Algorithm Description

Assume that $n > 0$.

Base Cases: Return the element ($n = 1$) or the maximum of the two elements by comparing the value of each element ($n = 2$).

Divide step: Compute the index of the middle element by floor division. If the middle element is lower than the first element of the subarray, recurse on the left half of the subarray (including the middle element). Otherwise, recurse on the right half of the subarray (including the middle element).

Merge step: Return the recursive result.

Proof of Correctness

We prove that the algorithm is correct by induction.

Base Cases: When $n = 1$, the algorithm returns the element. Since the largest element in an array of size 1 is simply the element, the algorithm is correct. When $n = 2$, the algorithm returns the maximum of the two elements, which is correct.

Inductive Case: When ($n > 2$), suppose the algorithm is correct for arrays with size less than n.

We observe that x is the number of times the elements inside a sorted array is moved to the left. For example, an array with $x = 0$ is a sorted array and an array with $x = 1$ has the second smallest number as the first element, the third smallest number as the second element, and so on. From this observation, we note that the first $n - x \mod n$ elements are sorted, and the rest of the elements are also sorted. Hence, we conclude that there is always at least one sorted array to the left and/or right of the middle element.

If the middle element is greater or equal to the starting element, then we know that the left half is sorted. This means that the middle element is the biggest in the left half of the array. Hence, we only need to recurse on the right half of the array (including the middle element).

Consider the case where the middle element is less than the starting element. We observe that for a bigger number to be on the left of the middle element, $x \mod n > \frac{n}{2}$. This means that the $(n-1)$-th smallest number will be shifted enough times to the left so that it is always in the left half of the array. Hence, we only need to recurse on the left half of the array (including the middle element).

By the inductive hypothesis, the algorithm is correct for arrays with size n.

Time Complexity

The algorithm only searches one-half of the array at each recursive call. The divide and merge step takes $O(1)$ time. The recurrence relation is then $T(n) = T(n/2) + O(1)$. Using Master's Theorem, where $a = 1, b = 2, f(n) = O(1)$ and $f(n) = \Theta(n^{\log_b a} \log^k n)$ with $k = 0$, $T(n) = O(\log n)$.