Part I Solution

1. Binary tree properties.

- There are n nodes. Each node has two links, so there are $2n$ links. Each node but the root has one incoming link from its parent, which accounts for $n-1$ links. The rest are therefore null.

- Let F be the number of full nodes, H be the number of half nodes, and L be the number of leaves. Clearly, we have:

$$F + H + L = n,$$

where n is the total number of nodes in the binary tree. The total number of links in a binary tree having n nodes is $2n$. Each full node has 2 non-null links and each half node has 1 non-null link. The rest of the links are null. From the previous question, we know that there are $n+1$ null links. Thus we have:

$$2n - (2F + H) = n + 1.$$

From the above two equations, we can conclude that $F + 1 = L$.

2. Linear time algorithm to test whether a binary tree is a binary search tree.

There are different ways to do this. Perhaps the most intuitive way is to perform an in-order traversal and verify that it yields an ordered sequence of values. We can do this in one pass as follows:

```
boolean isBST( BSTNode<T> p ) {
  boolean returnVal = true;
  if ( p.left != null )
    returnVal = returnVal &&
      (p.left.el.compareTo(p.el) < 0 ) &&
      isBST( p.left );

  if ( p.right != null )
    returnVal = returnVal &&
      (p.right.el.compareTo(p.el) >= 0 ) &&
      isBST( p.right );

  return returnVal;
}
```

To analyze the complexity of this, let's assume a perfect binary tree and let $T(n)$ be the number of node traversals. Inspecting the code above, we see that $T(n)$ satisfies the following recurrence:

$$T(1) = 1$$
$$T(n) = 2T(\frac{n-1}{2}) + 1$$

Using the substitution method, the solution is found to be $T(n) = n$.

3. Complexity of building a BST.

We'll analyze the number of node traversals required to insert n elements in the worst and best cases.

- *Worst case*:
  In the worst case, the elements are inserted in sorted order. In this case, the tree will degenerate into a list. To insert the first element, we need 0 traversals, to insert the second element, we need 1 traversal, and so on. Thus, the total number of traversals is:

$$\begin{aligned} &\sum_{i=1}^{n} i-1 \\ &= \sum_{i=0}^{n-1} i \\ &= \Theta(n^2). \end{aligned}$$

- *Best case*:
  In the best case, every insertion yields a binary tree that stays balanced. In this case, the total number of traversals is equal to the internal path length $P(n)$ of the binary tree. In class, we derived the following formula for the internal path length of a perfect binary tree:

$$P(n) = \sum_{k=0}^{\lg n} k2^k = (n+1)(\lg(n+1)-1)-(n+1)+2.$$

  Thus, in the best case, we need $\Theta(n \lg n)$ traversals.