```
#include "btNode.h"
#include <iomanip>

// write definition for bst_insert here
void bst_insert(btNode*& bst_root, int insInt)
{
  if(bst_root == 0)
  {
     btNode* new_root = new btNode;
     new_root->data = insInt;
     new_root->left = new_root->right = 0;
     bst_root = new_root;
     return;
  }
  btNode* cursor = bst_root;

  while (cursor != 0)
  {
     if (cursor->data > insInt)
     {
        if(cursor->left == 0)
        {
           cursor->left = new btNode;
           cursor->left->data = insInt;
           cursor->left->left = cursor->left->right = 0;
           return;
        }
        else
        {

           cursor = cursor->left;
        }
     }
     else if(cursor->data < insInt)
     {

        if(cursor->right == 0)
        {
           cursor->right = new btNode;
           cursor->right->data = insInt;
           cursor->right->left = cursor->right->right = 0;
           return;
        }
        else
        {
           cursor = cursor->right;
        }
     }
     else
     {

        return;
     }
  }
}
```

```
// write definition for bst_remove here
bool bst_remove(btNode*& bst_root, int remInt)
{
  if(bst_root == NULL)
  {
    return false;
  }

  if(remInt< bst_root->data)
  {
    return bst_remove(bst_root->left, remInt);
  }

  if(remInt > bst_root->data)
  {
    return bst_remove(bst_root->right, remInt);
  }

  if(remInt == bst_root->data)
  {
    if(bst_root->left == NULL)
    {
        btNode* oldroot_ptr = bst_root;
        bst_root = bst_root->right;
        delete oldroot_ptr;
        return true;
    }

    bst_remove_max(bst_root->left, bst_root->data);
    return true;
   }

   return false;
}

// write definition for bst_remove_max here
void bst_remove_max(btNode*& bst_root, int& data)
{
  if(bst_root->right != NULL)
  {
    return bst_remove_max(bst_root->right, data);
  }
  data = bst_root->data;

  if(bst_root->left == NULL)
  {
        bst_root = NULL;
        return;
  }

  bst_root = bst_root->left;
  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);
}
```