1. It was shown in class that the maximum of \( n \) elements can be found in \( O(1) \) time using \( n^2 \) common CRCW PRAM processors.

Consider the case when \( \epsilon = \frac{1}{2} \). Divide the elements into groups of size \( \sqrt{n} \). Assign the first \( \sqrt{n} \) elements to the first \( n \) processors and the second \( \sqrt{n} \) elements to the next \( n \) processors and so on. The maximum element in each group can be found in \( O(1) \) time. At this stage, we have \( \sqrt{n} \) elements and \( n\sqrt{n} \) processors. Hence, the maximum of these elements can be found in \( O(1) \) time. Total time = \( O(1) \).

Next, consider the case when \( \epsilon = \frac{1}{3} \). Here, divide the elements into groups of size \( n^{1/3} \). Assign the first \( n^{1/3} \) elements to the first \( n^{2/3} \) processors and the second \( n^{1/3} \) elements to the next \( n^{2/3} \) processors and so on. The maximum element of each group can be found in \( O(1) \) time and using \( n^{4/3} \) processors the maximum of these maximum elements can be found in \( O(1) \) time.

For the general case, partition the input into groups with \( n^\epsilon \) elements in each group. Find the maximum of each group assigning \( n^{2\epsilon} \) processors to each group. This takes \( O(1) \) time. Now the problem reduces to finding the maximum of \( n^{1-\epsilon} \) elements. Again, partition the elements with \( n^\epsilon \) elements in each group and find the maximum of each group. There will be only \( n^{1-2\epsilon} \) elements left. Proceed in a similar fashion until the number of remaining elements is \( \leq \sqrt{n} \). The maximum of these can be found in \( O(1) \) time. Clearly, the run time of this algorithm is \( O(1/\epsilon) \). This will be a constant if \( \epsilon \) is a constant.

2. Let \( A \) and \( B \) be the two given \( n \times n \) matrices. Let \( C \) be the product. Clearly, \( C[i, j] = \sum_{k=1}^{n} A[i, k] \times B[k, j], \) for \( 1 \leq i, j \leq n \). We can assign \( n \) processors to calculate each entry in the product matrix \( C \). Consider the computation of \( C[i, j] \) for some specific values \( i \) and \( j \). Let the \( n \) associated processors be 1, 2, \ldots, \( n \). In parallel processor \( k \) computes \( A[i, k] \times B[k, j] = c_k \), for \( k = 1, 2, \ldots, n \). This takes one step. Followed by this, all the \( n \) processors compute the prefix sums value of the sequence \( c_1, c_2, \ldots, c_n \). This takes \( O(\log n) \) time. Let the prefix sums be \( c'_1, c'_2, \ldots, c'_n \). Note that \( C[i, j] = c'_n \).

The run time of the above algorithm is \( O(\log n) \) and the processor bound is \( n^3 \). We can reduce the processor bound to \( \frac{n^3}{\log n} \).
3. We know that $\pi_1$ polynomially reduces to $\pi_2$. Let $x$ be an instance of $\pi_1$ with $|x| = n$. We can convert this into an instance $x'$ of $\pi_2$ in $O(n^c)$ time (for some constant $c$). Note that $c$ could be any constant (10, for instance) and we can only say that $|x'| = O(n^c)$ and in fact $|x'|$ could be $\Omega(n^c)$. If $|x'|$ is $\Omega(n^c)$, the run time needed for solving $x'$ will be $O(2^{\sqrt{\Omega(n^c)}})$ which can be asymptotically greater than $2^{\sqrt{n}}$. Thus the given statement is not correct.

4. Use the following algorithm, $\text{Size}(\text{Graph } G)$ -

\[
\text{for } i := |V| \text{ to } 0 \text{ do }
\]
\[
\text{if } \text{CLQ}(i) = \text{yes} \text{ then }
\]
\[
\text{output } i \\
\text{quit}
\]
\[
\text{end}
\]

Note that we increase the runtime of the CLQ algorithm, by a factor of $|V|$, yet maintaining it polynomial.