1. Consider **stooge sort**, an inefficient but correct recursive algorithm for sorting an array:

```
function stoogesort(A[], lo, hi) {
    if (A[lo] > A[hi])
        swap(A[lo], A[hi]);
    if (hi - lo + 1 >= 3) {
        oneThird = floor((hi - lo + 1) / 3);
        stoogesort(A, lo, hi - oneThird);
        stoogesort(A, lo + oneThird, hi);
        stoogesort(A, lo, hi - oneThird);
    }
    return A;
}
```

   Use the Master Theorem to find the asymptotic runtime complexity of this algorithm.

2. You are given n keycards, each of which is coded with an account ID. Unfortunately, there is no efficient way to read the IDs off the keycards. In fact, the only operation you can perform with them is to check whether two keycards match each other or not (which takes $\Theta(1)$ time).

   A keycard is a **majority card** if it is part of a set of at least $n/2$ that all match each other. Design an $\mathcal{O}(n \log n)$ algorithm that can determine whether there is a majority card and return one if so.

3. Recall that a **complete binary tree** is a binary tree where every layer is full, except possibly the bottom layer. Suppose you are given a complete binary tree where every vertex has a numerical value associated with it. A vertex is a **local minimum** if its value is less than or equal to those of all its neighbors.

   Design an $\mathcal{O}(\log n)$ algorithm that finds a local minimum in a complete binary tree with n vertices. (Note that this tree is not necessarily a binary *search* tree!)