CSCI 111 - COMPUTER SCIENCE I
FALL 2015, PROJECT 6
ASSIGNED: NOV 9TH, DUE: 11:59PM ON NOV 18TH

Instruction

In this project, you will merge two arrays into one array that is in **ascending order**. **You may assume that both individual arrays are in ascending order**. Consider the 2 following examples.

Example 1: *first* and *second* arrays are in **ascending** order

first Array

2	26	39	44	58	59	72

second Array

3	4	5	27	42	43	60	61	62

merged Array

2	3	4	5	26	27	39	42	43	44	58	59	60	61	62	72

Example 2: *first* and *second* arrays are in **ascending** order

first Array

12	14	16	18	20	22	24

second Array

2	4	6	8	10

merged Array

2	4	6	8	10	12	14	16	18	20	22	24

Algorithm

Download the 6 input files (*input1_1.txt*, *input1_2.txt, input2_1.txt, input2_2.txt*) from the Blackboard. The file, inputX_1.txt contains the array element for ***first*** array and the file, inputX_2.txt contains the array element for ***second*** array. You are going to write a program according to the followings:

1. Ask user to enter input file names and read data from the input files and populate the *first* array and the *second* array. The file indexed as inputX_1.txt must populate *first* array and the file indexed as inputX_2.txt must populate the *second* array. You may need to maintain variables to show how many elements are in each of the arrays. For simplicity, declare all arrays with **size 20.**

2. Traverse both arrays using a **while** loop until either one of the arrays has been entirely traversed (i.e., consider the count variables of each of the arrays - ***countFirst*** & ***countSecond*** - these act as the length of each of the arrays). In the above examples, Example 1 will completely traverse *second* array and only partially traverse *first* array. In example 2, *first* array will be completely traversed and *second* array will only partially be traversed. You must consider the following:

    a. Use three variables to traverse the indexes in each of the arrays-these will keep track of the current index of the current element to use/populate in each. ***indexFirst*** will keep track of the current index in ***first***, ***indexSecond*** the current index in ***second***, and ***indexMerge*** the current index in ***merge***.

    b. Within this while-loop, compare the current elements of first and second. If ***first's*** element is smaller, then assign it to ***merge*** and increment the appropriate counters. If ***second***'s element is smaller, then assign it to ***merge*** and increment the appropriate counters.

    c. Note that as long as ***indexFirst*** is less than ***countFirst*** and ***indexSecond*** is less than ***countSecond*** you need to keep traversing

3. Once one of the two arrays is completely traversed, you must still finish traversing the array that was only partially traversed. If the remaining array is ***first***, populate the ***merge*** array with the remaining elements in ***first***. If the remaining array is ***second***, populate the ***merge*** array with the remaining elements in ***second***.

4. Traverse and output the ***merge*** array.

5. (**Extra credit 20 points**) Modify your program to generate *merged* array with unordered arrays. You still need to generate ordered ***merged*** array as in the Example 3. Use *input3_1.txt* and *input3_2.txt* for input data. First, sort the *first* array and *second* array and reuse the algorithm described in step 2, 3, and 4. **DO NOT** use built-in sort function or method.

    Example 3 (For extra credit): *first* and *second* arrays are in **random** order

    first Array (unordered)

8	3	13	5	10

    second Array (unordered)

9	6	11	12	4	7	2

    Merged Array (ordered)

2	3	4	5	6	7	8	9	10	11	12	13

Requirements

1. Name your program with MergedArrays and your account ID. For example if your account id is "kchoo" then use **MergedArrays_kchoo** for your class name and **MergedArrays_kchoo.java**.

2. Include header comments formatted as show below.

```
// CSCI 111, Section X
// Jane Doe
// Student ID 12345678
// Project 6
```

```
//
// Honor Code Statement: In keeping with the honor code policies of the
// University of Mississippi, the School of Engineering, and the Department
// of Computer and Information Science, I affirm that I have neither given
// nor received assistance on this programming assignment. This assignment
// represents my individual, original effort.
//
// Put a program description here
```

3. You MUST use multiple **while-loop** for traversing arrays.

4. You MUST follow the algorithm specified in the description.

5. You **MUST NOT** use the sort method for array sorting for extra credit.

Submitting your work: submit your program on Blackboard. As stated in the syllabus, 10 points will be deducted per day for late submission. Program will be accepted by 4 days after the due day. Late submission can also be made to the Blackboard.

Grading criteria

Point Deduction	Project Requirement
-10 / day late	Submit source code (i.e. ".java") file to Blackboard on time.
-70	Program must compile.
-5	Follow alignment, indentation and spacing conventions.
-5	Output must be formatted as specified by the project description or in a meaningful way. For example, provide labels for all outputs.
will vary	Logic Errors. (incorrect output)
-5	Do not use a break (unless within a switch) and/or a continue statement. (reference page 233 of text)
will vary	Run-time Errors. Program throws and unhandled exception.
-5 / items	Follow the requirements specified in the programming assignment