This homework is due Thursday, September 17, at 2:00pm. Please upload a single PDF file containing your submission (ensuring scans of handwritten work are legible) on Gradescope by that time.

The questions are drawn from Chapters 1 and 2 of the text, and from the introductory lectures in class on algorithm design for the Maximum Sum Subarray Problem. The homework is worth a total of 100 points.

For questions that ask you to design an algorithm, you should:

(i) describe an algorithm for the problem using prose and pictures,
(ii) argue that your algorithm is correct, and
(iii) analyze the running time of your algorithm.

Pseudocode is not required, but do clearly explain the ideas behind your algorithm. As a general rule, only give high-level pseudocode if it aids the time analysis of your algorithm, or if it is needed to clarify the steps.

Remember to (a) start each problem on a new page, and (b) put your answers in the correct order. If you can’t solve a problem, state this, and write only what you know to be correct. Neatness and conciseness counts.

(1) **(Best-case time)** (10 points) Show how to take nearly any algorithm (even a poor one), and modify it so that it has good best-case time.

To answer this question, describe a procedure that (1) takes as input a problem statement $P$ together with an algorithm $A$ that solves $P$, and (2) outputs a new algorithm $A'$ for $P$ whose best-case running time is “as good as possible.” An intelligent human being (such as yourself) should be able to carry out your procedure.

(Note: The original algorithm $A$ might be very inefficient. So what does this tell you about best-case time?)

(2) **(Counting flips)** (20 points) A flip in an array $A[1:n]$ of numbers is a pair of indices $(i, j)$ such that $i < j$ and $A[i] > A[j]$. In other words, a flip is a pair of elements that are not in sorted order.

(a) (10 points) Find an array of length $n$ over the elements \{1, ..., $n$\} that maximizes the number of flips, and prove that it is optimal.

(b) (10 points) Suppose you run insertion sort on an array of length $n$ that has $k$ flips. Derive a tight big-$O$-bound on the running time of insertion sort as a function of both $n$ and $k$, and explain your analysis.


(c) **(bonus)** (10 points) By modifying merge sort, design an algorithm that counts the number of flips in an array of length $n$ in $O(n \log n)$ time.

(Note: There can be $\Theta(n^2)$ flips in an $n$-element array, so your algorithm cannot explicitly list them and achieve the time bound. It is possible to count flips without listing them.)
(3) **(2-D maximum-sum subarray)** (30 points) In the 2-D Maximum-Sum Subarray Problem, you are given a two-dimensional \( m \times n \) array \( A[1 : m, 1 : n] \) of positive and negative numbers, and you are asked to find a subarray \( A[a : b, c : d] \), where \( 1 \leq a \leq b \leq m \) and \( 1 \leq c \leq d \leq n \), such that the sum of its elements, \( \sum_{i \leq a \leq b} \sum_{j \leq c \leq d} A[i, j] \), is maximum.

(a) (30 points) Using exhaustive search, design an algorithm that runs in \( O(m^2n^2) \) time using \( O(\min\{m, n\}) \) working space.

The *working space* of an algorithm is the memory it uses beyond what is needed to store the input.

(Hint: First design a straightforward algorithm that achieves \( O(m^3n^3) \) time, and then optimize it.)

(b) (bonus) (10 points) Using divide-and-conquer, design an algorithm that runs in \( O(m^2n \log n) \) time using \( O(n) \) working space, by dividing the array vertically.

(Hint: You may need to know that the recurrence
\[
T(m, n) = 2T(m, n/2) + O(m^2n)
\]
has the solution \( T(m, n) = O(m^2n \log n) \).)

(4) **(Minimum positive-sum subarray)** (40 points) In this variation of the Maximum-Sum Subarray Problem, you are given a one-dimensional array \( A[1 : n] \) of positive and negative numbers, and you are asked to find a subarray \( A[i : j] \) such that the sum of its elements is both: (1) strictly greater than zero, and (2) minimum. In other words, you want to find a subarray of smallest positive sum.

(a) (40 points) Using the *divide-and-conquer* strategy, design an algorithm that finds a minimum positive-sum subarray in an array of length \( n \) in \( O(n \log^2 n) \) time.

(Hint: Recall that \( n \) numbers can be sorted in \( O(n \log n) \) worst-case time. You may also need to know that the recurrence
\[
T(n) = 2T(n/2) + O(n \log n)
\]
has the solution \( T(n) = O(n \log^2 n) \).)

(b) (bonus) (10 points) Using an *incremental strategy*, now design an algorithm that finds a minimum positive-sum subarray in \( O(n \log n) \) time. The incremental strategy proceeds by solving a subproblem of size \( k \), adding one element, and updating the solution to solve a problem of size \( k+1 \).

(Hint: You may find a balanced search tree useful.)

Bonus questions (like Problems 2(c), 3(b), and 4(b) above) are *not* required. Bonus points are tallied separately, and are considered at the end of the semester for students who fall below the cutoff for a letter grade.