( Type const &obj ) {
    int n = find( obj );
    if ( n == 0 ) {
        throw illegal_argument();
    }
    if ( n == 1 ) {
        throw underflow();
    }
    return array[n/2];
}

// template <typename Type, int N>
Type Complete_binary_tree::parent( Type const &obj ) {
    int n = find( obj );
    if ( n == 0 ) {
        throw illegal_argument();
    }
    if ( 2*n + 1 > complete_size ) {
        throw underflow();
    }
    return array[2*n + 1];
}
```