For this project, you will revisit the close-to-zero problem and design a (hopefully) more efficient divide and conquer algorithm. Recall the close-to-zero problem:

Given array of small integers $a[0, 1, \ldots, n-1]$, compute

$$\min_{0 \leq i \leq j < n} \left| \sum_{k=1}^{j} a[k] \right|.$$ 

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

For example, $\text{CLOSETOZERO}([31, -41, 26, -53, 58, -6, 97, -93, -43, 84]) = 1$

To get there we will start with a warm-up problem, the sum-of-suffices problem.

**Sum of suffices**

Consider the following problem:

Given two arrays of small integers $b[0, 1, \ldots, \ell - 1]$ and $c[0, 1, \ldots, m - 1]$, compute:

$$\min_{0 \leq s < \ell, 0 \leq t < m} \left| \sum_{i=s}^{\ell-1} b[i] + \sum_{j=t}^{m-1} c[j] \right|$$

That is, find a suffix of $b$ and a suffix of $c$ such that the sum of both is as close to zero as possible.

For example, $\text{SUMOFSUFINES}([59, 26, -53, 58, -6, 97, -93, -23], [9, -74, 68, 4, 100, 67, 95]) = 21$

It may be helpful to think of this as finding a suffix $b'$ of $b$ and a suffix $c'$ of $c$ such that the sum of $b'$ is as close as possible to the negative of the sum of $c$. In the following, let

$$S_b[s] = \sum_{i=s}^{\ell-1} b[i] \text{ for } s = 0, 1, \ldots, \ell - 1 \text{ and } S_c[t] = \sum_{j=t}^{m-1} c[j] \text{ for } t = 0, 1, \ldots, m - 1$$

Note that you can compute all these sums in linear time. You will implement two algorithms for \text{SUMOFSUFINES} based on these ideas:

**Algorithm 1: Enumerate** Loop over every pair $s$ and $t$ and compute the sum, keeping the sum closest to zero.
Algorithm 2: Sort and Compare

Sort the array

\[ A = [S_b[0], S_b[1], \ldots, S_b[\ell - 1], -S_c[0], -S_c[1], \ldots, -S_c[m - 1]]. \]

Can you use the resulting sorted array to determine \textsc{SumOfSuffices}[b, c] more quickly than the enumeration algorithm? Which elements in the sorted list should you compare?

Be sure to test your algorithm for correctness! It is helpful to hand-create some instances for this purpose.

**Divide and conquer**

You will use \textsc{SumOfSuffices} to help design a divide and conquer algorithm \textsc{CloseToZero} for the close-to-zero problem. If we split the input array for \textsc{CloseToZero} 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.

The first two cases can be found recursively. How can you use \textsc{SumOfSuffices} to implement the last case?

You will implement two versions of \textsc{CloseToZero}:

**Algorithm 3: Divide and Conquer using Enumeration**

Using the enumeration implementation of \textsc{SumOfSuffices} (Algorithm 1) to implement this last case.

**Algorithm 4: Divide and Conquer using Sort and Compare**

Using the sort-and-compare implementation of \textsc{SumOfSuffices} (Algorithm 2) to implement this last case.

Be sure to test your algorithms! You can compare your results to those obtained in the first project.

**Project report**

Your report must include:

**Run-time analysis**

Give pseudocode for each of the four algorithms and an analysis of the asymptotic running-times of the algorithms.

**Proofs of Correctness**

Give a proof by contradiction that Algorithm 2 returns the correct solution. Give a proof by induction that Algorithm 4 returns the correct solution.

**Experimental analysis**

Perform an experimental analysis of the two implementations of \textsc{CloseToZero} as described in the first project. For your plots, include the data collected for the two implementations of \textsc{CloseToZero} that you performed in the first project.

**Extrapolation and interpretation**

Use the data from the experimental analysis to answer the following questions:

1. For algorithms 3 and 4, what is the size of the biggest instance that you could solve with your algorithm in one hour?
2. Determine the slope of the lines for algorithms 3 and 4 in your log-log plot and from these slopes infer the experimental running time for these algorithms. Discuss any discrepancies between the experimental and theoretical running times.

**Code**

Upload your code to T.E.A.C.H. Only one student from each group should do this.