Assignment 4 – Recursive and Sorting Methods

Required Reading

Java Foundations – Chapter 13 – Linked Structures
Chapter 17 – Recursion
Chapter 18 – Searching and Sorting
Chapter 19 – Trees

Instructions – PLEASE READ *(notice **bold** and underlined phrases)*

This assignment has four parts:

A. Written – Answer the questions, **submit in Lab Tutorial Nov 25th**
B. Written – Answer the questions, **submit to CourSys by Dec 2nd**
C. Programming – Code must be well commented, compile/test, then submit
D. Submission – Submit specified assignment files by deadline

1. **This assignment must be done individually. You may discuss ideas with others, but you must answer all questions and write all the code within your group. Plagiarism by students will result penalties that may include receiving zero for the assignment.**

2. **All Files Submitted for Programming Assignments must include JavaDoc block comments for the Class and each Method. Comments for the Class must include the name of the Java Class plus a short description of what the Class does, must also include the Student's name and Student Number. Each Method should describe what the Method does, its parameters, and return values using JavaDoc compatible comments.**

3. **Submission deadline: 11:59pm Monday Dec 2nd. You should be able to complete the work if you have completed the required reading for the assignment. More material is available in the Class Slides. While you have seen most of what is discussed here, some topics discussed in this assignment and needed for your submission may not be seen until another class! You may submit before the deadline, if you so prefer. You may resubmit again later without penalty provided you resubmit before the deadline.**

A. Written Assignment – Submit at Lab Tutorial on Nov 25th – ReSubmit to CourSys by Dec 2nd

Section A does not require any programming. Write your answers neatly in a document such as a Word Document or handwritten on a piece of paper. Keep each answer brief with one or two sentences being sufficient in most cases. Write the answers in your own words.

Your document must include your name, student number, Course Name, Assignment Number, and Date along with your answers to the following questions. Use Courier font and double-space in your write up. Image files of handwritten work are not acceptable except for UML and Data Flow Diagrams.

1. Chapter 17 – Recursion 5 Marks

(a) What is Recursion?

(b) What is Infinite Recursion?

(c) When is a base case needed for Recursive Programming?

(d) Is Recursion necessary?

(e) Given the added overhead of Recursion, when is it a good idea?

2. Chapter 18 – Searching and Sorting 10 Marks

(a) For linear searching: what is the best case, expected case, and worst case search time with respect to N, the number of items to search through?

(b) For logarithmic search: what is the best case, expected case, and worst case search time with respect to N, the number of items to search through?

(c) When would a linear search be preferable to a logarithmic search?

(d) Which searching method requires that the list be sorted?

(e) When would a sequential sort be preferable to a recursive one?

(f) The Selection Sort algorithm sorts using what technique?

(g) The Bubble Sort algorithm sorts using what technique?

(h) Using Big-O notation, how quickly does Bubble Sort run for N items.

(i) The Quick Sort algorithm sorts using what technique?

(j) Using Big-O notation, how quickly does Quick Sort run for N items.

3. Chapter 19 – Trees **5 Marks**

(a) What is a Tree?

(b) What is the root of a Tree?

(c) What is a leaf of a Tree?

(d) Define the height of a tree?

(e) Given a tree with Height N, how many comparison steps does it take to search for a specified element within the tree?

4. UML Diagrams **20 Marks**

Create a UML Diagram of the TicTacToe game from Assignment 3. Your UML diagram must include all class definitions and public methods and public or protected class variables. You do not need to include private methods or members in your diagram. Try to organize your diagram show that the important high level details all fit on the first page. Simplify as required so that it does. If you choose to hand draw your diagram, then you must scan in your diagram into a JPEG file and include the picture in your write-up when you submit to CourSys before the final due date.

Base your diagram on the solution provided on the Class Website for question C5. As a reminder, the following java files each define a Class that must be represented in your UML diagram:

TicTacToe.java	**- The main method for playing the game**
Board.java	**- Records the current state of play on the board**
Move.java	**- Move object used for creating a new move**
Player.java	**- Abstract Class defining Player child class method**
PlayerType.java	**- Defines playerTypeNames and `createPlayer`**
HumanPlayer.java	**- The Human player class**
RandomPlayer.java	**- The Random Computer player class**
SmarterPlayer.java	**- The Smarter Computer player class**

5. Submission Instructions for Part A

You must have these questions answered and ready to hand in at the start of your Lab Tutorial on Nov 25th. They will be marked and handed back to you during the Tutorial.

