Question 1

$* \ BTree\ memory$

$$= n \text{ nodes} \cdot (2t-1)\frac{\text{blocks}}{\text{node}} \cdot D\frac{\text{memory}}{\text{block}} + 1\frac{\text{memory}}{\text{pointer}} \cdot 2t\frac{\text{pointers}}{\text{node}} \cdot n \text{ nodes}$$

$$= n \cdot (2t-1) \cdot D + 2t \cdot n$$

$* \ MerkleBTree\ memory$

$$= 20\ \frac{\text{memory}}{\text{hashValue}} \cdot 1\frac{\text{hashValue}}{\text{node}} \cdot n \text{ nodes} + 1\frac{\text{memory}}{\text{pointer}} \cdot 2t\frac{\text{pointers}}{\text{node}} \cdot n \text{ nodes}$$

$$= 20 \cdot n + 2t \cdot n$$

$$\frac{BTree.memory}{MerkleBtree.memory} = \frac{(2t-1) \cdot D \cdot \boldsymbol{n} + 2t \cdot \boldsymbol{n}}{20 \cdot \boldsymbol{n} + 2t \cdot \boldsymbol{n}} = \frac{(2t-1) \cdot D + 2t}{2t + 20} = \frac{D + 1 - \frac{D}{2t}}{1 + \frac{20}{2t}}$$

$$= \sim D\ (assuming\ 2t\ has\ a\ large\ value)$$

Question 2

During $Insertion$ there's no need to update the hash values of ***all*** the values of the Merkle-B-Tree. For example, Leaf's in which the element isn't inserted don't change; and since the hash values of leaves are determined by the leaves themselves, the hash value doesn't need to be updated.

The node-types that we do need to update:

1. The node where the element is inserted.
2. Nodes that the $SplitChild$ operation of the B-tree has affected.
3. The parents of the previous 2 node-types, their parents, and in general – all their "ancestors" – all the way up to the root.

We'll do this by adding a Boolean $needUpdate$ field which is by default initialized as false to all nodes, and two new functions called "Update" and "Update2" (below).

$\boldsymbol{Update}\ (Tree\ T, Node\ toUpdate)$

1. $a \leftarrow toUpdate$
2. $\boldsymbol{while}\ a \neq\ T.root\ and\ a.needUpdate\ \neq true$
3. $\quad a.needUpdate = true$
4. $\quad a \leftarrow a.parent$
5. $\boldsymbol{if}\ toUpdate.isLeaf\ \boldsymbol{then}$
6. $\quad \boldsymbol{Update2}\ (T.root)$

$\boldsymbol{Update2}\ (Node\ toUpdate)$

1. $toUpdate.needUpdate \leftarrow false$
2. $toUpdate.updateTheHashValue$
3. $\boldsymbol{for}\ i = 1\ \boldsymbol{upto}\ toUpdate.numberOfChildren$
4. $\quad \boldsymbol{if}\ toUpdate.Child(i).needUpdate = true\ \boldsymbol{then}$
5. $\quad\quad \boldsymbol{Update2}(\boldsymbol{toUpdate.Child(i)})$

The $Update$ function will be added two times at the end of the $SplitChild$ operation, one for each of the two children resulting from the split. The changed father will be updated by these $Update$ calls. The update function will also be added to the end of the $Insert$ function, for the leaf in which the element was inserted.

Question 3

The values of $h_{0<->4}$ aren't randomized with each use of the has function, but are constant. This is important, because we expect that applying the function twice on the same function will return the same value.

The values of $h_{0<->4}$ are public because we want different people using SHA-1 on a certain message to reach the same result. This doesn't pose a problem, since SHA-1 is a one-way function (ideally, although not necessarily against a well-funded attacker), and knowing the details of how it works won't help an attacker.