Project 8 Implementation Notes

Faysal Khatri

bstSort()

Adds all elements to a temp BST, and then grabs the min from the temp BST for each index in data[].

```
	public static <T> void bstSort(T[] data) {
		LinkedBinarySearchTree sortTree = new LinkedBinarySearchTree();
		for (T element : data) {
			sortTree.addElement(element);
		}

		for (int i=0; i<data.length; i++) {
			data[i] = (T) sortTree.removeMin();
		}
	}
```

LinkedBinarySearchTree

find()

Recursively traverses the tree until a match is found or the tree is exhausted. Had to add null checks to prevent NPEs.

```
	public T find(T targetElement) throws EmptyCollectionException
	{
		T result = null;

		if (isEmpty())
			throw new EmptyCollectionException("LinkedBinarySearchTree");
		else
		{
			if (((Comparable<T>)targetElement).equals(root.element))
			{
				result = root.element;
			}
			else if (((Comparable<T>)targetElement).compareTo(root.element) < 0 && root.getLeft() != null) {
				result = find(targetElement, root.getLeft());
			}
			else if (root.getRight() != null) {
				result = find(targetElement, root.getRight());
			}
		}
		return result;
	}
	public T find(T targetElement, BinaryTreeNode<T> node) throws ElementNotFoundException {
		T result = null;
		if (node == null) {
			throw new ElementNotFoundException("BST is null");
		}
		else {
			if (((Comparable<T>)targetElement).equals(node.element))  {
				result = node.element;
			}
			else if (((Comparable<T>)targetElement).compareTo(node.element) < 0 && node.getLeft() != null) {
				result = find(targetElement, node.getLeft());

			}
			else if (node.getRight() != null) {
				result = find(targetElement, node.getRight());

			}

		}
		return result;
	}
```

findMin()

Finds the left-most leaf or internal node.

```
public T findMin() throws EmptyCollectionException
	{
			T result = null;

			if (isEmpty())
				throw new EmptyCollectionException("LinkedBinarySearchTree");
			else
			{
				if (root.left == null)
				{
					result = root.element;
				}
				else
				{
					BinaryTreeNode<T> current = root.left;
					while (current.left != null)
					{
						current = current.left;
					}
					result =  current.element;
				}
				modCount++;
			}
			return result;
	}
```

removeMax()

finds the right-most node, returns it and replaces it if it is an internal node

```
public T removeMax() throws EmptyCollectionException
{
	T result = null;

	if (isEmpty())
		throw new EmptyCollectionException("LinkedBinarySearchTree");
	else
	{
		if (root.right == null)
		{
			result = root.element;
			root = root.left;
		}
		else
		{
			BinaryTreeNode<T> parent = root;
			BinaryTreeNode<T> current = root.right;
			while (current.right != null)
			{
				parent = current;
				current = current.right;
			}
			result =  current.element;
			parent.right = current.left;
		}

		modCount++;
	}

	return result;
}
```

LinkedBSTOrderedSet

addElement()

```
	public void addElement(T element) {
		if (!contains(element)) {
			super.addElement(element);
		}
	}
```