COMP 3270 FALL 2016
PROGRAMMING PROJECT: COMPUTATIONAL PROBLEM SOLVING AND THEORETICAL & EMPIRICAL ANALYSES OF ALGORITHM COMPLEXITY
DUE BEFORE 11:59 pm WEDNESDAY NOVEMBER 16
NO EXTENSIONS AND NO LATE SUBMISSIONS
IF YOU DO NOT SUBMIT THE NO-PLAGIARISM CERTIFICATION YOUR PROJECT GRADE WILL BE ZERO- THIS IS NON-NEGOTIABLE!

This homework will give you some experience in computational problem solving, implementing and comparing empirical (i.e., by experimentally measuring) and theoretical (i.e., by calculating T(n)) efficiencies of different algorithms to solve the same problem.

No late submissions will be accepted. If you want to use a programming language other than Java, C or C++ you must first contact the TA and obtain his prior permission.

You may borrow and modify algorithms from books or web sites **as long as** the source of your ideas is clearly and explicitly stated in your submission. For example, Section 4.1 of the textbook will be helpful for this project. You may borrow and modify code from books or web sites **as long as** the source of your ideas is clearly and explicitly stated in your submission. However, what is best for your learning is to come up with at least one strategy and algorithm by yourself, and to write your own code!

Computational Problem Solving

Many computational problems that arise in science and engineering are maximization or minimization problems. Here is one example: given a series of real numbers, find a subset of consecutive numbers that together add up to the smallest possible value. If the input series is -2.2, -1.3, -5.6, -3.5, -7.4 then the subset that produces the minimum total value is the input series itself and that value is -20. If on the other hand the input series is 2.2, 1.3, 5.6, 3.5,7.4 the subset that produces the minimum total value is 1.3 and that value is 1.3. Here is another example: input: 1, -2, 3, 7, -9, 5, 6, 7, -2, 0, 5, 2, -8, 1, -9, 0, -8, 19. Answer: -8, 1, -9, 0, -8 with the smallest possible total value of -24.

Problem Specification

Input: A non-empty array of integers
Output: One integer and two non-negative integers
Correctness criterion: The first output integer must be the smallest possible sum of any subset (including the entire series) of consecutive numbers drawn from the input. The second output non-negative integer should be the input array index of the first integer in this subset. The third output non-negative integer should be the input array index of the last integer in this subset. If there are many subsets that produce the same smallest sum, the output can be any one of these subsets.

PART I

1. Brainstorm and/or do research and come up with three correct but different strategies. Name your strategies strategy-1, strategy-2 and strategy-3.Your three strategies should be different from each other. Write all of them down clearly in plain English and explain them. Consider every design strategy that you know of, including, but not limited to recursion, divide and conquer, incrementally solving the problem through iteration, the "brute force strategy" or "exhaustive search strategy", i.e., calculating the sums of every possible subset of consecutive integers, as well as clever and-non-obvious solutions. **Your submission should include (1) each strategy written out and explained clearly, and (2) for each strategy how you came up with it – by yourself, discussing with friends, from the web (provide the URL), from a book (provide the title), etc.**
2. Develop an algorithm corresponding to each strategy. Name your algorithms algorithm-1, algorithm-2 and algorithm-3. **Your submission should include each algorithm written using Pseudocode with numbered steps.**
3. Calculate the T(n) for each algorithm. **Your submission should show your work in developing the T(n), not just the final T(n). Show details of working out the T(n) using the various methods discussed in class.**

4. Plot each T(n) of each algorithm against n using a spreadsheet. Create a single graph that shows the T(n) plots of all three algorithms. **Your submission should include this graph.**
5. Rank your algorithms in terms of efficiency based on your mathematical analyses. **Your submission should explicitly state the ranking, from most to least efficient, and explain your reasoning.**

If you have any questions about the problem specification or need clarifications, email the instructor. Answers will be broadcast to the class to benefit everyone.

PART II

1. Implement all three algorithms.
2. Write a program that does the following:
    a. Reads a text file named **input.txt** from the current directory formatted as follows:
        i. First line contains a positive integer indicating the size of the input series, i.e., how many numbers are in it.
        ii. The second line will contain a comma-delimited set of negative and positive integers and zeroes. E.g., first line: 18, second line: 1,-2,3,7,-9,5,6,7,-2,0,5,2,-8,1,-9,0,-8,19
    b. Creates a text file named **output.txt** in the current directory containing three lines, with each line containing three comma-delimited integers: smallest subseries sum from the input, array index of the first integer in the smallest subseries, and array index of the last integer in the smallest subseries. The first line should be the output produced by your algorithm-1, the second line output from algorithm-2 and third line algorithm-3. E.g., for the above example input.txt, the corresponding output.txt should be three lines, each with -24,13,17 assuming array indexes start from 1, or -24,12,16 if array indexes start at 0.
    c. Generates input arrays of size 100, 500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000, 5500, 6000, 6500, 7000, 7500, 8000, 8500, 9000, 9500 and 10,000, each containing randomly generated integers in the range [-1000, +1000].
    d. Executes each algorithm on each array and measures the system time taken to produce the answer as follows:
        i. Create the first input array with randomly generated content. You should use a random number generator so that every time your overall program executes, it generates different numbers to fill the input arrays.
        ii. Select your first algorithm.
        iii. Record the system clock value.
        iv. Run the first algorithm on that input array a large number of times – start with 5000 runs, and if needed increase this number so that you can measure the time elapsed correctly given the resolution of your system clock.
        v. Record the system clock value.
        vi. Take the difference in the two system clock values and divide by the number of runs to get the average execution time of that algorithm on that input array.
        vii. Repeat ii-vi for each of the other two algorithms.
        viii. Repeat i-vii for each of the other input sizes.
        ix. Create an output file **time.txt** that contains the data you collected as a 3X21 real-valued matrix written as a comma-delimited Excel-readable file with the following interpretation:
            1. It contains three lines of comma-delimited real numbers.
            2. Each line is a row of this matrix, with each row standing for one of your algorithms. Specifically, the first line should correspond to data relating to your algorithm-1, the second line to algorithm-2 and third line to algorithm-3.
            3. Each line should have 21 comma-delimited real numbers corresponding to the corresponding algorithm's average execution time (step vi above) data for input sizes of 100, 500, 1000, 1500, 2000, 2500, 3000, 3500,