You must submit your corrected answers to Part A into CourSys by the final due date as a Word or PDF document called A4aWriteUp.doc, A4aWriteUp.docx, or A4aWriteUp.pdf. You will also hand in a hardcopy of Part A (*submitted separately from part B*) into the CMPT125 Assignment box in CSIL by 11:59pm on the final due date as described in Part D.

B. Written Assignment – Submit in CSIL Box and CourSys

Students may discuss the problems with fellow students, but students MUST complete the work on their own and the submitted answers must be your own work.

Section B does not require any programming. Write your answers neatly in a document such as a Word Document that will be submitted to CourSys as part of the assignment. Your document must be called a4bWriteup.doc, a4bWriteup.docx, or a4bWriteup.pdf and must include your name, student number, Course Name, Assignment Number, and Date along with your answers to the following questions. Use Courier font and double-space in your write up. Image files of handwritten work is not acceptable except for the Data Flow Diagram.

1. Data Flow Diagrams 20 Marks

Create a Dataflow Diagram of the TicTacToe game. Your diagram must include all defined classes, and all public methods invocations between classes. Label the arcs which go into a method with the names of the method's formal parameters, and label the arcs represent return values from a method with the class of that return value. Your top-level diagram should fit on one page.

Base your diagram on the solution provided on the Class Website for question C5. As a reminder, the following java files each define a Class that must be represented in your Dataflow diagram:

TicTacToe.java	**- The main method for playing the game**
Board.java	**- Records the current state of play on the board**
Move.java	**- Move object used for creating a new move**
Player.java	**- Abstract Class defining Player child class method**
PlayerType.java	**- Defines playerTypeNames and `createPlayer`**
HumanPlayer.java	**- The Human player class**
RandomPlayer.java	**- The Random Computer player class**
SmarterPlayer.java	**- The Smarter Computer player class**

If you choose to hand draw your diagram, you must scan your diagram into a JPEG file and include the scanned picture in your write-up when you submit to CourSys.

2. Analysis of Algorithms

Using Big-O notation, analyze the following code fragments and specify what their run-time complexity are. In your write-up, label loops and loop bodies that are included in your calculation with the appropriate Big-O notation. Then show your calculations to compute the overall run-time cost using Big-O notation. Your Big-O notation should be specified in terms of N which is the length of the input array `data`**.**

Reminder: Analyzing the run-time cost of an algorithm first requires that you estimate the cost of the method body using Big-O notation, and then multiply that by the number of times the loop (or loops) is executed with respect to the size of the array N.

(a) Sorting Numbers — 10 Marks

One way to sort an array of numbers is to scan through the array looking for pairs of numbers that are in the wrong order and swapping them. Once call pairs have been compared, and possibly swapped, then the entire array is sorted. Using Big-O notation, analyze the following sort algorithm.

```
// Sort data[] array in order from smallest to largest
public static void sort(int[] data)
{
 int N = data.length;
 int position;
 int scan;
 int temp;

 for (position=data.length-1; position>=0; position--)
    for (scan = 0; scan <= position - 1; scan++)
       if (data[scan] > data[scan+1]) {
          /* Elements are in wrong order, swap them */
          temp         = data[scan];
          data[scan]   = data[scan+1];
          data[scan+1] = temp;
       }
    }
```

(b) Searching for a Target in an unsorted Array of Numbers 10 Marks

Searching involves looking for some target in a collection of elements and returning true if the target is found, and false otherwise. When we have an unsorted array of objects, each element must be compared with the target until we find the target element, or until we hit the end of the array.

