James J. Kim
Homework 4

**3.2, 3.5, 4.1, 4.3, 5.1, 5.3, 5.6**

**3.2 In Sorting Algorithms : Section 9 : Merge Sort Pseudocode we discussed the high-level pseudocode for the merge sort algorithm. I wrote three methods: recursiveMergeSort(), merge(), and copyRest().**

**Continuing the previous exercise, how many times will recursiveMergeSort() be called when sorting the list of Exercise 3.1. Include the original nonrecursive call to recursiveMergeSort() and all of the recursive calls in the sum**

*list = { 4, 2, 7, 3, 5, 13, 11, 8, 6, 2 }*

*Utilizing the formula of a(n) = 5n lgn, we can determine that the list with 10, we the recursive method is called 50 times.*

**3.5 Consider list = { 5, 4, 2, 9, 1, 7, 3, 8, 6 } and the quick sort algorithm. Assume we always choose the first list item in the list being partitioned as the pivot. Trace the partition method showing how list is partitioned into listL and listR. To get you started, here is the format of what I am looking for in your solution (next page):**

*list = { 5, 4, 2, 9, 1, 7, 3, 8, 6 },*
*pivot = 5, leftIndex = -1, rightIndex = 9*
*While loop pass 1:*

> *increment leftIndex++ ; leftIndex == 0*
> *while list[leftIndex] < list[rightIndex]; increment leftIndex++*
> *list[0] == 5 && 5 is  NOT < 5; decrement rightIndex*
> *rightIndex--; rightIndex == 8*
> *list[8] == 6 && 6 > 5; decrement rightIndex*
> *rightIndex--; rightIndex == 7*
> *list[7] == 8 && 8 > 5; decrement rightIndex*
> *rightIndex--; rightIndex == 6*
> *list[6] == 3 && 3 is NOT > 5;*
> *leftIndex == 0 < rightIndex == 6, swap list[0] == 5 with list[6] == 3*
> *{ 3, 4, 2, 9, 1, 7, 5, 8, 6 }*

*{ 3, 4, 2, 9, 1, 7, 5, 8, 6 }*
*While loop pass 2:*

```
        while list[leftIndex] < list[rightIndex]; increment leftIndex++
        increment leftIndex++; leftIndex == 1
        list[1] == 4 && 4 is < 5,
        increment leftIndex++; leftIndex == 2
        list[2] == 2 && 2 is < 5,
        increments leftIndex == 3
        list[3] == 9 && 9 is NOT < 5; leftIndex == 3
        decrement rightIndex--; rightIndex == 5
        list[5] == 7 is > 5
        decrement rightIndex--; rightIndex == 4
        list[4] == 1 && 1 is NOT > 5
        leftIndex == 3 < rightIndex == 4, so swap list[3] == 9 with list[4] == 1:
        { 3, 4, 2, 1, 9, 7, 5, 8, 6 }
```

*{ 3, 4, 2, 1, 9, 7, 5, 8, 6 }*
*While loop pass 3:*

```
        while list[leftIndex] < list[rightIndex]; increment leftIndex++
        increment leftIndex++; leftIndex == 4
        list[4] == 9 && 9 is NOT < 5; leftIndex == 4
        decrement rightIndex--; rightIndex == 3
        list[3] == 1 && 1 is NOT > 5
        leftIndex == 4 !< rightIndex == 3
```

*While loop terminates because leftIndex = 4 and rightIndex = 3*
*partition() returns 2 so listL = {3,4,2,1}, listR = {9,7,5,8,6}*

**4.1 (Include your modified DList.java source code file in your homework solution zip archive) Using whatever Java IDE you prefer, create a project and add DList.java and DListTest.java to it (these files are provided in the Week 6 Source zip archive). Modify DList to implement a method that removes all occurrences of a specific integer from the list.**
**Here is the pseudocode:**

```
Method removeAll(In: Integer pData) Returns Nothing
    Define index variable i and
    initialize i to 0
    While i < the size of this Dlist
        Do If get(i) equals pData
            Then remove(i)
        Else Increment i
        End If
    End While

End Method removeAll
```

