1. Let $M$ be the adjacency matrix and let $|V| = n$. Each of the $n^2$ processors is assigned one entry of $M$. These $n^2$ processors then compute the Boolean AND of the $n^2$ bits of $M$ in $O(1)$ time.

2. Make use of an array $b[1 : n]$. The $n$ processors fill $b[1 : n]$ as follows: $b[i] = i$ if $a[i] = n$ and $b[i] = 0$ otherwise, for $1 \leq i \leq n$. This can be done in 1 time unit. The $n$ processors then collectively find and output the maximum of $b[1], b[2], \ldots, b[n]$.

3. Let the input be $X = k_1, k_2, \ldots, k_n$.

**Fact 1.** If there are $n$ elements, we can solve the prefix minima problem in $O(1)$ time using $n^3$ processors.

Here is an algorithm: Assign $n^2$ processors to each prefix of $X$. The $n^2$ processors associated with any prefix can compute the minimum of the prefix in $O(1)$ time.

**Fact 2.** Given $n$ processors, we can solve the prefix minima problem in $O(\log \log n)$ time.

Here is an algorithm: Partition $X$ into $n^{1/3}$ parts where there are $n^{2/3}$ elements in each part. Assign $n^{2/3}$ processors to each part and solve the prefix minima problem for each part recursively and in parallel. If $T(n)$ is the time needed to solve the prefix minima problem on $n$ elements using $n$ processors, then this step takes $T(n^{2/3})$ time. Let $q_1, q_2, \ldots, q_{n^{1/3}}$ be the group minima (i.e., $q_i$ is the minimum of all the elements in part $i$, for $1 \leq i \leq n^{1/3}$). Solve the prefix minima problem on the sequence $q_1, q_2, \ldots, q_{n^{1/3}}$ using all the $n$ processors. This can be done in $O(1)$ time (using Fact 1). Let $r_1, r_2, \ldots, r_{n^{1/3}}$ be the result. Now we update the prefix values in each part using this sequence. In particular, if $w$ is any element in part $i$ it will be updated as the minimum of $w$ and $r_{i-1}$ (for $2 \leq i \leq n^{1/3}$). This takes $O(1)$ time.

As a result we infer that $T(n) = T(n^{2/3}) + O(1)$ that solves to $O(\log \log n)$.

Now we are ready to prove the main result. Assume that we have $P = \frac{n}{\log\log n}$ processors. We partition the input into $\frac{n}{\log\log n}$ groups where each group has $\log\log n$ elements. Assigning one processor per group we solve the prefix minima problem for each group in parallel. This takes $O(\log\log n)$ time. Let $q_1, q_2, \ldots, q_P$ be the group minima. Using all the $\frac{n}{\log\log n}$ processors compute the prefix minima of $q_1, q_2, \ldots, q_P$. Let the result be $r_1, r_2, \ldots, r_P$. We can obtain this sequence in $O(\log\log n)$ time (as per Fact 2). Now update the elements in each group. In particular, if $w$ is any element in part $i$ it will be updated as the minimum of $w$ and $r_{i-1}$ (for $2 \leq i \leq P$). This takes $O(\log\log n)$ time.

Thus the entire algorithm takes $O(\log\log n)$ time.

4. We first sort $B$ in $\tilde{O}(\log m)$ time using $m$ CRCW PRAM processors. Let the sorted form of $B$ be $k_1, k_2, \ldots, k_m$. We then let $m$ of the processors fill an array $C[1 : m]$ all with zeros. Followed by this, we assign one processor per key of $A$. Each of these
processors, in parallel, performs a binary search in \( B \) to check if its key is in \( B \). As a result of this parallel binary search, these processors modify \( C[1 : m] \) such that \( C[i] = 1 \) if \( k_i \) is in both \( A \) and \( B \) (and \( C[i] = 0 \), otherwise). Finally \( m \) of the processors perform a prefix computation on \( C[1 : m] \) to write all the elements of \( A \cap B \) in successive cells of the common memory.

Sorting of \( B \) can be done in \( \tilde{O}(\log m) \) time using \( m \) processors. \( C[1 : m] \) can be initialized to zeros in \( O(1) \) time. Parallel binary search takes \( O(\log m) \) time using \( n \) processors. The final prefix computation takes \( O(\log m) \) time using \( m \) processors.

5. We make use of radix sort where we sort one bit at a time starting from the LSBs of the keys. Sorting of \( n \) bits can be done using a prefix computation in \( O(\log n) \) time using \( \frac{n}{\log n} \) CREW PRAM processors (as was shown in class). Thus the entire algorithm takes \( O(\log n) \) time.

6. Let \( X = k_1, k_2, \ldots, k_n \) be the given input sequence. We partition the input into two equal parts \( X_1 \) and \( X_2 \) where \( X_1 = k_1, k_2, \ldots, k_{n/2} \) and \( X_2 = k_{n/2+1}, k_{n/2+2}, \ldots, k_n \). Assign \( n/2 \) processors to recursively sort \( X_1 \); Assign the other \( n/2 \) processors to recursively sort \( X_2 \). Once \( X_1 \) and \( X_2 \) are sorted, merge them in \( O(\log \log n) \) time using \( n \) processors. Let \( T(n) \) be the time taken to sort \( n \) elements using \( n \) processors. Then, \( T(n) = T(n/2) + O(\log \log n) \) which solves to \( O(\log n \log \log n) \).