```cpp
#include "btNode.h"

void bst_insert (btNode*& bst_root, int newInt)
{
    //if their is no values in the tree we create a new value and add it as the first value
    if (bst_root == 0){
        btNode* new_root = new btNode;
        new_root->data = newInt;
        new_root->left = 0;     //right child will become null
        new_root->right = 0;    //left child is also assigned null value
        bst_root = new_root;
    }
    //variable to traverse the list
    btNode* curr = bst_root;

    while(curr != 0){
        //entering the left side of the binary tree in order to insert a new leaf
        if (curr->data > newInt){
            if (curr->left == 0){
                curr->left = new btNode;
                curr->left->data = newInt;
                curr->left->left = 0;   //right child is assigned a null value
                curr->left->right = 0;  //left child is assigned a null value
            }
            else {
                curr = curr->left;   // else we traverse the list until we find an empty spot
            }
        }
        else if (curr->data < newInt){
                //entering the right side of the binary tree in order to insert a new leaf
                if (curr->right == 0){
                    curr->right = new btNode;
                    curr->right->data = newInt;
                    curr->right->left = 0;   //right child is assigned a null value
                    curr->right->right = 0;  //left child is assigned a null value
            }
            else {
                curr = curr->right;  // else we traverse the list until we find an empty spot
            }
        }
    else return;
    }
}

bool bst_remove(btNode*& bst_root, int remInt)
{
    //if the tree is empty exit the function
    if (bst_root == 0) return false;

    if (remInt != bst_root->data){
        //if the target is greater than the root
        if (remInt > bst_root->data){
            return bst_remove(bst_root->right, remInt);
        }
        //if the target is less than the root
        else{
            return bst_remove(bst_root->left, remInt);
        }
    }
    if (bst_root->left == 0){
        btNode* older_root = bst_root;
        if (bst_root->right != 0){
            bst_root = bst_root->right;
        }
        else{
            bst_root = 0;
        }
        delete older_root;
        return true;
    }
    else {
        bst_remove_max(bst_root->left, bst_root->data);
        return true;
    }
    return false;
}

void bst_remove_max(btNode*& bst_root, int& data)
{
    //if the tree is empty exit the function
    if (bst_root == 0) return;

    if (bst_root->right == 0){
        btNode* temp = bst_root;
        data = bst_root->data;
        bst_root = bst_root->left;
        delete temp;
    }
    else{
        bst_remove_max(bst_root->right, data);
    }
    return;
}


void dumpToArrayInOrder(btNode* bst_root, int* dumpArray)
{
   if (bst_root == 0) return;
   int dumpIndex = 0;
   dumpToArrayInOrderAux(bst_root, dumpArray, dumpIndex);
}

void dumpToArrayInOrderAux(btNode* bst_root, int* dumpArray, int& dumpIndex)
{
   if (bst_root == 0) return;
   dumpToArrayInOrderAux(bst_root->left, dumpArray, dumpIndex);
   dumpArray[dumpIndex++] = bst_root->data;
   dumpToArrayInOrderAux(bst_root->right, dumpArray, dumpIndex);
}

void tree_clear(btNode*& root)
{
   if (root == 0) return;
   tree_clear(root->left);
   tree_clear(root->right);
   delete root;
   root = 0;
}

int bst_size(btNode* bst_root)
{
   if (bst_root == 0) return 0;
   return 1 + bst_size(bst_root->left) + bst_size(bst_root->right);
}
```