~~March 12 2018 9:20 AM~~ **March 14 9:47 AM**

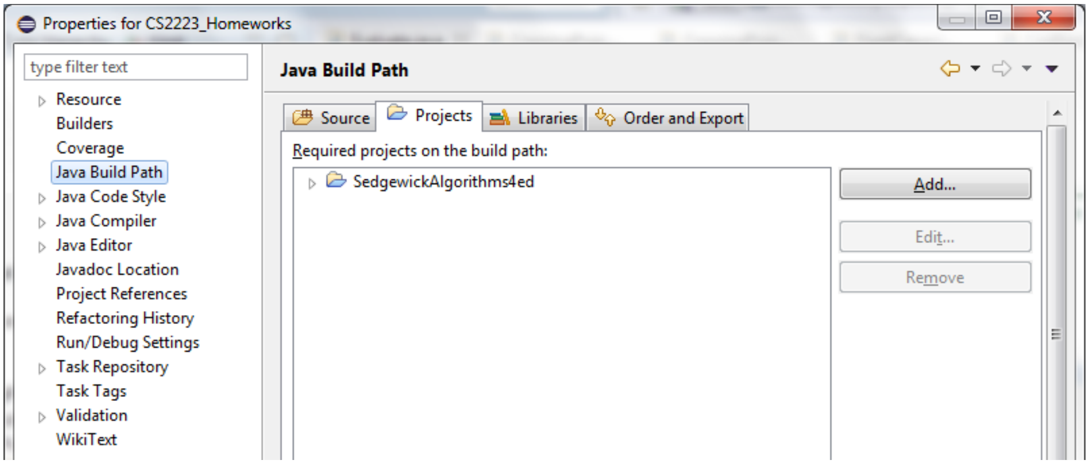
CS 2223 D18 Term. Homework 1 (100 pts.)

Homework Instructions

- This homework is to be completed individually. If you have any questions as to what constitutes improper behavior, review the examples I have posted online http://web.cs.wpi.edu/~heineman/html/teaching_/cs2223/d18/#policies .
- Due Date for this assignment is 2PM Friday March 23rd. Homeworks received after 2PM receive a 25% late penalty. Homeworks received after 6PM will receive zero credit.
- Submit your assignments electronically using the canvas site for CS2223. Login to canvas.wpi.edu and locate HW1. You must submit a single ZIP file that contains all of your code as well as the written answers to the assignment.
- All of your Java classes must be defined in a packager USERID where USERID is your CCC user id.
- Submission information is found at the end of this document.

First Steps

Your first task is to copy all of the files from the Git repository that you will be modifying/using for homework1. First, make sure you have created a Java Project within your workspace. Be sure to modify the build path so you will have access to the shared code I provide. To do this, select this project and right-click to bring up the Properties for the project. Choose the option **Java Build Path** on the left and click the Projects tab. Now **Add…** the SedgewickAlgorithms4ed project to your build path.



Properties for CS2223_Homeworks

Java Build Path

Required projects on the build path:

SedgewickAlgorithms4ed

Once done, create the package **`USERID.hw1`** inside this project which is where you will complete your work. Start by copying the source files from **`algs.hw1`** (**`Evaluate`**, **`FixedCapacityStackOfInts`**, **`TowerOfHanoi`**, **`TwiceSorted_Solution`**) and paste them into your **`USERID.hw1`** package[1].

In this way, I can provide sample code for you to easily modify.

[1] Do not copy class **`TwiceSorted`** into your local project.

~~March 12 2018 9:20 AM~~ **March 14 9:47 AM**

Stack Experiments (35 pts.)

On page 129 of the book there is an implementation of a rudimentary calculator using two stacks for expression evaluation. I have created the `Evaluate` class which you have copied into your **`USERID.hw1`** package. Note that all input (as described in the book) must have spaces that cleanly separate all operators and values.

1. **(5 pts.)** Run this program on input "`6 + 8`" and explain the observed output.
2. **(5 pts.)** Run this program on input "`-99`" and explain the observed output.
3. **(5 pts.)** Run this program on input "`- 99`" and explain the observed output (there is a space between the minus sign and the 9).
4. **(5 pts.)** Run this program on input "`( ( 2 / 3 ) + ( 3 * 7 ) )`" and explain the observed output.
5. **(5 pts.)** Run this program on input "`( 2 + ( 3 + ) 4 )`" and explain the observed output.
6. **(5 pts.)** Modify `Evaluate` to support two new operations
    a. Add a new exponent operation "^".
    b. Add a new floor operation "floor" which computes the largest integer less than or equal to x.
7. **(5 pts.)** Once done, run your program on the input "`( floor ( 5 ^ 0.5 ) )`" and explain the observed output.

