CS608 - Spring 2018 - Assignment #1

Assigned: February 7th, 2018
Due: February 16th, 2018
No late submissions. You will be able to complete your submission later if appropriate, but if you do not submit a significant part of the assignment by the due date you will not be able to submit afterwards. Extra credits do not have a deadline.

Collaboration policy: The goal of homework is to give you practice in mastering the course material. Consequently, you are encouraged to collaborate with others. In fact, students who form study groups generally do better on exams than do students who work alone. If you do work in a study group, however, you owe it to yourself and your group to be prepared for your study group meeting. Specifically, you should spend at least 30–45 minutes trying to solve each problem beforehand. If your group is unable to solve a problem, it is your responsibility to get help from the instructor before the assignment is due. **You must write up each problem solution and/or code any programming assignment by yourself** without assistance, even if you collaborate with others to solve the problem. **You are asked to identify your collaborators.** If you did not work with anyone, you must write "Collaborators: none." If you obtain a solution through research (e.g., on the web), acknowledge your source, but write up the solution in your own words. **It is a violation of this policy to submit a problem solution that you cannot orally explain to the instructor.** No other student may use your solutions; this includes your writing, code, tests, documentation, etc. It is a violation of this policy to permit anyone other than the instructor and yourself read-access to the location where you keep your solutions. **For this assignment, team submissions are allowed. Each team can be composed by at most three members. Only one submission per team is required.**

Submission Guidelines: You have to submit your work on Blackboard by the due date. The assignment has two parts: a set of exercises that are not for grade **but still required**, and a programming assignment that includes a coding part and some non-coding questions. Your solutions for the not-for-grade exercises may be submitted separately in any format (even handwriting). The file with your code must include the header template provided in Blackboard. This header must contain, your name, course number, semester, homework number, problem number, a list of collaborators (if none, put "none"), and the team members if you submit as a team. Your answers to non-coding questions must be included in this header as well, in the Remarks section.

Then, to complete your submission, you have to upload two files to Blackboard by the due date: a Java source file and a Java class file.

The submission will not be accepted in any other format.

Style and Correctness: Keep in mind that your goal is to communicate. Full credit will be given only to the correct solution which is described **clearly**. Convoluted and obtuse descriptions might receive low marks, even when they are correct. Also, aim for concise solutions, as it will save you time spent on write-ups, and also help you conceptualize the key idea of the problem.

Assignment #1

Programming Assignment Grading Rubric:

The following rubric applies only to programming assignments.

Program characteristic	Program feature	Credit possible		
Design 30%	Algorithm	30%		
Functionality 30%	Program runs without errors	20%		
	Correct result given	10%		
Input 15%	User friendly, typos, spacing	10%		
	Values read in correctly	5%		
Output 15%	Output provided	10%		
	Proper spelling, spacing, user friendly	5%		
Format 10%	Comments, name	5%		
	Indentation	5%		
	TOTAL	100%		

1(60)	2(20)	3(20)	**TOTAL**(100)	EC

Assignment:

- Problem Set (not-for-grade, not for submission):
  Textbook Exercises 3.1-2, 3.1-4, 4.3-1, 4.3-2, 4.3-3, and 4.3-7
  Textbook Problems 3.2 and 3.3

- Programming Assignment:

The Maximum Subarray Problem is the task of finding the contiguous subarray, within an array of numbers, that has the largest sum. For example, for the sequence of values $(-2, 1, -3, 4, -1, 2, 1, -5, 4)$ the contiguous subsequence with the largest sum is $(4, -1, 2, 1)$, with sum 6.

For an arbitrary input array of length n, two algorithms that compute the sum of the maximum subarray were discussed in class: (a) a brute-force algorithm that solves the problem in $O(n^2)$ steps, and (b) a divide-and-conquer algorithm that achieves $O(n \log n)$ running time.

1. (60 points) Implement in Java the algorithms attached below as Algorithms 1, and 2. Your program must prompt the user to enter the size of the vector n, and output the time taken by each of the three algorithms. Choose at random the numbers in the array (including the sign).

2. (20 points) Test the algorithms with different values of n and fill the following table with the running times measured (put the table in the code header).

	$n = 10^3$	$n = 10^4$	$n = 10^5$	$n = 10^6$	$n = 10^7$
Brute force					
Divide and conquer					

   You may run into problems, such as running out of memory or the program taking too much time. If that is the case, adjust the values of n accordingly.

3. (20 points) Based on the running times observed, draw conclusions about the running times obtained in the analysis. Do they match or not? It is not enough to simply say: yes, they match. You have to justify your claim based on the running times measured (the table). Provide your answers in the remarks section of the code header.

4. (*Extra credit*) There exists a dynamic-programming algorithm due to Kadane that runs in linear time, which is optimal because you need at least to read each number in the input. For extra credit, implement this dynamic programming algorithm as well and test it along the other three.

Algorithm 1: Brute force: $O(n^2)$.

---

input : a vector A of n numbers.
output: maximum subarray sum.

$S_{\max} \leftarrow -\infty$
for $i = 1$ **to** n **do**
  $sum \leftarrow 0$
  for $j = i$ **to** n **do**
    $sum \leftarrow sum + A[j]$
    $S_{\max} \leftarrow \max\{S_{\max}, sum\}$ if (sum > max) max = sum
  end
end
return $S_{\max}$

Algorithm 2: Divide and conquer: $O(n \log n)$.

input : a vector A of n numbers.
output: maximum subarray sum.

```
MAX-SUBARRAY(A,low,high)
    if high = low then
        return A[low]
    else
        mid ← ⌊(low + high)/2⌋
        leftsum ← MAX-SUBARRAY(A,low,mid)
        rightsum ← MAX-SUBARRAY(A,mid + 1,high)
        crosssum ← MAX-CROSSING-SUBARRAY(A,low,mid,high)
        return max{leftsum, crosssum, rightsum}
    end
end

MAX-CROSSING-SUBARRAY(A,low,mid,high)
    leftsum ← −∞
    sum ← 0
    for i = mid to low do
        sum ← sum + A[i]
        if sum > leftsum then
            leftsum ← sum
        end
    end
    rightsum ← −∞
    sum ← 0
    for j = mid + 1 to high do
        sum ← sum + A[j]
        if sum > rightsum then
            rightsum ← sum
        end
    end
    return leftsum + rightsum
end
```