Programming Assignment #1

Programming assignments are to be done individually. You may discuss the problem and general concepts with other students, but there should be no sharing of code. You may not submit code other than that which you write yourself or is provided with the assignment. This restriction specifically prohibits downdensitying code from the Internet. *If any code you submit is in violation of this policy, you will receive **no** credit for the entire assignment.*

This project has the following goal:

- Ensure that you understand the structure of divide-and-conquer algorithms and can navigate implementing them.
- Explore trade-offs to when an efficient algorithm design is not always the best solution.

We have provided Java code skeletons that you should fill in with the algorithmic logic for each of the two required solutions. If you change `Driver.java`, those changes will not carry over as we will replace `Driver.java` with our own version.

Problem Statement. You've landed a summer internship for CapMetro (Woot!). You've been asked to study the ebb and flow of passengers on a particular bus route. A single bus route can be represented as a series of bus stops. CapMetro keeps track of how many people get on and off at each stop, and you've generated an average value for each stop, which may be a positive or negative real number (positive numbers indicate an increase in the number of passengers right after a stop; negative numbers indicate an overall decrease in passengers). CapMetro wants to optimize the number of buses running on each route. To help perform a cost-benefit analysis of the bus network, you would like to write an algorithm to return the highest passenger density for a route, which CapMetro defines as the highest positive change in the number of riders from back-to-back bus stops (of any length).

For example: given $A = \{1, -5, 1, 9, -7, 9, -4\}$, the highest passenger density CapMetro can see is 12 which comes from adding $A[2], A[3], A[4]$, and $A[5]$ together.

Part 1: Design Document [30 points]

Complete the design document template. Make sure you answer every question. You will be graded for the *quality* of your written content, which includes the professional look-and-feel of the presentation and grammar. Take advantage of the campus' writing resources if you need.

Part 2: Test Cases [10 points]

Add 3 to 5 *interesting* test cases. We will judge if your test cases are interesting, and you will only receive credit for those deemed so. Your test cases should follow the same format in the given `Driver.java` class.

Part 3: A Divide and Conquer Approach [35 points]

Since you've taken an algorithms class, you decide to implement your algorithm employing a divide-and-conquer approach, an efficient design paradigm. The basic idea follows the strategy of MERGE-SORT discussed in class. We define an index m and divide the list of average passenger change values into two pieces $a_1, \ldots, a_m$ and $a_{m+1}, \ldots, a_n$. We recursively find the highest passenger density for these two halves. Then we find the highest passenger density that *crosses* the two halves. Lastly, we return the max of these three values.

The pseudocode for the top-level procedure is:

```
findHighestPassengerDensity(A){
    if (length(A) = 1) { return A[1] }
    B = A[1 ... m]
    C = A[(m + 1) ... n]
    r_B = findHighestPassengerDensity(B)
    r_C = findHighestPassengerDensity(C)
    r_A = findCrossPassengerDensity(B,C)
    return (max (r_A + r_B + r_C))
}
```

The more complicated part is defining the pseudocode for FINDCROSSPASSENGEDENSITY(B,C) It turns out you can do this with a fairly straightforward procedure. We'll find the highest passenger density that starts somewhere to the left and ends at the midppoint and the highest passenger density that starts at the midpoint and ends somewhere to the right. Then we add these two together. You have been provided a skeleton file called `DivideAndConquer.java`, which contains a method called `findHighestPassengerDensity`. Add in the code for this method.

Part 4: An Interative Approach [15 points]

You know divide-and-conquer is an efficient algorithm design. However, as you are preparing to hand in your implementation to your boss, you hesitate. You have a nagging feeling that this problem can actually be solved iteratively in $O(n)$ time. Specifically, you think of an approach in which you iterate over each bus-stop and check to update two values: the highest passenger density ending at this bustop and the highest passenger density discovered so far. To update "the highest passenger density ending at this bustop" you can take the highest passenger density from the previous stop and add the current bus stop's value to it. The more complicated part you have to work out is what should happen if "the highest passenger density ending at this bustop" ever turns negative. You have been provided a skeleton file called `Iterative.java`, which also contains a method called `findHighestPassengerDensity`. Add in the code for this method.

Part 5: Reflection Document [10 points]

Complete the reflection document template. Make sure you answer every question. You will be graded for the *quality* of your written content, which includes the professional look-and-feel of the presentation and grammar. Take advantage of the campus' writing resources if you need. This reflection document will ask you to provide evidence of the tangible difference in runtime between your two algorithms.

Getting Started:

- Downdensity and import the code into your favorite environment. There are 3 `.java` files, but you only need to make substantial modifications to `DivideAndConquer.java` and `Iterative.java`. Carefully look through the code provided to you before you start. The set of provided files should compile and run successfully before you touch them at all.
- You must implement the method `findHighestPassengerdensity` in the two classes `DivideAndConquer.java` and `Iterative.java`. You may add methods to these classes if you feel it necessary or useful.
- Class `Driver.java` is provided to test your program. You will add your own test cases to the file and submit it for credit for part 2.

What To Submit and When:

Design Document on 9/16. You should submit a single pdf file titled `LastName_FirstName_Design.pdf`. You ***will*** loose points if you do not follow this submission rule. Why? (1) A pdf file means the formatting will be the same for anyone opening the file on their own machine, and (2) submitting with your name in the file means that when the submissions are downdensityed for grading, your name is attached to the file.

Interesting Test Cases on 9/18. You should submit an updated `Driver.java` file containing your additional tests.

Code on 9/25. You should submit a single file titled `LastName_FirstName_Lab1.tar.gz` or `LastName_FirstName_Lab1.zip` that contains the two java files with your solutions (i.e., `DivideAndConquer.java`, and `Iterative.java`). Do **NOT** include anything else (e.g., we do NOT need the `Driver.java` you used). Failure to follow these submission rules will result in a loss of points.

Reflection on 9/25. You should submit a single pdf file titled `LastName_FirstName_Reflection.pdf`. You ***will*** loose points if you do not follow this submission rule.