1. Give an \( \mathcal{NC} \) algorithm which finds the maximum of \( n \) numbers is \( O(\log n) \) time using \( n/\log n \) processors.

Let \( m = \lceil n/\log n \rceil \). Let \( x_1 \ldots x_n \) be the numbers, and let \( p_1 \ldots p_m \) be the processors. Partition the numbers into \( m \) batches of size no greater than \( \lceil \log n \rceil \) and let \( y_j \) be the maximum of the \( j^{th} \) batch. Processor \( j \) computes \( y_j \) in time which is a linear function of the size of the batch, that is, in \( O(\log n) \) time. We then compute the overall maximum using the tournament method, in \( O(\log m) \) steps. Initially, all \( \{y_j\} \) are candidates for the maximum value. During each step, the candidates are compared, in pairs. Each pair is compared by one processor, and the smaller is discarded, in \( O(1) \) time. Within \( O(\log m) = O(\log n) \) steps, there is one remaining candidate, the maximum value.

2. Prove that every regular language is in Nick’s Class.

Since we use “+” to denote “or,” we use the summation sign to denote disjunction of several Booleans. For example, if \( x_1, \ldots, x_n \) are Boolean expressions, \( \sum_{i=1}^{n} x_i \) means \( x_1 + \ldots + x_n \).

Let \( L \) be a regular language over an alphabet \( \Sigma \), and let \( M \) be an NFA which accepts \( L \). For simplicity, we do not allow \( M \) to have \( \lambda \) transitions. Let \( Q = \{ q_0, \ldots, q_m \} \) be the states of \( M \). Since the size of \( M \) does not change as we consider strings over \( \Sigma \) of any length, \( m \) is a constant. Let \( F \subseteq Q \) be the final states of \( M \), and \( \delta : Q \times \Sigma \rightarrow 2^Q \) the transition function. Let \( \delta^*: Q \times \Sigma^* \rightarrow 2^Q \) be the transitive closure of \( \delta \): that is, for any \( u \in \Sigma^* \), \( q \in \delta^*(q_0, u) \) means that if \( M \) is in state \( q_0 \), it could be in state \( q \) after reading the string \( u \). For any \( 0 \leq i, j \leq m \) and any \( u \in \Sigma^* \) define \( S[i, u, j] \) to be the Boolean value which is true if and only if \( q_j \in \delta^*(q_i, u) \).

Let \( w \in \Sigma^* \), and let \( n = |w| \). We can determine whether \( w \in L \) in \( O(\log n) \) time using \( O(n^2) \) processors. Recall that \( w \in L \) if and only if there is some \( q_f \in F \) such that \( S[0, w, j] \) is true. We execute a dynamic program where the subproblems are the \( S[i, u, j] \). There are infinitely many such subproblems, but we only consider \( O(n^2) \) of them, those for which \( u \) is a substring of \( w \).

Observe that

(a) \( S[i, \lambda, i] \) for all \( i \).

(b) \( S[i, \lambda, j] \) is false for all \( i \neq j \).

(c) If \( a \in \Sigma \) then \( S[i, a, j] \) if and only if \( q_j \in \delta(a, q_i) \).

(d) Concatenation rule: for any \( u, v \in \Sigma^* \) and any \( q, q_j \in Q \), \( S[i, uv, j] = \sum_{k=0}^{n} S[i, u, k] \ast S[k, v, j] \).

The dynamic program is initialized using the first three rules. The remainder of the computation consists of repeatedly applying the concatenation rule, but only when \( uv \) is a substring of \( w \) and \( S[i, u, k] \) and \( S[k, v, j] \) have already been computed. There are \( O(n^2) \) such concatenation steps, since we only allow choices of \( u \) and \( v \) of approximately the same length, and we use one processor for each concatenation. After \( t \) time steps, \( S[i, u, j] \) has been computed for every substring \( u \) of \( w \) of length no greater than \( 2^t \). Thus, after \( O(\log n) \) time, \( S[i, w, j] \) has been computed for all \( i, j \), and we are done.

**Fewer Processors.** The problem can actually be solved in \( O(\log n) \) time using only \( n/\log n \) processors. Do you see how?