

CSE 8B: Introduction to Programming and Computational Problem Solving - 2

Assignment 4

Loops, Recursion, and Arrays

Due Date: Wednesday, May 3, 11:59 PM

Learning goals:

- Getting practice with loops
- Getting practice with 1D arrays and 2D arrays
- Getting practice with recursion

NOTE: This programming assignment must be done individually.

Your grade will be determined by your most recent submission. If you submit to Gradescope after the deadline, it will be marked late and the late penalty will apply regardless of whether you had past submissions before the deadline.

If your code does not compile on Gradescope, you will receive an automatic zero on the assignment.

Coding Style (10 points)

For this programming assignment, we will be enforcing the [CSE 8B Coding Style Guidelines](#). These guidelines can also be found on Canvas. Please ensure to have COMPLETE file headers, class headers, and method headers, to use descriptive variable names and proper indentation, and to avoid using magic numbers.

Part 0: Getting started with the starter code (0 points)

1. If using a personal computer, then ensure your Java software development environment does not have any issues. If there are any issues, then review Assignment 1, or come to the office/lab hours before you start Assignment 4.
2. First, navigate to the cse8b folder that you have created in Assignment 1 and create a new folder titled `assignment4`

3. Download the starter code. You can download the starter code from Piazza → Resources → Homework → `assignment4.zip`. The starter code should contain two files: `FunWithArrays.java`, and `RecursiveHourglass.java`. Place the starter code within the `assignment4` folder that you have just created

Part 1: Implement Methods in FunWithArrays (70 points)

In class `FunWithArrays` of the starter code, 9 methods are already declared for you: `findRange`, `findAvg`, `arrayCopy`, `containsIntegerSequence`, `reverseArray`, `differentPairs`, `findLargestSubarraySum`, `unitTests`, and `main`. **For this part of the assignment, your task is to implement the first seven methods** (`findRange`, `findAvg`, `arrayCopy`, `containsIntegerSequence`, `reverseArray`, `differentPairs`, and `findLargestSubarraySum`).

`findRange` (10 points):

This method takes an integer array variable, `array`, as the input. You must find the range of the integers within this array where the range is defined as the minimum integer subtracted from the maximum integer. If `array` is `null` or if the length is `0`, you must return `0`.

Sample:

```
findRange({1, 2, 3, 4, 5}) → 4
```

Explanation: This is because the maximum integer is 5 and the minimum integer is 1. $5 - 1 = 4$.

`findAvg` (10 points):

This method takes an integer array variable, `array`, as the input. You must find the average of the integers within this array and return the result as a `double`. Average is defined as the sum of all integers divided by the number of integers. If `array` is `null` or if the length is `0`, you must return `0`.

Sample:

```
findAvg({1, 2, 3, 4, 5}) → 3.0
```

Explanation: The sum of all integers is 15. The number of integers in the array is 5. $15/5 = 3.0$.

`arrayCopy` (10 points):

This method takes an integer array variable, `array`, as the input. You must return a **deep** copy of the array with the same contents. A deep copy of the array is defined as an array that has the same contents but is not the same array that is stored in memory. So `arr == arrCopy` should be `false`. If you just returned the same reference that was passed in as an argument, you will fail the autograder test cases. If `array` is `null`, you must return `null`. You are **not** allowed to use `System.arraycopy()`.

Sample:

```
arrayCopy({1, 2, 3, 4, 5}) → {1, 2, 3, 4, 5}
```

`containsIntegerSequence` (10 points):

This method takes an integer array variable, `array`, as the input and must return a `boolean`. It must return `true` if the contents of the array has exactly the integers from $[0 \dots n-1]$ where n is the length of the array. Examples below will make this clearer. If `array` is `null`, you must return `false`. If the length is 0 , you must return `true`.

Sample:

```
containsIntegerSequence({0, 1, 3}) → false
```

```
containsIntegerSequence({3, 4, 1, 0, 2}) → true
```

Explanation: The first array is of length 3. Hence, the contents that should only exist in this array are 0, 1, and 2. However, it contained 3 and not 2. Hence we return false. The second array is of length 5. Hence, it should contain only the integers 0, 1, 2, 3, and 4. This is exactly what it contains, so we return true.

`reverseArray` (10 points):

This method takes an integer array variable, `array`, as the input and does **not return** anything. Instead, it must reverse the array that was passed in as an argument in place. If `array` is `null`, you do not have to do anything and must simply return from the method.