4000, 4500, 5000, 5500, 6000, 6500, 7000, 7500, 8000, 8500, 9000, 9500 and 10,000

3. Read the time.txt into a spreadsheet and create a single graph that includes the execution time plots of all three algorithms. **Your submission should include this graph.**
4. Rank your algorithms in terms of efficiency based on your empirical analyses. **Your submission should explicitly state the ranking, from most to least efficient, and explain your reasoning.**
5. Create three additional plots as follows for each of the three algorithms: overlay its T(n) plot and execution time plot on a single graph. Explain why or why not the two plots match or not match. Do your efficiency rankings based on mathematical and empirical analyses differ? If so, why? **Your submission should include these three overlay plots and your explanations.**

PART III

If you have any doubts/questions about logic or algorithms, ask the instructor before you code. Our expectation is that you know how to program by now. So the instructor or TA will not debug your source code for you. **Your project submission must include:**

1. Every submission item described in **bold** above compiled into one pdf document.
2. Source files that are properly commented (if you reuse code fragments from another source such as the text or web, clearly identify the fragments and their source in comments) and indented code including all procedures, functions, classes, etc. and the main program – in short, everything we need in order to compile and run your program.
3. Create a text file with your name, your email address, the names of all source files in the zipped archive, an explanation of how you compiled your program (what IDE or command line compilation command for the compiler you used), and the following certification statement (verbatim): "**NO-PLAGIARISM CERTIFICATION: I certify that I wrote the code I am submitting. I did not copy whole or parts of it from another student or have another person write the code for me. Any code I am reusing in my program is clearly marked as such with its source clearly identified in comments.**" Without this text file that includes the above statement, your submission will NOT be graded. If we suspect or find evidence of plagiarism, your certification will be used in academic dishonesty proceedings.

Submit to Canvas the certification file, all source files and the pdf file as a single zipped folder before the deadline on the due date.

If the TA has trouble with your submission you will get an email at the same email address that you provided in the text file described above. It is up to you to check this email address and sort out any such problem cooperatively with the TA. If this is not done in a timely manner, your homework will not be graded.

Caution: Your work should follow the academic integrity guidelines stated in the syllabus. Do your own work; do not copy that of another. We may run code plagiarism detection software on your submissions. If we find evidence of copying, you will automatically receive a zero for this homework **and** we will refer your plagiarism to the appropriate university committee and your certification that you did not copy will be used in the proceedings.

GRADING

We will not try to debug your program; it is your responsibility to send a correct program that will compile, run, will not crash, and produce the correct output. It is strongly recommended that you test your program before submitting. We will run it on several test files. Programs that are not well-organized, not commented properly, and not written following the directions are likely to lose points. If you make any additional assumptions about the input, beyond the ones stated in this document, your program may not run correctly and you may lose points.

The homework will be graded out of a maximum of 100 points.
The minimum requirement is to turn in items 1-3 listed above. If you do not meet this requirement, you will get 0 points. If you do, your grade will be determined through the following steps:

1. If your source code does not match your algorithms, you will get 0 points and the grading of your submission will end.
2. If your source code is properly indented and commented and therefore easy to read and understand, you will receive 10 points.
3. If your source does not compile, you will get 0 additional points and the grading of your submission will end.
4. If the source compiles correctly but the executable does not run or crashes, you will get 0 additional points and the grading of your submission will end.
5. If the executable runs, but produces no output or completely wrong output, you will get 0 additional points and the grading of your submission will end. Otherwise:
6. Your program will be tested on three input.txt files. For each, if it produces a fully correct output.txt file, you will receive 10 points for a total of 30 additional points. There will be partial credit for partially complete or partially correct output.
7. If your program correctly and completely produces the comma-delimited text file time.txt, you will get 30 additional points. There will be no partial credit for incomplete or incorrect output.
8. If your submission includes all items listed in PARTS
9. I & II, we will review and grade it for a maximum of 30 additional points.