Proof of Correctness for `merge_based`

Lemma.
In a sorted array, all occurrences of the same value appear next to each other in one contiguous block. Therefore, if an element appears exactly once in the array, it must be the unique value we are looking for.

Let the current subarray under consideration be $A[\text{start}, ..., \text{end}]$. The size of this subarray is defined as n = end - start + 1.

Proof by Induction:

Base Case ($n = 1$). If there is only one element (`start` = `end`), the algorithm returns `A[start]`. Since there is exactly one unique value in the array, this element must be it. Thus, the algorithm works correctly for $n = 1$.

Inductive Hypothesis. Assume that for all sub arrays of size $n = k$ or smaller, `merge_based(A, start, end)` correctly finds and returns the unique element if it is in that subarray, and returns `None` otherwise.

Inductive Step ($n = k + 1$). Consider a subarray $A[\text{start}, ..., \text{end}]$ of size $k + 1$.

$$\text{mid} = \left\lceil \frac{\text{start} + \text{end}}{2} \right\rceil$$

and $x = A[\text{mid}]$. It then searches left and right to find how far x repeats:

- i = the left most index such that $A[i] = x$
- j = the right most index such that $A[j] = x$

All elements in the interval $[i, j]$ are equal to x.

Case 1: $i = j$
This means x appears only once. By the lemma, x is the unique element, so returning $A[i]$ is correct.

Case 2: $i < j$
This means x appears more than once, so it cannot be unique. Because the array is sorted, all copies of x are in $[i, j]$, and the unique element must be in the left sub array $A[\text{start}, ..., i - 1]$ or the right subarray $A[j + 1, ..., \text{end}]$.
The algorithm first searches the left side. If the unique element is there, then by the inductive hypothesis, the recursive call correctly finds it. If not, it searches the right side, which will correctly return the unique element. Therefore, the algorithm is correct for arrays of size $n = k + 1$.

Conclusion. By the Principle of Mathematical Induction, `merge_based(A, start, end)` correctly returns the unique element for all $n \geq 1$.

Time Complexity. Let $T(n)$ be the running time for a subarray of size n. At each step, the algorithm searches through the consecutive equal elements around the middle. After that, it makes recursive calls on smaller subarrays that don't include the consecutive elements that equal the middle value. Since every element is only looked at once overall, the total work across all recursive calls adds up to about n operations. Therefore, the time complexity is $O(n)$. The space complexity is $O(\log n)$ due to recursion depth.

Base Case:	$n = 1 \Rightarrow$ Correct.
Inductive Step:	$n = k + 1 \Rightarrow$ Correct by hypothesis.
Therefore:	Algorithm is correct for all n.
Time Complexity:	$O(n)$
Space Complexity:	$O(\log n)$