Sample:

`reverseArray({1, 2, 3, 4, 5})` → the same array that was passed in as an argument should now be: `{5, 4, 3, 2, 1}`

Be sure that your method modifies the array that was passed in. Making a copy of the array, reversing the copy, then assigning the variable that was passed in to the reference of the copy would fail the autograder.

`differentPairs` (10 points):

This method takes in two integer 2D array variables, `arr1`, and `arr2` as the input and returns an 2D array where every row of the 2D array is of length 2 that denotes which pairs (i, j) within `arr1` and `arr2` are different. If all integers in both arrays are the same, then return an empty 2D array. Each row (pair) in the resulting 2D array should be sorted from least to greatest, prioritizing the row index first. If the row indexes are the same, then the pair with the smaller column index should go first in the array. Examples below will make this clearer.

Sample:

`differentPairs({{-1, 0, 3}, {5, 6, 7}, {1, 2, 3}}, {{-1, 0, 3}, {2, 6, 7}, {-1, 2, 3}})` → `{{1, 0}, {2, 0}}`

-1	0	3
5	6	7
1	2	3

-1	0	3
2	6	7
-1	2	3

`differentPairs({{1, 2, 3}}, {{1, 2, 3}})` → `{}`

1	2	3
---	---	---

1	2	3
---	---	---

Explanation: For the first example, note that arr1 and arr2 differ at the second row (index 1), first column (index 0). It also differs at the third row (index 2), first column (index 0). Hence the resulting array that is returned should be `{{1, 0}, {2, 0}}`. For the second example, the two arrays are the same in contents so we return an empty 2D array.

Definition on sorting:

To clarify on the order of the pairs that you result should have, take a look at the following.

Imagine if two arrays differed at these positions:

(5, 6)

(5, 4)

(1, 2)

(6,0)

(0, 3)

Then your resulting array should look like: `{{0, 3}, {1, 2}, {5, 4}, {5, 6}, {6,0}}`. We prioritize the row index first when sorting from least to greatest. Only when they are the same, do we look at the column index for judgment. Note that you should not be using any built in Java sorting methods and you do not even need to implement any code for sorting. Traversing the arrays in a straightforward way will yield the correct answer.

findLargestSubarraySum (10 points)

This method takes in one integer 2D array variable, `arr` as the input and returns an integer that is the maximum sum of a 2D subarray within this arr. A 2D subarray is defined as any non-empty rectangle/square embedded within arr (including the entire arr) itself.

If `arr` is `null`, the number of rows is `0`, or the number of columns is `0`, you should return `0`. Examples below will make this clearer:

Samples:

-10	-20	1
-100	900	-2
1000	1	3

`findLargestSubarraySum({{-10, -20, 1}, {-100, 900, -2}, {1000, 1, 3}}) → 1802`

1	1	1
1	1	1
1	1	1

`findLargestSubarraySum({{1, 1, 1}, {1, 1, 1}, {1, 1, 1}}) → 9`

-1	-20	-20
-5	10000	-1
-1	-1	-1

`findLargestSubarraySum({{-1, -20, -20}, {-5, 10000, -1}, {-1, -1, -1}}) → 10000`

Explanation:

The first example, the largest sum one can obtain in any subarray from the original array, is the rectangle highlighted in yellow. One can work out by hand to verify. $-100 + 900 - 2 + 1000 + 1 + 3 = 1802$.

In the second example, from the definition of a subarray, it can also be the entire original array itself and since the array contains only positive integers, the correct answer is the sum of all the integers in the 2D array.

A single element (square) is itself classified as a subarray and hence the answer in this case is 10000. One can verify that adding other elements would only decrease the answer.

Part 2: Implement RecursiveHourglass (10 points)

Remember the hourglass that you generated in Assignment 2? Well, this part of the assignment is extremely similar, except that you will be generating the hourglass recursively.

