1. Partition $X$ into $p$ equal sized parts. The first part will have $X[1], X[2], \ldots, X[n/p]$; the second part will have $X[(n/p) + 1], X[(n/p) + 2], \ldots, X[2n/p]$; and so on. Processor 1 checks if $x \in [X[1], X[n/p]]$; At the same time processor 2 checks if $x \in [X[(n/p) + 1], X[2n/p]]$; and so on. At the end of this checking at most one of the parts will survive, i.e., the value of $x$ will be within the boundary values of this part. We repeat this step until we either find $x$ or realize that $x$ is not in $X$. If $x$ is not in any of the parts, then we can infer that $x$ is not in $X$.

After each of the above checking steps, the size of the input decreases by a factor of $p$. Therefore, the run time is $O\left(\frac{\log n}{\log p}\right)$.

2. A memory cell $Result$ is initially set to zero. There are $\left(\begin{array}{c} n \\ 3 \end{array}\right)$ subsets of nodes of size 3 each. Assign one processor per subset. Each processor then checks if its subset forms a triangle or not in $O(1)$ time. Those processors that have a triangle will then try to write a one in $Result$. At the end of this write step, we have the correct answer in $Result$.

3. One possible algorithm is to randomly pick $k$ elements, find and output the maximum of these $k$ elements. The probability of an incorrect answer is $\leq (0.9)^k$. We want this probability to be $\leq n^{-\alpha}$. This implies that $k \geq \frac{\alpha \log n}{\log(10/9)}$.

How do we implement the above algorithm in parallel? Assume that we have $k$ processors. Then, the random sample can be picked in one unit of time. Using the slow-down lemma, this step can also be completed in $O(\log \log n)$ time given $\frac{\log n}{\log \log n}$ processors. Finding the maximum of $k$ elements can be done in $O(\log k)$ time using $\frac{k}{\log k}$ CREW PRAM processors. Using the slow-down lemma, the same can be done in $O(\log \log n)$ time using $\frac{\log n}{\log \log n}$ processors.

   
   1) Set $A[i] = 1$ if $X[i] \neq X[i + 1]$; otherwise set $A[i] = 0$ (for $1 \leq i \leq n$). If $A[i] = 1$, this means that $X[i]$ is the end of a run. (A run is nothing but a maximal subsequence of $X$ of equal values). We always set $A[n] = 1$.
   
   
   3) If $A[i] = 1$, then write $i$ in $C[B[i]]$, for $1 \leq i \leq n$.
   
   4) Set $C[i] = C[i] - C[i - 1]$, for $2 \leq i \leq k$. At the end of this step, $C$ stores the lengths of the $k$ runs.
   
   5) Find the maximum $M$ of $C[1], C[2], \ldots, C[k]$. $M$ is the number of occurrences of any mode of $X$.
   
   6) If $C[i] = M$, then the $i$th run corresponds to a mode (for $1 \leq i \leq k$). We output $X[i]$ if $A[i] = 1$ and $C[B[i]] = M$ (where $1 \leq i \leq n$). There could be more than one modes. We can output all the modes using an additional prefix computation.
5. The algorithm runs in phases. In each phase we eliminate a constant fraction of the
input keys that cannot be the element of interest. When the number of remaining keys
is \( \leq \sqrt{n} \), one of the processors performs an appropriate selection and outputs the right
element.

To begin with all the keys are \textit{alive}. In any phase of the algorithm let \( N \) stand for
the number of alive keys at the beginning of the phase. At the beginning of the first
phase, \( N = n \).

Consider a phase where the number of alive keys is \( N \) at the beginning of the phase. Let
\( Y \) be the collection of alive keys. We employ \( \sqrt{N} \) processors in this phase. Partition
the \( N \) keys into \( \sqrt{N} \) parts with \( \sqrt{N} \) keys in each part. Each processor is assigned
a part. Each processor in parallel finds the median of its keys in \( O(\sqrt{N}) \) time. Let
\( M_1, M_2, \ldots, M_{\sqrt{N}} \) be these group medians. One of the processors finds the median \( M \)
of these \( \sqrt{N} \) group medians. This will take \( O(\sqrt{N}) \) time. Now partition \( Y \) into \( Y_1 \)
and \( Y_2 \), where \( Y_1 = \{ q \in Y | q < M \} \) and \( Y_2 = \{ q \in Y | q > M \} \). There are 3 cases to

\textbf{Analysis:} Steps 1, 3, and 4 can be done in \( O(1) \) time using \( n \) CREW PRAM
processors. Using the slow-down lemma, they can also be completed in \( O(\log n) \)
time using \( \frac{n}{\log n} \) processors. Step 2 takes \( O(\log n) \) time using \( \frac{n}{\log n} \) processors. In
step 5, we have to find the maximum of \( n \) integers in the range \([1, n]\). This can
be done using a prefix computation that takes \( O(\log n) \) time using \( \frac{n}{\log n} \) CREW
PRAM processors. In Step 6 also we have to do a prefix computation.

