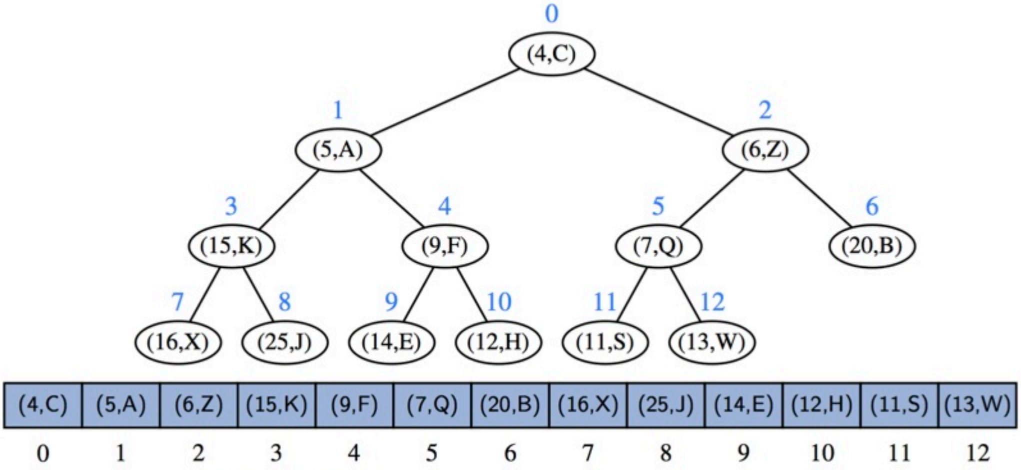
Array-Based Representation of a Complete Binary Tree

The array-based representation of a binary tree (Section 8.3.2) is especially suitable for a complete binary tree. We recall that in this implementation, the elements of the tree are stored in an array-based list A such that the element at position p is stored in A with index equal to the level number $f(p)$ of p, defined as follows:

- If p is the root, then $f(p) = 0$.
- If p is the left child of position q, then $f(p) = 2f(q) + 1$.
- If p is the right child of position q, then $f(p) = 2f(q) + 2$.

For a tree with of size n, the elements have contiguous indices in the range $[0, n-1]$ and the last position of is always at index $n-1$. (See Figure 9.4.)



(4,C)	(5,A)	(6,Z)	(15,K)	(9,F)	(7,Q)	(20,B)	(16,X)	(25,J)	(14,E)	(12,H)	(11,S)	(13,W)
0	1	2	3	4	5	6	7	8	9	10	11	12

Figure 9.4: Array-based representation of a heap.

The array-based heap representation avoids some complexities of a linked tree structure. Specifically, methods insert and removeMin depend on locating the last position of a heap. With the array-based representation of a heap of size n, the last position is simply at index $n-1$. Locating the last position in a heap implemented with a linked tree structure requires more effort. (See Exercise C-9.33.)

If the size of a priority queue is not known in advance, use of an array-based representation does introduce the need to dynamically resize the array on occasion, as is done with a Java ArrayList. The space usage of such an array-based representation of a complete binary tree with n nodes is $O(n)$, and the time bounds of methods for adding or removing elements become ***amortized***. (See Section 7.2.2.)