Let ℓ denote the number of ladders.
1. Binary search, select the largest ℓ numbers in the range in linear time, and use ladders for them. The running time satisfies the recurrence $T(n) = T(\frac{n}{2}) + O(n + \ell)$, which gives $T(n) = O(n + \ell \log n)$.
2. Add numbers from left to right, use heap to maintain the ℓ largest numbers. $O(n \log n)$.
3. We combine the two algorithms. Notice that in the second algorithm, if the answer lies in an interval of length r, then we can first build the heap in $O(n)$ time, then perform $O(r)$ insertion/deletions in $O(r \log n)$ time. Set $r = \frac{n}{\log n}$, we only need to perform $\log \log n$ binary searches, and the total running time is $O(n + \ell \log \log n) + O(n + \frac{n}{\log n} \cdot \log n) = O(n \log \log n)$.
4. If the answer lies in an interval of length r, then we will always select the $[1, \ell - r]$-th largest numbers before the interval, and always not select the (ℓ, ∞)-th largest numbers before the interval, so we only need to consider $O(r)$ numbers in the current round. The running time satisfies the recurrence $T(n) = T(\frac{n}{2}) + O(n)$, which gives $T(n) = O(n)$.

References