Sample solution to HW #4

12.6 Show that if the sybmols are sorted by frequency, Huffman's algorithm can be implemented in linear time.

(1) Use a linked list.

Use a linked list of sorted frequencies of symbols (in non-decreasing order). Also, initalize a pointer to the head of the list.

Iterate while more than one node on the list:

- create a new node which has as the children the two nodes at the beginning of the list (with the smallest weights);
- identify it by the weight equal to the sum of the children's weights, and
- insert it (according to the weight) into the proper place on the list by advancing the pointer;
- delete the two smallest elements.

The body of this loop maintains the invariant asserting that the list is non-empty, sorted in non-decreasing order of weights and that the weight of the node indicated by the pointer is not greater than the sum of the two smallest weights on the list. The complexity of a single execution of the body can only be bounded by n, but in total the number of times the pointer advances is bounded by the total number of deleted elements, which is $2n-2$.

(2) Use a queue (we concentrate only on the weights, binary tree construction is done as above).

(a) Initially we have a sorted list of symbols (by frequency in non-decreasing order) and an empty queue.

(b) If the queue has more than one element or the sorted list is not empty, check the two elements at the head of the queue (if they exist) and the two elements at the head of sorted list of symbols. Choose among them two smallest elements; delete the two elements from sorted list or queue; insert the sum of the two elements to the queue.

The correctness of this algorithm is guaranteed because elements in the queue are always sorted and the two smallest elements are always among the two elements at the head of the sorted list and two elements at the head of queue. The algorithm is linear because exactly $n-1$ elements are inerted into the queue and $2n-2$ elements are deleted; each operation taking a constant amount of time.

17.2

```
        a
      /   \
     b     c
      \   /
       d e
```

```
Inorder: bdaec

------------         ------------
| a0         ---->   | a1 b0        ---->
------------         ------------

------------         ------------
| a1 b1      ---->   | a1 d0        ---->
------------         ------------

-----------          ------------
| a1 d1      ---->   | a1           ---->
-----------          ------------

------------         ------------
| c0         ---->   | c1 e0        ---->
------------         ------------


-----------          ------------         ------------
| c1 e1      ---->   | c1           ---->  |
-----------          ------------         ------------


Preoder: abdce

------------          ------------
| a          ---->    | c b          ---->
------------          ------------

------------         ------------
```

```
| c d          ---->  | c            ---->
------------          -------------


------------          -------------
| e            ---->  |
------------          -------------
```

17.6 Prove that in a binary tree, the number of full nodes plus one is equal to the number of leaves.

Method 1

Let n be the number of nodes in a binary tree and n_1, n_2 and n_0 be the number of nodes with one child, two children and no children, respectively.
Obviously
$n = n_0 + n_1 + n_2$
Since n can also be counted by number of edges of the tree plus one, $n = n_1 + 2n_2 + 1$, each edge connecting a node with its child(ren).
Together we have
$n_0 + n_1 + n_2 = n_1 + 2n_2 + 1.$
Thus, $n_0 = n_2 + 1.$

Method 2
Proof by induction on the number of full nodes.