Binary Search Tree

Problem 1: Which of the followings are TRUE?
(1) If we use in-order traversal on any binary tree, the resulting values are always sorted.
(2) If we use in-order traversal on any binary search tree, the resulting values are always sorted.
(3) If we use pre-order traversal on any binary search tree, the resulting numbers are always sorted.

(4) There exists a binary tree such that the in-order, pre-order, post-order, and levelorder traversals all give the same output.
(5) If a binary tree is empty, then the outputs from pre-order, in-order, post-order, and level-order traversals are the same.

(6) If all the nodes in a binary tree do not have the right child node, then the outputs from in-order and post-order traversals are the same.
(7) If all the nodes in a binary tree do not have the right child node, then the outputs from pre-order and post-order traversals are the same.
(8) If all the nodes in a binary tree do not have the right child node, then the outputs from pre-order and in-order traversals are the same.

(9) If all the nodes in a binary tree do not have the left child node, then the outputs from in-order and post-order traversals are the same.
(10) If all the nodes in a binary tree do not have the left child node, then the outputs from pre-order and post-order traversals are the same.
(11) If all the nodes in a binary tree do not have the left child node, then the outputs from pre-order and in-order traversals are the same.

(12) The maximum number of leaf nodes at the L-th level in a binary tree is 2 assuming the root node is at Level 1.
(13) In a binary tree, assume P is the number of leaf nodes, and Q is the number of nodes that have 2 children. Then it's ALWAYS the case that P = Q + 2.

Problem 2: Implement insert, find, remove, and getMax for a BST.

```
struct Node
{
    Node(const int myVal) {
        value = myVal;
        left = right = parent = nullptr;
    }
    int value;
    Node *left,*right;
};

class BinarySearchTree
{
public:
    BinarySearchTree() { m_root = nullptr; }
    ~BinarySearchTree() { freeTree(m_root); }
    Node* getRoot() { return m_root; }
    void insert(int value, Node* ptr);
    bool find(int value, Node *ptr);
    void remove(Node* ptr);
   int getMax(Node *ptr);
private:
    Node *m_root;
    void freeTree(Node *cur);
};

int BinarySearchTree::GetMax(Node *ptr) {









}

void BinarySearchTree::insert(int value, Node* ptr) {









}
```

```
bool BinarySearchTree::find(int value, Node* ptr){









}
void BinarySearchTree::remove(Node* ptr){









}
```

Hash Table

Problem 1: What is a Closed Hash Table? What is an Open Hash Table?

Problem 2: What is the load factor?

Problem 3: Compare the performance of Hash Table and BST.

Problem 4: What is a Hash Set?