```
#ifndef BT_NODE_H
#define BT_NODE_H

struct btNode
{
   int data;
   btNode* left;
   btNode* right;
};

// pre:  bst_root is root pointer of a binary search tree (may be 0 for
//       empty tree) and dumpArray has the base address of an array large
//       enough to hold all the data items in the binary search tree
// post: The binary search tree has been traversed in-order and the data
//       values are written (as they are encountered) to dumpArray in
//       increasing positional order starting from the first element
void dumpToArrayInOrder(btNode* bst_root, int* dumpArray);
void dumpToArrayInOrderAux(btNode* bst_root, int* dumpArray, int&
dumpIndex);

// pre:  (none)
// post: dynamic memory of all the nodes of the tree rooted at root has been
//       freed up (returned back to heap/freestore) and the tree is now
empty
//       (root pointer contains the null address)
void tree_clear(btNode*& root);

// pre:  (none)
// post: # of nodes contained in tree rooted at root is returned
int bst_size(btNode* bst_root);

////////////////////////////////////////////////////////////////////////////
/

// pre:  bst_root is root pointer of a binary search tree (may be 0 for
//       empty tree)
// post: If no node in the binary search tree has data equals insInt, a
//       node with data insInt has been created and inserted at the proper
//       location in the tree to maintain binary search tree property.
//       If a node with data equals insInt is found, the node's data field
//       has been overwritten with insInt; no new node has been created.

// write prototype for bst_insert here
void bst_insert(btNode*& bst_root, int insInt);
// pre:  bst_root is root pointer of a binary search tree (may be 0 for
//       empty tree)
// post: If remInt was in the tree, then remInt has been removed, bst_root
//       now points to the root of the new (smaller) binary search tree,
//       and the function returns true. Otherwise, if remInt was not in the
//       tree, then the tree is unchanged, and the function returns false.

// write prototype for bst_remove here
bool bst_remove(btNode*& bst_root, int remInt);
// pre:  bst_root is root pointer of a non-empty binary search tree
// post: The largest item in the binary search tree has been removed, and
//       bst_root now points to the root of the new (smaller) binary search
//       tree. The reference parameter, removed, has been set to a copy of
//       the removed item.

// write prototype for bst_remove_max here
void bst_remove_max(btNode*& bst_root, int& data);

#endif
```