1. Henceforth in the course, given a a graph $G = (V, E)$, let n be the number of vertices and m be the number of edges.

   In the adjacency list representation of a graph, the vertices are numbered from 0 to n-1. Describe an $O(n + m)$ algorithm to sort every vertex's adjacency list so that its neighbors appear in ascending order. Explain why the $O(n + m)$ bound applies.

2. Let is say that an array of unique floating-point numbers is *mostly sorted* if no element is farther than 20 positions from the position where it belongs in a sort of the array.

   Give an $O(n)$ algorithm to sort a mostly-sorted array.

   Note that being mostly sorted does not place any constraint on the values of the elements, other than that they can each be represented with one word in the RAM model.

   Hint: Use one of the sorting algorithms we have talked about this semester.

3. What is the DFT of $(1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$? There is no reason for a lot of calculation; reason it out using your understanding of how the DFT is defined. Explain your reasoning.

4. What is the DFT of $(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)$? Proceed as in the previous problem.

5. Consider the following polynomial: $A(x) = 3 + 4x + x^2 + 3x^3$.

   (a) Compute the DFT of A by hand, using the naive $O(n^2)$ approach. Show your work.

   (b) What are the two polynomials passed to the two recursive calls when the FFT is called on A?

   (c) What sequences do these two calls pass back?

   (d) Show how the FFT combines these two sequences to produce the DFT of A.

6. Different-numbers of recursive calls for the FFT.

   (a) Let n be a degree bound for a polynomial $A()$, and assume n is a power of three that is greater than or equal to 3. Come up with a variant of Equation 30.9 on page 910 that expredsses A in terms of three polynomials, $A^{[0]}$, $A^{[1]}(x)$ and $A^{[2]}(x)$, each with each of degree bound $n/3$.

   (b) Derive a recurrence and big-Θ time bound in closed form for a version of the FFT that uses this decomposition of A instead of Equation 30.9.

   (c) Repeat these two steps to get an FFT that makes five recursive calls instead of two or three. Place any convenient constraint on n.

(d) It should now be clear that we can divide our polynomial into k recursive calls. Give the recurrence and big-Θ bound in closed form for this. Since k is now a variable, it should appear in your expressions.

Why does your big-Θ bound imply that it doesn't pay to choose large values of k?

7. Here is different idea for multiplying polynomials $A()$ and $B()$: Find the following products recursively: $A^{[0]} * B^{[0]}$, $A^{[0]} * B^{[1]}$, $A^{[1]} * B[0]$, $A^{[1]} * B^{[1]}$.

   (a) What is supposed to be returned by these four recursive calls when used to compute $A() * A()$ for the polynomial $A(x) = 1 + 2x + 3x^2 + 4x^3$. Your answer should be four polynomials of degree bound two.

   (b) What is $A() * A()$, where $A(x) = 1 + 2x + 3x^2 + 4x^3$. (I would calculate this using the naive approach.)

   (c) Illustrate how to combine the coefficients of the four polynomials you obtained in part (a)to obtain the coefficients of $A() * A()$. Your approach should illustrate an operation that takes $O(n)$ time in general.

   (d) Derive a recurrence $T(n)$ and big-Θ time bound in closed form for this recursive algorithm.

   (e) Illustrate how to get a faster time bound than this by instead making recursive calls to compute $(A^{[0]} + A^{[1]}) * (B^{[0]} + B^{[1]})$, $A^{[0]} * B^{[0]}$ and $A^{[1]} * B[1]$.

   Illustrate an $O(n)$ step that combines the results of these recursive calls.

   Then derive a recurrence and the improved time bound.