I have included a zip file that has the source code for the programs that I was working with. I downloaded it from the website mentioned in the text-book, and then modified it a little, and added my functions.

### Chapter 3:

**Exercise 3.**

My method of doing the SplitLists function is the same for both the Array version and the Linked List version. I was able to do this because I only use member functions to perform the split, and don't access the data directly. I also passed a reference to the list, instead of copying it. This is so that it works with a linked list, and also has the added advantage of using less memory. Here is my function:

void UnsortedType::SplitLists(UnsortedType& list, ItemType item, UnsortedType& list1, UnsortedType& list2)

// Pre: list has been initialised and is not empty

// Post: list1 contains all the items of list whose keys are less

// than os equal to item's key;

// list2 contains all the items of list whose keys are greater

// than item'; key.

{

// Some variables

ItemType currentItem;

int length = list.GetLength();

// First, start at the beginning of list

list.ResetList();

// And make sure the two other lists are empty

list1.MakeEmpty();

list2.MakeEmpty();

for(int counter = 0; counter < length; counter++)

{

currentItem = list.GetNextItem();

if(GREATER == currentItem.ComparedTo(item))

list2.PutItem(currentItem);

else

list1.PutItem(currentItem);

}

}

### Chapter 4:

**Exercise 4.**

**Part a.**

Here is a prototype for my function. I used references again:

void MergeLists(SortedType& list1, SortedType& list2, SortedType& result);

**Part b,c.**

I again am able to use the same function for both array and linked list versions of the ADT. Here is my function:

void MergeLists(SortedType& list1, SortedType& list2, SortedType& result)

{

int length = list1.GetLength();

int i;

list1.ResetList();

for(i = 0; i < length; i++) {

result.PutItem(list1.GetNextItem());

}

length = list2.GetLength();

list2.ResetList();

for(i = 0; i < length; i++) {

result.PutItem(list2.GetNextItem());

}

}

**Part d.**

The Big-O Complexity of the program is as follows:

getlength = O(1) +

resetlist = O(1) +

for loop of putitem = O(N) +

getlength = O(1) +

resetlist = O(1) +

for loop of putitem = O(N) +

total = O(1) + O(1) + O(N) + O(1) + O(1) + O(N) = O(N)

**Exercises 5 and 6.**

Because of the way I wrote the function, it's the same in both the Array and Linked List version.

void SortedType::MergeLists(SortedType& list1, SortedType& list2, SortedType& result)

{

int length = list1.GetLength();

int i;

list1.ResetList();

for(i = 0; i < length; i++) {

result.PutItem(list1.GetNextItem());

}

length = list2.GetLength();

list2.ResetList();

for(i = 0; i < length; i++) {

result.PutItem(list2.GetNextItem());

}

}