Problem 1: Given a polynomial \( P(x) \) of degree \( n \) and a value \( a \), show how to divide \( P(x) \) by \( (x - a) \) in \( \Theta(n) \) time. In other words you must compute polynomial \( Q(X) \) of degree \( n - 1 \) and value \( r \) such that

\[
P(x) = Q(X) \cdot (x - a) + r
\]

How can you use the above algorithm to compute the value \( P(a) \)?

Solution: Let \( P(x) = p_0 + p_1 x + p_2 x^2 + \ldots + p_{n-1} x^{n-1} + p_n x^n \) and \( Q(x) = q_0 + q_1 x + q_2 x^2 + \ldots + q_{n-1} x^{n-1} \). Set \( q_n = 0 \) and \( q_{-1} = r \). The equality

\[
P(x) = Q(x) \cdot (x - a) + r
\]

implies that \( p_i = -aq_i + q_{i-1} \) for all \( i = 0, \ldots, n \). This immediately yields the desired algorithm since one can set \( q_{n-1} = p_n \) and compute all the coefficients of \( Q \) in descending order.

Notice that \( P(a) = r \) and therefore this is an alternative way to compute a polynomial at a value \( a \) in linear time.

Problem 2: Given a sequence of values \( a_1, \ldots, a_n \) give an algorithm that finds the coefficients of the polynomial \( P(x) \) of degree \( n \) such that \( P(x) = 0 \) if and only if \( x = a_i \) for some \( i \). You can assume the elements \( a_i \) are distinct. Your algorithm should run in time \( O(n \log^2 n) \).

Solution: The desired polynomial can be written as \( P(x) = \prod_{i=1}^{n} (x - a_i) \), i.e. as the product of \( n \) degree \( 1 \) polynomials. To compute the coefficient representation we use a divide and conquer approach. We compute \( P(x) = P_l(X) \cdot P_r(x) \) where \( P_l(x) = \prod_{i=1}^{n/2} (x - a_i) \) and \( P_r(x) = \prod_{i=n/2+1}^{n} (x - a_i) \). We compute \( P_l(X) \) and \( P_r(x) \) by recursion, and we compute \( P(x) \) using the FFT-based polynomial multiplication algorithm, that on polynomials of degree \( n/2 \) runs in time \( O(n \log n) \).

Therefore the entire running time of the algorithm is described by the recurrence \( T(n) = 2T(n/2) + O(n \log n) \) which yields \( T(n) = O(n \log^2 n) \).

Problem 3: Given two patterns \( P \) and \( P' \) you want an algorithm that determines where \( P \) or \( P' \) appears in a given text \( T \). Builds a finite automaton that enters an accepting state only in the above situation. Try to make your automaton as simpler as possible (meaning your score will depends on how small the automaton is).

Solution: Let \( A \) be the maximal prefix shared by \( P \) and \( P' \), and let \( B \) be the maximal suffix shared by \( P \) and \( P' \) (note that these could be the empty string). Therefore \( P = AMB \) and \( P' = AM'B \) where \( M \) and \( M' \) are the "middle" strings in each pattern.

Build the finite automata for \( P \) and \( P' \) and then merge the sections that correspond to \( A \) and \( B \) (see picture).