CSCI B505 – Fall 2018

Programming Assignment 2: Due online September 15 , 2018, 11:59pm EST.

In this assignment, you are given a problem for which you are asked to write two programs: First, you use a brute-force approach to solve the problem. Second, you use the divide and conquer method to solve the problem.

What to turn in: There will be **2 files** that you will be turning in via canvas. The first one is your source code, containing your two programs (please stick to healthy coding practices: indent, put in comments, etc); the second one should contain (a) plot and (b) the discussion, as described below. Please try to submit a PDF for the second file. However, other files such as .docx will also be accepted.

The problem: Given an array A of integers, find the nonempty, contiguous subarray of A whose values have the largest sum. For example, consider the following array of integers:
A = 13 -3 -25 20 -3 -16 -23 18 20 -7 12 -5 -22 15 -4 7
Here, the subarray A[8..11] (that is, the subarray 18 20 -7 12), with sum 43, has the greatest sum of any contiguous subarray of array A. So for this example, your program should give the following output: 18 20 -7 12. Obviously, the maximum subarray is not necessarily unique. If there are more than one subarray with the maximum sum, your program should output just one of them.

What to do:

- **Implement** a brute-force solution to the problem by writing YOUR OWN code (in C, C++, Java, or Python).
- **Implement** a solution for the problem, using the divide-and-conquer technique by writing YOUR OWN code (in C, C++, Java, or Python).

  hint: Divide-and-conquer suggests that we divide the array into two subarrays of as equal size as possible. That is, we find the midpoint, say mid, of the input array, and consider the subarrays A[1..mid] and A[mid+1...n]. Now any contiguous subarray A[i...j] of A[1..n] must lie in exactly one of the following places:

  - entirely in the subarray A[1..mid]

- entirely in the subarray A[mid+1...n]
- crossing the midpoint

We can find maximum subarrays of A[1..mid] and A[mid+1...n] recursively, because these two subproblems are smaller instances of the problem. Thus, all that is left to do, is finding a maximum subarray that crosses the midpoint. So, we find X, the maximum subarray of A[1..mid] and Y, the maximum subarray of A[mid+1...n] and Z the maximum subarray that crosses the midpoint. Now, among X, Y and Z, we choose the one with the largest sum. Regarding finding Z, you can easily find a maximum subarray crossing the midpoint in time linear in the size of the input.

- We provide you with an input file that has 100000 random numbers, one number per line. Using this input file, run your two implementations on the first 5000, 10000, 15000, ... up to 100000 numbers and **measure its running time**.
- When you're measuring the running time, insert commands into your code that will compute the time elapsed for finding the maximum subarray only. Disregard time spent on reading the input.
- Run each measurement 3 times to get the average running time. This means that you will be making 20 x 3 = 60 runs, but you should have 20 observations.
- **Visually plot** your measurements on a graph, where the X-axis is the number of inputs and the Y-axis is the time.
- **Discuss:** What does your program return when all elements of A are negative? What input size gives the crossover point at which the divide-and-conquer algorithm beats the brute-force algorithm? Also, if you ran into any special difficulties or made any interesting observations, feel free to mention them here.