Group Assignment 3

CS325 Winter 2016

Due: Tuesday, February 23 at 2PM

You are required to work in groups of two or three students. See the provided group guidelines for how to form your groups on canvas. Each group should only submit one copy of the work to canvas, including the **project report as a pdf** and the code that implements the algorithms. The report should have **all member's names included**. You may use Java, Python, C/C++, Matlab, or R. The submission to Canvas can include multiple files. Acceptable file types include pdf, zip, gzip. So you should zip up your code for submission. It is also required that the report is submitted as a separate pdf file so that the TA can directly comment on it via canvas.

The Closest-to-Zero Problem

Consider the following problem: Given an array of n integers $A[0, 1, \ldots, n-1]$, find indices i and j such that $i \leq j$ that minimizes

$$\left| \sum_{k=i}^{j} A[k] \right| .$$

That is, given any array, find a subarray whose sum is closest to zero. For example if

$$A = [31, -41, 59, 26, -53, 58, -6, 97, -93, -23]$$

then using $i = 4$ and $j = 6$ gives $\left|\sum_{k=4}^{6} A[k]\right| = |-53 + 58 - 6| = 1$ and this is the best you can do.

You can solve this problem by enumerating over all $\Theta(n^2)$ choices for i and j and computing the sum, keeping the sum closest to zero found so far. By reusing computation as you did for the maximum subarray problem in the first group assignment, this algorithm will take $\Theta(n^2)$ time. In this assignment, you will develop a divide-and-conquer algorithm that is asymptotically faster.

A divide-and-conquer approach

If we split the input array into two halves, we know that the subarray whose sum is closest to zero will either be

- contained entirely in the first half,
- contained entirely in the second half, or
- made of a suffix of the first half and a prefix of the second half.

If we have computed these three options (the first two options recursively), then we simply take the best of the three options. For our example array A used above, we would take the best of

- the closest-to-zero subarray of $[31, -41, 59, 26, -53]$
- the closest-to-zero subarray of $[58, -6, 97, -93, -23]$

- the closest-to-zero subarray formed by a suffix of $[31, -41, 59, 26, -53]$ and a prefix of $[58, -6, 97, -93, -23]$

To find the latter case, we can consider the sums of the suffices of $[31, -41, 59, 26, -53]$:

$$22, -9, 32, -27, -53$$

and the sums of prefices of $[58, -6, 97, -93, -23]$:

$$\underline{58}, \underline{52}, \underline{149}, \underline{56}, \underline{33}$$

The best suffix-prefix pair can be found by finding a number in the first list and a number in the second list that when added together is closest to zero or, equivalently, finding a number in the first list that is closest to *the negative* of a number in the second list. Consider the following three methods for solving this *suffix-prefix identification problem*:

Method 1 Compare every number in the first list with every number in the second list.

Method 2 Sort the first list $(-53, -27, -9, 22, 32)$ and sort the second list $(33, 52, 56, 58, 149)$ and iterate the two lists carefully to identify the two numbers you are looking for in (-53 and 52).

Method 3 Combine the first list with the negative of the second list:

$$22, -9, 32, -27, -53, \underline{-58}, \underline{-52}, \underline{-149}, \underline{-56}, \underline{-33}$$

and sort this combined list:

$$\underline{-149}, \underline{-58}, \underline{-56}, -53, \underline{-52}, \underline{-33} - 27, -9, 22, 32$$

keeping track of which list the numbers come from (as we have done with underlining) and noticing that the two numbers you are interested in are adjacent to one another (that is, $-53, \underline{-52}$).

Tasks

1. Write pseudocode for each of the three methods for the suffix-prefix identification problem described above. Analyze the running time of each of the three methods.

2. Write pseudocode for a divide and conquer algorithm for the closest-to-zero problem that uses the suffix-prefix identification problem as a subroutine, but doesn't specify which of the methods to use..

3. The three methods for suffix-prefix identification potentially give three different (although not necessarily unique, asymptotically) running times when used as a subroutine for the divide and conquer algorithm. For each method, write a recurrence relation that describes the running time of the resulting divide and conquer algorithm; solve each of these recurrence relations.

4. Implement the divide and conquer algorithm using *either* method 2 or 3 for suffix-prefix identification. A file containing test sets, named `test_cases_with_solutions.txt` is provided on canvas. The file has one test case per line (10 cases each with 100 entries) with the input array followed by the <u>absolute value</u> of the sum of the closest-to-zero subarray and the corresponding start and end indices (with indices start at 0). A line corresponding to the example above would be:

   ```
   [31, -41, 59, 26, -53, 58, -6, 97, -93, -23],1,4,6
   ```

   You may use this test file to check that your code is correct. You should also test your code on small hand-generated instances.

5. Another file containing several instances without solutions (`test_cases_without_solutions.txt`) is also provided on canvas. Similarly, each line is a separate input array. **Your submission to Canvas should contain a text file named `answers.txt` that contains your answers for the test cases in this file.** Each line of your file represents your answer for the test case on the corresponding line of test cases file. This means that your file should contain exactly the same number of lines that the test cases file contains. Each line of your file should contain three numbers separated by white space (spaces or tabs). The first number should be the absolute value of the sum of the closest-to-zero subarray, the second number should be the start index, and the last number should be the end index; note that indices start at 0.

6. Plot the experimental run time of your algorithm as a function of input size for the inputs in the file `test_cases_without_solutions.txt`. (You may want to run the algorithm several times and average the running times to get a good measurement.)

7. **Your pdf report should include pseudocode for each of the three methods for the suffix-prefix identification problem, pseudocode for a divide and conquer algorithm for the closest-to-zero problem, three recurrence relations and their solutions and the plot described above.**

Optional Bonus (up to 20%) There is an algorithm whose run-time for this problem is asymptotically better than the divide-and-conquer method suggested. Give pseudocode for this algorithm and analyze its run time. Prove that your algorithm is correct. No hints will be provided for the bonus part.

Note on sorting You may use a built-in sorting subroutine. That is, you do not need to implement your own sorting algorithm. However, in analyzing the above algorithms, you should specify the running time of the sorting algorithm *you* use.