```
findMin(A)
    // since we know that A[1...j] is increasing, A[j+1...n] is increasing
    // A[j] > A[j+1] and A[n] < A[1].
    // We know that every value of the first part (A[1...j]) must be greater than
    // every other value in the second part; and the minimum value in the vector
    // must be A[j+1]. Therefore, the algorithm needs to find j+1 where A[j] > A[j+1]

    if (n < some small constant) then
        use brute force to find (j+1)
        return A[j+1]

    p <- n/2 // let the pivot be the midpoint of the vector (assume integer division)
    if A[1] > A[p] then
        // the minimum must be in the part of vector between index 1 and p,
        // so we recursively call the algorithm with the sub-vector
        // there is no need to search the other half of the vector
        // since there is only one minimum value
        L <- sub-vector from A[1] to A[p]
        return findMin(L)
    else
        // same reasoning, this time we only need the sub-vector from p+1 to n
        G <- sub-vector from A[p+1] to A[n]
        return findMin(G)

// The recurrence relation of this algorithm is T(n) = T(n/2) + 1
// and according to the Master Theorem, the overall time complexity is Θ(log n)
```