**Next, modify DListTest() to add test case 21 which tests that removeAll() works correctly.**

*Source Code Included*

**4.3**
**(Include DList.java in your solution zip archive) Here is the Java implementation of three useful methods (which are not currently in Dlist).**

**Removes the head node from this DList. It would be inadvisable to call this method on an empty list because we do not check for that condition. Returns the data stored in the head node.**

```
protected Integer removeHead() {
    Integer data = getHead().getData();
    if (getSize() == 1) {
        setHead(null);
        setTail(null);
    } else {
        getHead().getNext().setPrev(null);
        setHead(getHead().getNext());
    }
    setSize(getSize() - 1);
    return data; }
```

**Removes an interior node pNode from this DList. It would be inadvisable to call this method
when pNode is null because we do not check for that condition. Returns the data stored in pNode.**

```
    protected Integer removeInterior(Node pNode) {
        Integer data = pNode.getData();
        pNode.getPrev().setNext(pNode.getNext());
        pNode.getNext().setPrev(pNode.getPrev());
        setSize(getSize() - 1); return data;
    }
```

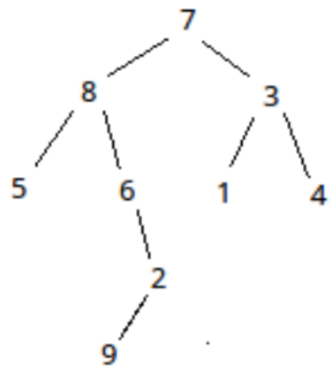
**Removes the tail node from this DList. It would be inadvisable to call this method on an empty list because we do not check for that condition. Returns the data stored in the tail node.**

```
    protected Integer removeTail() {
        Integer data = getTail().getData();
        if (getSize() == 1) {
            setHead(null);
            setTail(null);
        } else {
            getTail().getPrev().setNext(null);
            setTail(getTail().getPrev());
        }
        setSize(getSize() - 1);
        return data;
        }
```

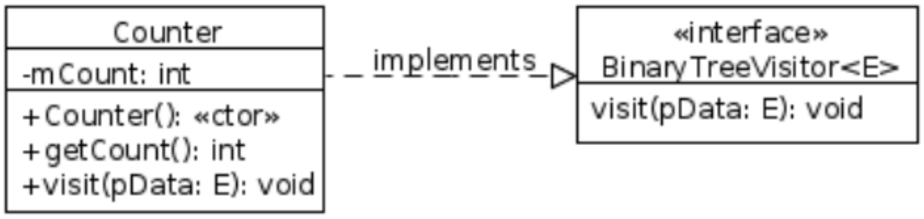
**Using these three methods, rewrite the provided remove(index) method to make the code in that method simpler and more readable (my new and improved remove() method is half-a-dozen lines of code). Be sure to run the test cases in DListTest to ensure that your new remove() method still works correctly. Also, make sure remove() throws an IndexOutOfBoundsException if pIndex is less than 0 or greater than or equal to getSize().**

*Source Code Included*

5.1 Consider this binary tree. (a) List the descendants of node 8. (b) List the ancestors of 1. (c) List the leaf nodes. (d) List the internal nodes. (e) What are the levels of nodes 3, 1, and 9? (f) What is the height of the tree? (g) What is the height of the subtree rooted at 6? (h) Is this a full binary tree? Explain. (i) Explain how we could transform this tree to be a complete binary tree, i.e., state which nodes we would move and where we would move them to.



5.3 (Include your modified BinaryTree.java source code file in your homework solution zip archive) Add a method int getSize() to BinaryTree that returns the size of the binary tree where the size of the tree is defined to be the number of nodes. Here is how I want you to implement this method. Write a local class (see Week 3 : Objects and Classes II : Section 2) in getSize() named Counter which implements the BinaryTree Visitor interface:



5.6 A BST is created (it is initially empty) where the key associated with the data in each node is an integer. Elements are added to the BST with these keys in this order: 5, 4, 8, 7, 6, 9, 3, 2, 1. (a) Draw the resulting BST. (b) What is the height of the tree?