1. Associate a counter with each node in the tree. It gives the number of nodes in the subtree rooted at that node. Modify the \textit{PostOrder} tree traversal algorithm as below:

\begin{verbatim}
PostOrder(t)
{
    if t is a leaf node {
        t → count = 1
        return 1
    }
    else {
        if (t → lchild ≠ 0)
            left = PostOrder(t → lchild) else left = 0;
        if (t → rchild ≠ 0)
            right = PostOrder(t → rchild) else right = 0;
        t → count = left + right + 1;
        return (t → count)
    }
}
\end{verbatim}

The run time of the algorithm is $O(n)$ since we only spend $O(1)$ time per node in the tree.

2. Partition $X$ into $\sqrt{n}$ equal sized parts. The first part will have $X[1], X[2], \ldots, X[\sqrt{n}]$; the second part will have $X[\sqrt{n} + 1], X[\sqrt{n} + 2], \ldots, X[2\sqrt{n}]$; and so on. Processor 1 checks if $x \in [X[1], X[\sqrt{n}]]$; At the same time processor 2 checks if $x \in [X[\sqrt{n} + 1], X[2\sqrt{n}]]$; 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 now use all the $\sqrt{n}$ processors to check if $x$ equals one of the elements in the surviving part. The total run time is $O(1)$.

3. A memory cell $\text{Result}$ is initially set to zero. There are $\left(\frac{n}{3}\right)$ 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 $\text{Result}$. At the end of this write step, we have the correct answer in $\text{Result}$.

4. 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 $\leq (1/2)^k$. We want this probability to be $\leq n^{-\alpha}$. This implies that $k \geq \alpha \log n$.

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.
5. If \( n \) is the size of the instance of \( \pi_1 \), we can translate this instance into an instance of \( \pi_2 \) in \( n^3 \) time. This implies that the size \( m \) of the corresponding instance of \( \pi_2 \) cannot be more than \( n^3 \). Since the algorithm for \( \pi_2 \) takes \( m^{\log m} \) time, this run time is no more than \((n^3)^{3\log n}\). Thus, using the given reduction, \( \pi_1 \) can be solved in time no more than \( n^3 + n^{9\log n} = O(n^{9\log n}) \neq O(n^{\log n}) \). Therefore we may not be able to solve \( \pi_1 \) in time \( O(n^{\log n}) \).

6. Let the input set be \( S = \{k_1, k_2, \ldots, k_n\} \). Here is an algorithm that outputs a subset \( S' \) whose sum is \( M \).

\[
S' := S \\
\text{for } i := 1 \text{ to } n \text{ do} \\
\quad \text{if SUBSUM}(S' - \{k_i\}) \text{ then } S' := S' - \{k_i\} \\
\text{Output } S'
\]

The above algorithm runs in polynomial time since we call \text{SUBSUM} only \( n \) times and \text{SUBSUM} runs in polynomial time.