Xavier Kuehn

1. Write the implementations of `tree_copy` and `operator=`

```
template <class Item>
btn<Item>* tree_copy(const btn<Item> *&root_ptr){
    if (root_ptr == NULL) return NULL;
    btn<Item> *tmp_left = tree_copy(root_ptr->left());
    btn<Item> *tmp_right = tree_copy(root_ptr->right());
    return new btn<Item> (root_ptr->data(), tmp_left, tmp_right);
}

template <class Item>
void bag<Item>::operator=(const bag<Item> &src){
   if (this == &src) return;
   tree_clear(root_ptr);
   root_ptr = tree_copy(src.root_ptr);
}
```

2. For the bag class defined in Appendix 1, please complete the `insert` function.

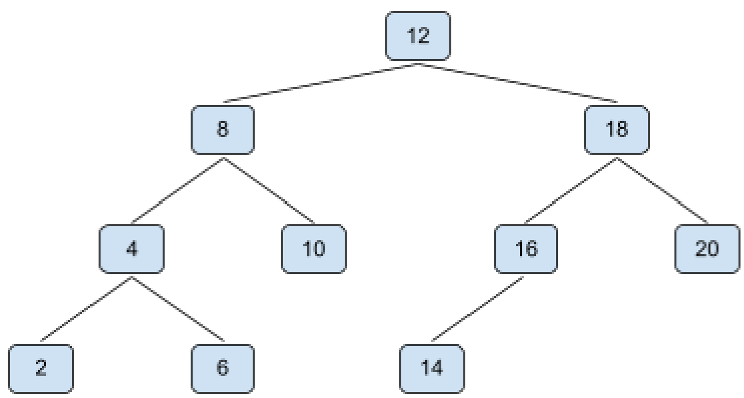
```
template <class Item>
void bag<Item>::insert (const Item &entry) {
    if (root == NULL) {
        root = new btn<Item>(entry);
        return;
    }
    else {
        btn<Item> *cursor = root;
        while (cursor != NULL) {
            if (entry <= cursor->data()) {
                if (cursor->left() == NULL) {
                    cursor->left() = new btn<Item>(entry);
                    return;
                }
                cursor = cursor->left();
            }
            else {
                if (cursor->right() == NULL) {
                    cursor->right() = new btn<Item>(entry);
                    return;
                }
                cursor = cursor->right();
            }
        }
    }
}
```

3. If there is one level below the root of 1000 nodes, then the root must contain 999 nodes so that it is a B-tree. Each child node must have at least 1000 entries and at most 2000 entries. Thus the range of number of entries in the B-tree is 999 + 1000*[1000,2000].

4. Write a function to perform a left-right rotation on the following AVL tree.

```
template <class Item>
binary_tree_node<Item>* left_right_rotation(binary_tree_node<Item>*&
parent){
   binary_tree_node<Item>* temp = root->right();
   root->right() = tmp->left();
   tmp->left() =root;
   root = tmp;
   tmp = root->left();
   root->left() = tmp->right();
   tmp->right() = root;
   root = tmp;
}
```

5. Add the following numbers (left to right) to an AVL tree and draw the answer.



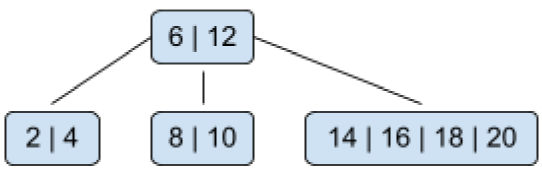
6. Write the implementation of a function that balances a tree rooted at temp.

```
template <class Item>
binary_tree_node<Item>* balance(binary_tree_node <Item>*& temp){
   int temp_bf = diff(temp);
   if (height(temp) <= 2) return temp;
   if (temp_bf > -2 && temp_bf < 2) {
       balance(temp->left());
       balance(temp->right());
   }
   else
   {
       // temp balance factor >= 2
       if (temp_bf >= 2)
```

```
        {
            if (diff(temp->left()->right() == nullptr)) right_rotation(temp);
            else left_right_rotation(temp);
        }

        // temp balance factor <= -2
        if (temp_bf <= -2) {
            if (diff(temp->right()->left() == nullptr)) left_rotation(temp);
            else right_left_rotation(temp);
        }
    }
    return temp;
}
```

7. Please add the following numbers to a B-Tree with MIN = 2. Draw the final tree.



8. Write the implementation of a function that removes the max element from a BST.

```
void bstRemoveMax(binary_tree_node<Item> *&root, Item &target) {
   if (root->right() == NULL) {
      binary_tree_node<Item> *tmp = root;
      target = root->data();
      root = root->left();
      delete tmp;
   }
   else bstRemoveMax(root->right(), target);
}
```

9. Write the implementation of a function that flips a tree to its mirror image.

```
template <class Item>
void flip(binary_tree_node<Item>* root_ptr) {
   if (root_ptr->left() == NULL && root_ptr->right() == NULL) return;
   else
   {
       binary_tree_node<Item>* temp = root_ptr->left();
       root_ptr->left() = root_ptr->right();
       root_ptr->right() = temp;
       flip(root_ptr->left());
       flip(root_prt->right());
   }
}
```

10. Write the outputs of programs 1-4.

1. `Output:    Derived's destructor`
   ```
               Base2's destructor
               Base1's destructor
   ```

2. `Output:    Compiler error, Derive does not have access to i and j member variables (they are private within Base).`

3. `Output:    Inside Q`

4. `Output:    Compiler error from line 10, cannot downcast base class pointer to derived class object pointer;`