Examine the following code fragment for searching for a target in an unsorted array of integers. Analyze the algorithm to determine its run-time cost using Big-O notation.

```
public static boolean
   slowSearch(int[] data, int min, int max, int target)
   {
        int N = data.length;
        boolean found = false;

        for(int i=min; !found && i<=max; i++)
            if (data[i] == target)
               found = true;

        return found;
   }
```

(c) Searching for a Target in a Sorted Array of Numbers 10 Marks

When we have a sorted array, we can search the list much faster by using recursion. We start the search by calling `search` with an int[] array called data, and setting min=0 and max=data.length-1.

After comparing our target value to the `int` at the midpoint in the array, we can rule out one half of the array based on whether our target is smaller or larger than the value stored in `data[midpoint]`.

The algorithm performs a recursive step to search for the target in the remaining half of the array. Each recursive step involves searching through a list wit half the number of elements as the previous step.

The recursion stops when the target is found, or there are no further elements to check between min and max. This occurs when min becomes larger than max.

```
public static boolean
   fastSearch(int[] data, int min, int max, int target)
{
 int     N        = data.length;
 boolean found    = false;
 int     midpoint = (min+max)/2; // determine the midpoint

 if (data[midpoint] == target)
    found = true;
 else
    if (data[midpoint] > target)
    {
     if (min <= midpoint - 1)
        // Recursion on the right half of the array
        found = fastSearch(data, min, midpoint-1, target);
    }
    else
     if (midpoint + 1 <= max)
        // Recursion on the left half of the array
        found = fastSearch(data, midpoint+1, max, target);

 return found;
}
```

C. Programming – To be completed by Students Individually

You may *not* use the ArrayList class for this assignment.

1. Implementing Stacks with Linked Lists 30 Marks

Section 13.6 of the Textbook describes how to implement a Stack using Linked Objects. Complete the implementation of the LinkedStack<T> class by implementing each of the methods: push, pop, peek, size, isEmpty, and toString methods.

As a starting point, use the following files provided on the Assignment page for question C1:

```
LinkedStack.java
LinearNode.java
StackADT.java
EmptyCollectionException.java
StudentCodingError.java
```

For this question, you will only modify file LinkedStack.java. The other files are for your convenience and should not be modified.

To get started, create a new project in Eclipse with the name of your choice. Right click on the project and select: *New->Package*. Create a new package called jsjf. This will appear under your `src` directory as its own package sub-directory.

Right click on the `jsjf` package, and select New -> Class and create a new class called `LinkedStack`. Copy the contents of `LinkedStack.java` from the Assignment page to the `LinkedStack.java` in the eclipse editor using copy/paste. Repeat this for the LinearNode Class.

Right click on the `jsjf` package again, and select New -> Interface. Create an Interface called StackADT and copy/paste the contents of file StackADT.java from the assignment page into your class definition.

Create a second package called jsjf.exceptions by right clicking on the project, and select New -> Package and specifying that name. This will create a new package in the Project Explorer window. Right click on that package and create a new class called `EmptyCollectionException`. Copy/Paste the contents of `EmptyCollectionException.java` into that class. Repeat these steps to create class `StudentCodingError`.

You will be implementing your methods in LinkedStack.java. It is the only file that you may change and submit. You may use the implementations for push and pop provided in the text, or write your own. The implementation for the other methods must be your own.

Once you have implemented all methods in your LinkedStack<T> class and there are no compiler errors, you will need to test it using the postfix

Calculator test program. Create two new classes for the following two files:

`PostfixCalculator.java`
`PostfixEvaluator.java`

Run your program, your output should match the example test run below. Note that user input is marked in blue and must be typed in by the student after running the program.

Example test run:

```
Invoking test of LinkedStack implementation
LinkedStack Test Passed! Congratulations!
Running the calculator now...

Enter a valid post-fix expression one token at a time
with a space between each token. (ex: 5 4 + 3 2 1 - + *)

Each postfix expression to evaluate: 5 4 + 3 2 1 - + *
That expression equals 36

Evaluate another expression [Y/N]? n
Exiting calculator
```

2. Basic CodingBat Exercises — 30 Marks

Complete the follow sets of exercises in CodingBat.com. If you have not already done so, register on CodingBat using your SFU email account, and set your preferences for Teacher Share to allow skristja@sfu.ca to view your progress. Your email account will be used to track your progress on CodingBat.

