We are given k leftist trees and the size of the i-th tree is s_i. Let $n = \sum_{i=1}^{n} s_i$. Now we show a method to meld them using time $O(k + k\log(n/k))$.

Without loss of generality, let's assume that $k = 2^m$ for some m. The melding works in a same way like merging sort. First, *arbitrarily* make $k/2$ pairs of leftist trees and meld each pair. The running time for this step is $\sum_{i=1}^{k} \log(s_i) = \log(\prod_{i=1}^{k} s_i) \le \log\left((\sum_{i=1}^{k} s_i)/k\right)^k = k\log(n/k)$. Now, we have $k/2$ bigger leftist trees. Again, we arbitrarily make $k/4$ pairs and meld each of them. The running time for this step is bounded by $\frac{k}{2}\log\left(\frac{2n}{k}\right)$ (we just need to replace k by $k/2$ in the above formula). We repeat this process m times to meld them to a single leftist tree. Thus, the total running time for melding k leftist trees is

$$\sum_{i=0}^{m-1} \frac{k}{2^i} \log\left(\frac{2^i n}{k}\right) = \sum_{i=0}^{m-1} \frac{k}{2^i}(i + \log(n/k)) = k\sum_{i=0}^{m-1} \frac{i}{2^i} + k\log(n/k)\sum_{i=1}^{m-1} \frac{1}{2^i} = O(k + k\log(n/k)).$$

Please compare the above problem with the following sorting problem: given k sorted arrays, and the length the i-th array is $\log(s_i)$, sort these k arrays.

We might use a heap of size k to sort all of them. The heap maintains the first element in each array. Each time we delete the minimum of the heap and insert a new element, which costs $O(\log k)$. Thus, the total running time is $\log k \cdot \sum_{i=1}^{k} \log(s_i) \le k\log k\log(n/k)$.

The difference between these two problems is that building heap is cheaper than sorting (unlike search tree, elements in heaps are not sorted). Recall that building a heap of size n costs $O(n)$ while sorting n elements costs $O(n\log n)$. Another intuitive explanation is that after melding two leftist trees of equal size s the length of the rightmost path of the new leftist tree is not $2 \cdot \log s$, but roughly $1 + \log s$.