Write the answers to these questions in the "WrittenQuestions.txt" text file. For question 5, modify your copy of the **`Evaluate`** class. **For each question, describe the state of the two stacks when the program terminates.**

Tower Of Hanoi Stack Exercise (20 pts.)

Modify the **`TowerOfHanoi`** class that you copied into your **`USERID.hw1`** package. Instructions for playing Tower of Hanoi can be found at https://en.wikipedia.org/wiki/Tower_of_Hanoi. In this program, there are four disks to be moved. The first move must either be to move a disk from stack 1 to 2 ("1 2") or from stack 1 to 3 ("1 3"). **Appendix A1** contains a sample transcript of a winning game.

1. **(10 pts.)** Modify **`outputState()`** to properly show the state of each stack from left to right, where the larger disk appears to the left of smaller disks. Thus the initial state must appear as:

```
Stack1: 4321
Stack2:
Stack3:
```

2. **(10 pts.)** Modify **`move(int from, int to)`** to move the topmost disk from stack 'from' to stack 'to' if it is a valid move. If the move is invalid, then return false and do not change the state.

~~March 12 2018 9:20 AM~~ **March 14 9:47 AM**

TwiceSorted Programming Exercise (25 pts.)

A *TwiceSorted* two dimensional (2D) square array of **`int`** values has the following characteristics. Each row contains values sorted from left to right in increasing order; in addition, each column contains values sorted from top to bottom in increasing order. The number of columns is the same as the number of rows, and there are at least 2 rows (and columns). Finally, all values in the array are distinct. A sample array appears below.

	0	**1**	**2**	**3**
0	5	12	18	22
1	7	24	37	50
2	26	27	38	57
3	29	33	60	62

The goal is to write a method that searches for a specific target value in this 2D array that uses the fewest number of array[r][c] lookups. Naturally, the search must also be functionally correct. As you will see, your **`locate(int target)`** method – see below – will be called 512 times, to look for integer values from 0 to 511. Once you have copied the default **`TwiceSorted_Solution`** class into your **`USERID.hw1`** package, execute it, and you will see the following Exception :

```
Exception in thread "main" java.lang.IllegalStateException: Only found 1 values.
	at USERID.hw1.TwiceSorted.trial(TwiceSorted.java:159)
	at USERID.hw1.TwiceSorted_Solution.main(TwiceSorted_Solution.java:36)
```

After you have properly modified the class so it works properly, executing your class will result in a message that states the total number of array inspections used.

```
Number of inspections:30752
```

For this question, you are to modify the **`TwiceSorted_Solution`** Java class which you have copied into your **`USERID.hw1`** package. In particular, edit the **`locate(int target)`** method so it works properly.

To access a value in the *TwiceSorted* array, you must call the inherited method **`inspect(r,c)`** which retrieves the **`int`** value found in the **r**th row and **c**th column. Both **r** and **c** are integers in the range from 0 up to but not equal to **`length()`**. Note that **`length()`** is an inherited method that you can use to return the number of rows (or columns) in the array.

The existing code will run using a pre-defined *TwiceSorted* array that you can find as the **`big`** field in the **`algs.hw1.TwiceSorted`** class. Do not modify this array. If you want to debug your code using a smaller array as an example, then (temporarily) modify the main method of your **`TwiceSorted_Solution`** class.

```
System.out.println("Number of inspections:" + new TwiceSorted_Solution(sample).trial(512));
```

~~March 12 2018 9:20 AM~~ **March 14 9:47 AM**

1. **(15 pts.)** Modify the **TwiceSorted_Solution** class so it is functionally correct. That is, for values known to be in the array, it returns the location as an array of two integers [r, c]. For values not in the array, it returns **null**.

```
package USERID.hw1;
public class TwiceSorted_Solution {
  ...
  public int[] locate(int target) {
    // Only look at the value in the upper right corner of the TwiceSorted array.
    if (inspect(0,length()-1) == target) {
      return new int[] {0, length()-1};
    }
    return null;
  }
}
```

As you can see in this sample code, your locate method can call **inspect(r,c)** for any proper **r** and **c** value. You can also find the number of rows (or columns) by calling **length()**.

Your code must be functionally correct and for these 10 points, it doesn't matter how many array inspections are reported.

2. **(10 pts.)** To receive an additional 10 points, you must implement an alternative implementation that requires fewer array inspections while still remaining functionally correct.   Note: you should be able to reduce the number of array inspections by 50% when compared against a straightforward search that doesn't take advantage of the structure of a TwiceSorted array.

Hint: Can Binary Array Search come to your rescue?

BONUS QUESTION (don't attempt until everything is done)

3. **(1 pts.)** Can you discover an even more efficient implementation that requires even fewer array inspections? I have a solution that needs only **4,267** inspections for the default trial using the default **big** TwiceSorted array. Can you do as good (or better)?

Binary Array Search Exercise (20 pts.)

