These are practice problems for the upcoming midterm exam. You will be given a sheet of notes for the exam. Also, go over your homework assignments. **Warning:** This does not necessarily reflect the length, difficulty, or coverage of the actual exam.

Problem 1. Consider the following algorithm for finding the two smallest elements in a list: Sort the first two elements. Call them *special*. Then iterate through the remainder of the elements one at a time. First compare each new element to the larger of the two special elements, and if necessary compare it to the smaller. The two new special elements will be the smallest two of those three elements.

(a) Write the pseudocode for this algorithm.

(b) What is the exact best case number of comparisons? Justify and simplify.

(c) What is the exact worst case number of comparisons? Justify and simplify.

(d) What is the exact average case number of comparisons? Justify and simplify.

Problem 2. Solve the following recurrences exactly using the tree method. You may assume n is "nice". Prove your answers using mathematical induction.

(i) $T(n) = 2T(n/2) + n^3$, $T(1) = 1$.

(ii) $T(n) = T(\sqrt{n}) + 1$, $T(2) = 1$.

(iii) $T(n) = 2T(n/2) + n \lg n$, $T(1) = 1$.

(iv) $T(n) = T(n-3) + 5$, $T(1) = 2$.

Problem 3. Assume you have an array of numbers, where each value occurs *at most* twice. We consider sums of contiguous numbers in the array. But we only consider such sums whose two endpoints have the same value. The sum includes the two equal values themselves. So if the two equal numbers are at index i and index j ($i < j$) in array A, then we sum all the values $A[i], A[i+1], \ldots, A[j]$.

(a) Give an algorithm that finds the maximum such sum. Make your algorithm as efficient as possible. Describe the algorithm briefly in English *and* in psuedo code.

(b) Analyze the running time of your algorithm.

Problem 4. Let $A[1, .., n]$ be an array of n numbers (some positive and some negative).

(a) Give an algorithm to find which two numbers have sum closest to zero. Make your algorithm as efficient as possible. Write it in pseudo-code.

(b) Analyze its running time.