1. In lecture, we saw an efficient algorithm for counting the number of inversions in an array. Suppose we define a significant inversion to be where $i < j$ but $A[i] > 2 \cdot A[j]$. Give an $O(n \log n)$ algorithm to count the number of significant inversions in an array.

2. Suppose you are given a sorted array of distinct integers. Give an efficient algorithm to determine if there is an element $i$ such that $A[i] = i$.

3. We’ve all heard of square root, but what about rectangle root? I define the $k$-rectangle root $r$ of $n$ to be:

$$ (r)(r + k) = n $$

**Example 1:** The 3-rectangle root of 130 is 10 because $10(10 + 3) = 130$.

**Example 2:** The 3-rectangle root of 54 is 6 because $6(6 + 3) = 54$

Write or describe an $O(\log n)$ time algorithm to find the $\lfloor r \rfloor$, the floored $k$-rectangle root of $n$, given that $\lfloor r \rfloor, k, n$ are all natural numbers. Do not use a square root operation as part of your solution.

4. Suppose you are given an array $A$ of distinct numbers that is circularly sorted. That is, it begins at some value, and increases until it reaches the maximum value in the array. The next number is the minimum, and values increase from there until the last value, which will be less than the first value.

Given such an array $A$, give an efficient algorithm to find the minimum value in the array. State the running time of your algorithm and explain its correctness.

For example, if your input array $A$ is:

| 10 | 13 | 17 | 20 | 1 | 2 | 3 | 4 | 5 | 6 |

Then your return value should be 1.

Note: A brute-force algorithm can solve this in time $O(n)$. For credit, your solution must have a running time strictly better than this.

5. You are given an $n \times n$ matrix $A$ where every row is in sorted order and every column is in sorted order.

Design an efficient (better than $\Omega(n^2)$) divide-and-conquer algorithm to search for an element $x$ in the matrix, and analyze your algorithm’s runtime complexity with respect to $n$. 