1. As regards Accelerated Cascading Max, analyze this algorithm if the binary tree reduction cutoff is:
   a.) lg lg lg lg N
   b.) square root(lg N)

   Determine which are fast and/or efficient. Do precise analysis.

   a) lg lg lg lg N steps in binary reduction requires O(lg lg lg lg N) time and no more than O(N) work. The problem size is now reduced to N/ 2^(lg lg lg lg N) or N/(lg lg lg N) elements. The doubly log algorithm now completes in O(lg lg N) time, but takes N (lg lg N)/(lg lg lg N) work. Thus, even though the time is fine, the work exceeds our goal of O(N).
   b) sqrt(lg N) steps in binary reduction requires more than lg lg N time, as O(sqrt k) contains O(lg N), but not vice versa. The work is still O(N). The doubly log has N/2^sqrt(lg N) elements to reduce, but that’s fewer than N/lg N, and can be reduced in O(lg lg N) time taking (N/lg N * lg lg N) work. But that’s O(N) work. Thus, the work is fine, but the time is too long.

2. For each of (a) Bitonic Sort and (b) lg lg trees Max, operating on N values, determine if there is a magic p (similar to Brent's Scheduling), for which this algorithm is work efficient and fast ($\lg^2 N$ and lg lg N, resp.) when virtualized with each processor starting with N/p values. Prove that your value of p is optimal, as in Brent's choice of p = N/lg N, or argue convincingly that no such p can be found for arbitrary N.

   a) At a prepass, we do local sorts, taking N/p lg(N/p) time and N lg(N/p) work. At each pass, we take N/p time and do N work. At convergence, we have spent N/p * lg(N/p) + (N/p) $\lg^2 p$ time and done N lg(N/p) + N$\lg^2 p$ work. To get work down to N lg N, we will assume p = 2^k, for some k. That's a good assumption for Bitonic. Now, $\lg^2 p$ is then k^2. Letting N=2^j, we can choose p=$2^{\text{sqrt}(j)}$. Thus, p must be $2^{\text{sqrt}(\lg N)}$ in order to keep the work under control. Plugging in, we get time of $N/2^{\text{sqrt}(\lg N)} * (\lg(N/2^{\text{sqrt}(\lg N)}) + \lg^2 2^{\text{sqrt}(\lg N)}$ or $2^j / 2^{\text{sqrt}(j)} * (\lg(2^j / 2^{\text{sqrt}(j)}) + \lg^2 2^{\text{sqrt}(j)}$ or $2^{j-\text{sqrt}(j)}$ * (j-sqrt(j) + j) = O((N-sqrt(N)) lg N). Unfortunately, that's a bad time, so there is no p that satisfies our needs.
   b) At a prepass, we do local max, taking N/p time (really (N-1)/p) and N work (really N-1). We would then compute the max in lg lg (p) time and p lg lg (p) work. Total time is N/p + lg lg p. Total work is N + p lg lg p. Let p = N/lg lg N. Time is then lg lg N and work is N, as desired.