To submit, paste each solution into its own java class file. For example, paste the solution to left2 into file `left2.java`. Create a zip file called a4c2.zip containing all your solutions and submit to CourSys.

Earn a 5-Star badge on CodingBat by completing the following 15 problems from the 5 basic categories: (2 Marks each)

(a) String-1: left2 firstHalf seeColor
(b) Logic-1: cigarParty caughtSpeeding alarmClock
(c) Array-1: firstLast6 rotateLeft3 make2
(d) String-2: doubleChar countCode catDog
(e) Array-2: countEvens sum13 fizzArray

3. Recursion CodingBat Exercises — 30 Marks

Gain practice with writing Recursive methods by completing the following recursive problems from CodingBat. (5 marks each)

(a) Recursion-1: factorial fibonacci bunnyEars bunnyEars2 array11
(b) Recursion-2: groupSum

4. Sorting 20 Marks

Write an application that reads a single line of input from the user containing at least one integer and possibly many more. You may reuse and adapt your input validation loop from Assignment 3 for this if you chose. Use your LinkedStack data structure to push all the integers onto the stack as your scan them in using your scanner input parsing loop. If any of the input values are non-ints, reject all the input and re-prompt the user to enter one or more integers.

***Your program should count the number of ints that it pushed onto the stack. Once your program has successfully stacked all the valid int values, use this count* to determine how many were entered by the user. Allocate an int[] array of that size, and fill the array with the int values from the stack. Recall that stacks return data elements in a last-in-first-out order so the first int retrieved from the stack.pop() method call should be stored in the last element in the array. The last item to be removed from the stack should be stored in the array element with index value of zero.**

Your program must store the integers in an int[] array of the appropriate length and then call your sort routine to sort them. Write a public method within your class called myIntSort **with the following signature:**

```
public static void myIntSort(int[] nums) {
}
```

Method myIntSort **modifies the nums array such that integers are sorted from smallest to largest. So {1, 10, -10, 2} yields {-10, 1, 2, 10}. You may choose to implement any of the following sort routines: Insertion Sort, Selection Sort, Bubble Sort, QuickSort, or MergeSort. The routines are described in the text in Chapter 18. You must write the code yourself however, you may use the text examples as a guide. Add comments. There is a 5 mark bonus for implementing either QuickSort or MergeSort.**

Main should display a count of how many integers where entered, then display the integers in the original order, followed by displaying them in sorted order from smallest to largest.

Your Java Class file should be called `MyIntSort.java`.

Example test run:

```
Enter one or more integers: 23 10 -5 Fred 99 1 0 3
Sorry but Fred is not an int! Please try again!
Enter one or more integers: 23 10 -5 123  99 1 0 3
Number of integers: 8
myStack: [3 0 1 99 123 -5 10 23]
myArray: [23 10 -5 123 99 1 0 3]
Sorted : [-5 0 1 3 10 23 99 123]
```

D. Submission – To be completed by Students Individually

Due date: Monday Dec 2nd 11:59pm (Last Day of Class)

Students are responsible for submitting the requested work files by the stated deadline for full marks. It is the student's responsibility to submit on time. Submissions via email will *NOT* be accepted.

You must submit your final version of the following files before the deadline. All written answers for Parts A and Parts B must be included in separate Word or PDF documents which are double spaced and include the Student's Name and Student Number at the top, along with the Course Number and Assignment Number. Students must ensure that all submitted code compiles and is properly commented and formatted for readability.

Submit Separate Hardcopies for Parts A and for B into the CMPT125 Assignment Box in CSIL:

Part A : a4aWriteup
Part B : a4bWriteup

Submit into CourSys:

Part A : a4aWriteup.doc or a4aWriteup.pdf
Part B : a4bWriteup.doc or a4bWriteup.pdf
Question C1 : LinkedStack.java
Question C2 : a4c2.zip
Question C3 : a4c3.zip
Question C4 : MyIntSort.java

Files are to be submitted into CourSys under assignment A4 as individuals. No group name is required and none should one be used. See the course website for submission instructions at:

http://blogs.sfu.ca/courses/fall2013/cmpt125/labs/submitting/