Problem 1:
(2) : inorder traversal of BST is alway sorted

(4)(5): Tree is empty or only root exists

(6)(11): preorder: print(value), traverse(left), traverse(right)
inorder: traverse(left), print(value), traverse(right)
postorder: traverse(left), traverse(right), print(value)

(12): 2^(L-1)
(13): P=Q+1

```
int BinarySearchTree::getMax(Node *ptr) {
        if (ptr == nullptr)
                return(-1); // empty
        while ( ptr->right != nullptr )
                ptr = ptr->right;
        return(ptr->value);
}

void BinarySearchTree::insert(int value, node * ptr) {
      if (ptr == nullptr) {
         ptr = new node(value);
      }else if (value < ptr->value)
         insert(value, tree->left);
       else if (value > ptr->value)
         insert(value, tree->right);
   }


bool BinarySearchTree::find(int value, node * ptr) {
       if (ptr == nullptr) return false;
      if(ptr->value = value) return true;
      if (value < ptr->value)
         find(value, tree->left);
       else if (value > tree->value)
         insert(value, tree->right);
   }

bool BinarySearchTree:remove()
   //if the node to_be_removed has no child or have only one child
   if (to_be_removed->left==NULL|| to_be_removed->right==NULL){
      TreeNode<ItemType,cmp>*new_child;
      if (to_be_removed->left==NULL)
         new_child=to_be_removed->right;
      else new_child=to_be_removed->left;
```

```
    if (parent==NULL)
        root=new_child;
    else if(parent->left==to_be_removed)
        parent->left=new_child;
    else parent->right=new_child;
    delete to_be_removed;
    return;
  }
  //Neither subtree is empty
  //Find the largest element of the left subtree
  TreeNode<ItemType,cmp>*largest_parent=to_be_removed;
  TreeNode<ItemType,cmp>*largest=to_be_removed->left;
  while (largest->right!=NULL){
    largest_parent=largest;
    largest=largest->right;
  }
  //largest contains the largest child in the left subtree
  //move contents, unlink child
  to_be_removed->data=largest->data;
  if (largest_parent==to_be_removed)
    largest_parent->left=largest->left;
  else largest_parent->right=largest->left;
  delete largest;
```

```
10-17
20-24
10 11 12 13 14 15 16 17 -1 -1
no room left in hash table!!!
no room left in hash table!!!
10 11 12 13 14 15 16 17 20 21
```

max insert/buckets

speed O(1), O(logn)
load factor unlimited space
waste memory, use as much as needed
no ordering, ordering