Please take a look at the helper method provided called `getLine` that you **must** use in your implementation below. This method takes in three parameters, a character `c`, and integer `n`, and another integer `numSpaces`. `getLine` returns a `String` of `n` characters (`c`) with `numSpaces` spaces in the front and back of the line, along with a newline character at the end. Examples will make this clear:

```
getLine('@', 3, 0) → "@@@\n"
getLine('*', 5, 1) → " ***** \n"
getLine('#', 1, 2) → "  #  \n"
```

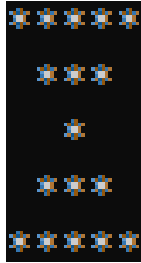
You must NOT change this method and you MUST use it in your implementation below. Though however you use it is up to you.

`recursiveHourglass` (10 points):

This method takes in the three variables as parameters `c`, `height`, and `numSpaces`. This method must generate the exact same structure of the hourglass in Assignment 2 based on the parameters that are passed in. More concretely this method must generate an hourglass that is made up of the char `c` with `height` rows. The first row starts with `n` `cs` and decreases by two each row (one at the front and one at the end). However, the `cs` that were deleted must now be replaced by space characters. When you reach the middle of a single `c`, the hourglass starts to expand again with additional `cs` placed at the beginning and back (with the corresponding

spaces deleted). If `height` is less than or equal to zero or if `height` is not an odd number, you must return the empty string (`""`).

For example, if we create an hourglass of height **five** with `c` as `'*'`, then the result string should be: `*****\n *** \n * \n *** \n*****\n` (where `\n` is the new line character). When this string is printed out, it should look like this:



NOTE: Notice how each row has one extra space in the **beginning and end** until you reach the middle of the hourglass which is just a single `'*'` which then starts reversing. In this example, there are 5 rows.

IMPORTANT (Will lose points if not followed):

1. There must be the right amount of spaces **before and after** each row. More importantly, a common mistake is if you forget the spaces **after** the asterisks and print out the string, it may look correct in your terminal but will fail the autograder.
2. Append the newline character `'\n'` after the last space (if any) in every row **including the last row**. Do not append anything else. Any extra characters will fail the autograder.
3. You must implement this method recursively (which means calling `recursiveHourglass` within the `recursiveHourglass` method). We will be testing to see if your answer is correct and if you actually made recursive calls. Hence if you use loops in any way to implement your method, your grade for this portion of the assignment will result in a 0.
4. You must not implement or use any other helper methods to solve this problem and must only rely on `getLine()` and `recursiveHourglass()`.

Part 3: Compile, Run, and UnitTest Your Code (10 points)

In this part of the assignment, you need to implement your own test cases in the method called `unitTests` for both files (`FunWithArrays.java` and `RecursiveHourglass.java`)

In the starter code, several test cases are already implemented for you. You can regard it as an example to implement other cases. The general approach is to come up with different inputs and manually give the expected output, then call the method with that input and compare the result with expected output.


You are encouraged to create as many test cases as you think to be necessary to cover all the edge cases. The `unitTests` method should `return true` only when all the test cases are passed. Otherwise, it should `return false`. **To get full credit for this section, for each of your eight methods above, you should have at least 3 total test cases that cover different situations (including the ones we have provided).** In other words, you will need to create two more tests for `findRange`, two more tests for `findAvg`, two more tests for `arrayCopy`, two more tests for `containsIntegerSequence`, two more tests for `reverseArray`, two more tests for `differentPairs`, two more tests for `findLargestSubArraySum`, and two more tests for `recursiveHourglass`.

Submission

You're almost there! Please follow the instructions below carefully and use the **exact submission format**. Because we will use scripts to grade, **you may receive a zero** if you do not follow the same submission format.

1. Open Gradescope and login. Then, select this course → Assignment 4.
2. Click the DRAG & DROP section and directly select the required files `FunWithArrays.java`, and `RecursiveHourglass.java`. Drag & drop is fine. Please make sure you do not submit a zip, just the two files in one Gradescope submission. Make sure the names of the files are correct.
3. You can resubmit unlimited times before the due date. Your score will depend on your final (most recent) submission, even if your former submissions have higher scores.
4. Your submission should look like the below screenshot. If you have any questions, feel free to post on [Piazza](#)!

Submit Programming Assignment

 Upload all files for your submission

Submission Method

 Upload  GitHub  Bitbucket

Add files via Drag & Drop or [Browse Files](#).

Name	Size	Progress	✕
FunWithArrays.java	7.1 KB	<div style="width: 100%;"></div>	✕
RecursiveHouglass.java	2 KB	<div style="width: 100%;"></div>	✕

Submitting For

Darren Yeung

Cancel

Upload