You have a 2D array that consists of M rows (where M>2) and N columns (where N=2^k for integer k>0). You know that each row in the array contains different values sorted from left to right. You also know that there exists one special value that appears M times, once on each row. Below is an example array that fits this description, where **12** is the special value, M=6 and k=2

5	12	18	22
2	10	12	70
1	3	9	12
12	17	24	76
8	11	12	19
7	12	49	51

Note: the only value that is ever duplicated in the entire 2D array is the special value, which appears M times, once on each row.

~~March 12 2018 9:20 AM~~ **March 14 9:47 AM**

You are to write an algorithm that determines (a) the special value; and (b) the column number where the special value appears in each row. For example, given the above array as input, the output would be:

```
Special value is 12
Located: (0,1) (1,2) (2,3) (3,0) (4,2) (5,1)
```

You can choose to implement the algorithm or describe it using pseudocode, much as you have seen me do in lecture. **Note: this would have been a good exam question but it requires just a bit more work than I would like in an exam.**

1. **(10 pts.)** Describe your algorithm for locating special and the location in the array (using zero-based indexing).

2. **(10 pts.)** In the worst case, count the maximum number of **<u>array inspections</u>** needed for this task.

BONUS QUESTION (don't attempt until everything is done)

3. **(1 pts.)** Provide a sample 6x4 array that forces the worst case (in terms of # of array inspections) for your algorithm

4. **(1 pts.)** Provide a sample 6x4 array that gives you the best case behavior, and compute the minimum number of array inspections needed for this best case.

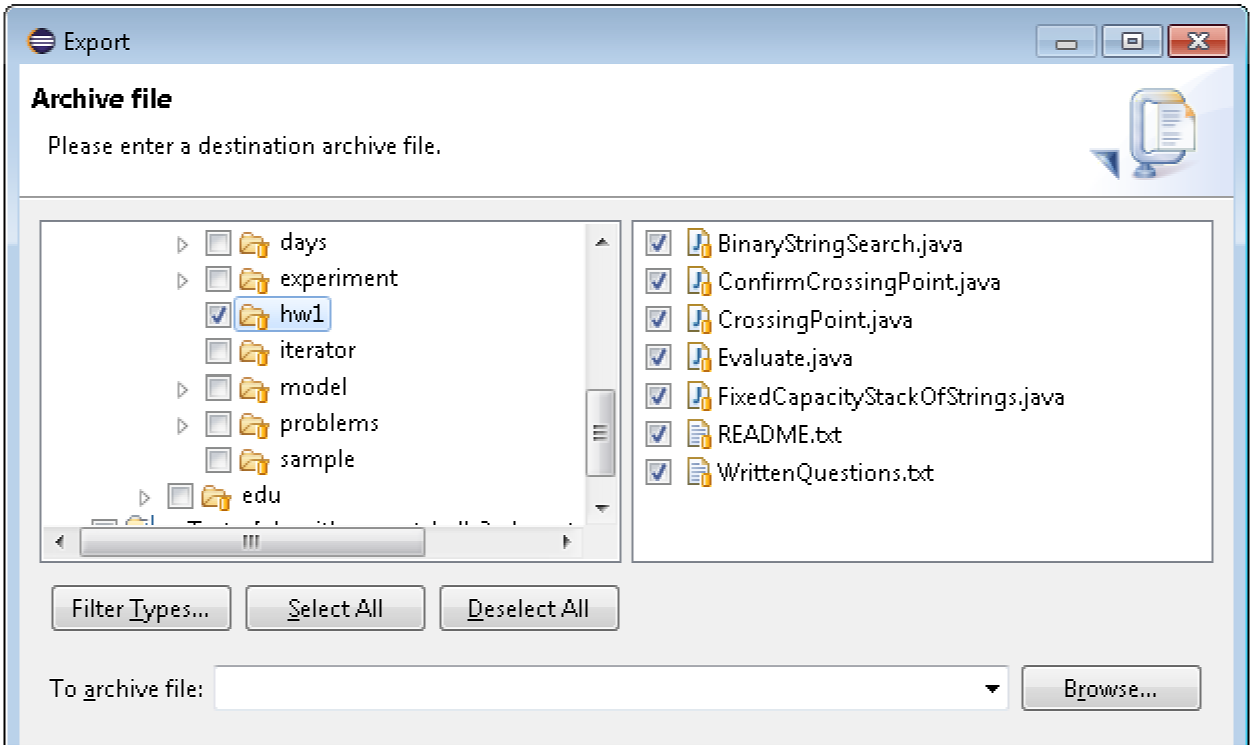
5. **(1 pts.)** To determine average case, you would need to write a program and then try all possible 6x4 arrays. Total number of arrays is 4^6 = 4096. Then you would need to compute the total number of array inspections for each of these inputs, and then average the results to determine the average number of array inspections needed.