In summary, the entire algorithm takes \( O(\log n) \) time using \( \frac{n}{\log n} \) CREW PRAM
processors.

\textbullet \textbf{Solution 2:} By a segment of \( X \) we mean a maximal subsequence of \( X \) of equal
values. For instance if \( X = 1.2, 1.2, 3, 3, 3, 3, 3, 3, 5.5, 6.1, 7.4, 7.4, 7.4, 8, 8 \), then there
are six segments in \( X \). They are: 1.2, 1.2; 3, 3, 3, 3; 5.5; 6.1; 7.4, 7.4, 7.4; and 8, 8.
1) We initialize an array \( A[1 : n] \) with all 1’s.
2) We also initialize another array \( B[1 : n] \) as follows. The beginning cell of each
segment will be 1 and the other entries will be zeros. For the above example, \( B \)
will have: 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0.
3) We perform a prefix sums computation in \( A \) for every segment of \( X \). For
the above example, the prefix sums will be 1, 2, 1, 2, 3, 4, 1, 1, 1, 2, 3, 1, 2. Call this
sequence \( C \).
4) We find the maximum of the sequence \( C \). If \( C[i] \) is a maximum, then, \( X[i] \) is
a mode.

\textbf{Analysis:} Steps 1 and 2 can be done in \( O(1) \) time using \( n \) processors. Using
the slow-down lemma, they can also be completed in \( O(\log n) \) time using \( \frac{n}{\log n} \)
processors. Step 3 can be completed formulating it as a prefix computation with
an operation \( \oplus \) where \( \oplus \) is defined as follows: \( (a, b) \oplus (a', b') = (a', 1) \) if \( b' = 1; (a, 0) \oplus (a', 0) = (a + a', 0) \); and \( (a, 1) \oplus (a', 0) = (a + a', 0) \). In this operation, \( a \) and
\( a' \) are elements of \( A \). Also, \( b \) and \( b' \) are the corresponding elements of \( B \). Step 3
thus can be done in \( O(\log n) \) time using \( \frac{n}{\log n} \) CREW PRAM processors. Finding
the maximum of \( C \) can also be done using a prefix maximum computation in
\( O(\log n) \) time using \( \frac{n}{\log n} \) CREW PRAM processors.
consider: **Case 1:** If $|Y_1| = i - 1$, $M$ is the element of interest. In this case, we output $M$ and quit. **Case 2:** If $|Y_1| \geq i$, $Y_1$ will constitute the alive keys for the next phase. **Case 3:** If the above two cases do not hold, $Y_2$ will constitute the collection of alive keys for the next phase. In this case we set $i := i - |Y_1| - 1$. In cases 2 and 3 we can perform the partitions using a prefix computation that can be done in $O(\sqrt{N})$ time using $\sqrt{N}$ processors.

It is easy to see that $|Y_1| \geq \frac{N}{4}$ and $|Y_2| \geq \frac{N}{4}$. As a result, it follows that the number of alive keys at the end of this phase is $\leq \frac{3}{4}N$.

Thus we infer that the run time of the algorithm is $O\left(\sqrt{N} + \sqrt{(3/4)N} + \sqrt{(3/4)^2N} + \ldots \right) = O(\sqrt{N})$.

6. For this problem, we will utilize the fact that we can merge two sorted sequences of length $n$ each in $O(\log \log n)$ time using $n$ CREW PRAM processors. Partition the input into $\log n$ parts where there are $\frac{n}{\log n}$ keys in each part. Each processor is assigned one part. To begin with each processor in parallel sorts its part. This takes $O(n)$ time. Now we have $\log n$ sorted sequences to merge. Consider a complete binary tree where each leaf has one of these sorted sequences. We start from the leaves and proceed toward the root. At each internal node we merge the two children sequences. When the root merges the two children sequences we get the desired result.

**Analysis:** Let the level of the leaves be 0, the level of their parents be 1, and so on. At each node of level $(i + 1)$, we have to merge two sorted sequences of length $2^i \frac{n}{\log n}$ each. This can be done in $O\left(\log \log \left(2^i \frac{n}{\log n}\right) \right) = O(\log \log n)$ time using $\frac{n}{\log n}$ processors. Using the slow-down lemma, we can also do the same in $O\left(\frac{n}{\log n} \log \log n \right)$ time using $2^i$ processors.

Note that there are $\log \log n$ levels. Thus the total time spent in the $\log \log n$ levels of merges is $O\left(\frac{n}{\log n} (\log \log n)^2 \right) = O(n)$.

Put together, the run time of the entire algorithm is $O(n)$. 