Sample question and solution:

Suppose you are given a sorted array of n distinct numbers that has been rotated k steps, for some unknown integer k between 1 and $n-1$. That is, you are given an array $A[1..n]$ such that some prefix $A[1..k]$ is sorted in increasing order, the corresponding suffix $A[k+1..n]$ is sorted in increasing order, and $A[n] < A[1]$.

For example, you might be given the following 16-element array where $k = 10$

9	13	16	18	19	23	28	31	37	42	1	3	4	5	7	8

(a) Describe and analyze an algorithm to compute the unknown integer k

(b) Describe and analyze an algorithm to determine if the given array contains a given number x.

Sample Solution:

(a) If we scan the array from left to right, we can find k-th element because it's the only element that is larger than its right neighbor. Such naive algorithm takes $O(n)$ time. We can do better by divide and conquer. We divide in the middle of the array, and find that one half must be sorted, the other half must be rotated sorted, unless the middle happens to be the k-th element. **The k-th element must be on the rotated sorted half**.

```
FindK(A[i...j]):
    m ← floor((i + j) / 2)
    if A[i] > A[m]
        return FindK(A[i...m)
    if A[m + 1] > A[j]
        return FindK(A[m + 1...j])
    return m
```

Denote the time complexity of FindK on a size n array by $T(n)$. From the algorithm, it recurses into at most one half of the array, hence the recurrence:

$$T(n) = T(n/2) + 1$$

which gives $T(n) = O(\log n)$.

(b) Scanning the array can find whether x is an element in $O(n)$ time, but this naive algorithm does not exploit any structure of the array. On the other hand, we know that bisection on an sorted array can find x in logrithmic time $O(\log n)$. First, we invoke the FindK algorithm from (a) to find k. This takes $O(\log n)$ time. With k, we can basically do bisection (every index gets "rotated" to the right by k positions, with wrap-around)

```
FindX(A[1...n], x, i, j):
    if i > j return false
    if i = j
        if A[(i + k − 1)%n + 1] = x
            return true
        else
            return false
    m ← floor((i + j) / 2)
    if A[(m + k − 1)%n + 1] > x
        return FindX(A[1...n], x, i, m)
    else if A[(m + k − 1)%n + 1] < x
        return FindX(A[1...n], x, m + 1, j)
    else
         return true
```

The time complexity is the same as bisection, which is $O(\log n)$.