~~March 12 2018 9:20 AM~~ **March 14 9:47 AM**

Submission Details

Each student is to submit a single ZIP file that will contain the implementations. In addition, there is a file “WrittenQuestions.txt” in which you are to complete the short answer problems on the homework.

The best way to prepare your ZIP file is to export your entire `USERID.hw1` package to a ZIP file using Eclipse. Select your package and then choose menu item “**Export…**” which will bring up the Export wizard. Expand the **General** folder and select **Archive File** then click **Next**.



☐ days
☐ experiment
☑ hw1
☐ iterator
☐ model
☐ problems
☐ sample
☐ edu

☑ BinaryStringSearch.java
☑ ConfirmCrossingPoint.java
☑ CrossingPoint.java
☑ Evaluate.java
☑ FixedCapacityStackOfStrings.java
☑ README.txt
☑ WrittenQuestions.txt

To archive file:

You will see something like the above. Make sure that the entire “hw1” package is selected and all of the files within it will also be selected. Then click on **Browse…** to place the exported file on disk and call it USERID-HW1.zip or something like that. Then you will submit this single zip file in my.wpi.edu as your homework1 submission.

Addendum

If you discover anything materially wrong with these questions, be sure to contact the professor or TA/SAs posting to the discussion forum for HW1 on piazza; you may also email cs2223h-staff@cs.wpi.edu.

When I make changes to the questions, I enter my changes in red colored text as shown here.

~~March 12 2018 9:20 AM~~ **March 14 9:47 AM**

Appendix A1: Sample Output For Tower Of Hanoi Completed Program

```
Stack1: 4321
Stack2:
Stack3:

Enter two disk numbers A B to move top disk on A to B. You win when all disks are on Stack 3.
1 2
Moving top disk from stack 1 to stack 2

Stack1: 432
Stack2: 1
Stack3:

Enter two disk numbers A B to move top disk on A to B. You win when all disks are on Stack 3.
1 3
Moving top disk from stack 1 to stack 3

Stack1: 43
Stack2: 1
Stack3: 2

Enter two disk numbers A B to move top disk on A to B. You win when all disks are on Stack 3.
2 3
Moving top disk from stack 2 to stack 3

Stack1: 43
Stack2:
Stack3: 21

Enter two disk numbers A B to move top disk on A to B. You win when all disks are on Stack 3.
1 2
Moving top disk from stack 1 to stack 2

Stack1: 4
Stack2: 3
Stack3: 21

Enter two disk numbers A B to move top disk on A to B. You win when all disks are on Stack 3.
3 1
Moving top disk from stack 3 to stack 1

Stack1: 41
```

~~March 12 2018 9:20 AM~~ **March 14 9:47 AM**

```
Stack2: 3
Stack3: 2

Enter two disk numbers A B to move top disk on A to B. You win when all disks are on Stack 3.
3 2
Moving top disk from stack 3 to stack 2

Stack1: 41
Stack2: 32
Stack3:

Enter two disk numbers A B to move top disk on A to B. You win when all disks are on Stack 3.
1 2
Moving top disk from stack 1 to stack 2

Stack1: 4
Stack2: 321
Stack3:

Enter two disk numbers A B to move top disk on A to B. You win when all disks are on Stack 3.
1 3
Moving top disk from stack 1 to stack 3

Stack1:
Stack2: 321
Stack3: 4

Enter two disk numbers A B to move top disk on A to B. You win when all disks are on Stack 3.
2 3
Moving top disk from stack 2 to stack 3

Stack1:
Stack2: 32
Stack3: 41

Enter two disk numbers A B to move top disk on A to B. You win when all disks are on Stack 3.
2 1
Moving top disk from stack 2 to stack 1

Stack1: 2
Stack2: 3
Stack3: 41

Enter two disk numbers A B to move top disk on A to B. You win when all disks are on Stack 3.
3 1
Moving top disk from stack 3 to stack 1

Stack1: 21
```

~~March 12 2018 9:20 AM~~ **March 14 9:47 AM**

```
Stack2: 3
Stack3: 4

Enter two disk numbers A B to move top disk on A to B. You win when all disks are on Stack 3.
2 3
Moving top disk from stack 2 to stack 3

Stack1: 21
Stack2:
Stack3: 43

Enter two disk numbers A B to move top disk on A to B. You win when all disks are on Stack 3.
1 2
Moving top disk from stack 1 to stack 2

Stack1: 2
Stack2: 1
Stack3: 43

Enter two disk numbers A B to move top disk on A to B. You win when all disks are on Stack 3.
1 3
Moving top disk from stack 1 to stack 3

Stack1:
Stack2: 1
Stack3: 432

Enter two disk numbers A B to move top disk on A to B. You win when all disks are on Stack 3.
2 3
Moving top disk from stack 2 to stack 3

Congratulations! You completed the puzzle in 15 moves.
```