Before Friday's 6:00 PM lab, I encourage you to try each of these, either alone or with friends in the class. I will present solutions to these in lab, along with tips for how to solve problems like them.

1. Consider the problem of finding a *local minimum* in an array. The value $A[i]$ is a local minimum if - and only if - $A[i-1] \geq A[i] \leq A[i+1]$. Give an efficient **divide and conquer** algorithm to find a local minimum in an array A in which $A[1] = A[n] = \infty$. State the runtime of your algorithm, both as a recurrence relation and in asymptotic notation and briefly (1-2 sentences is sufficient) explain the correctness of your algorithm.

   For example, if your input array A is:

∞	10	23	17	20	1	2	3	4	-3	5	∞

   then any of the grayed cells could be returned as a valid answer (you need only to find *one* local minimum, not all of them).

   A solution with running time $\Omega(n)$ will receive no credit for this problem.

2. The Sorting Hats problem is as follows. I have a collection of n hats and n mannequins (statues that resemble a human and are used to model clothing and accessories). My goal is to put one hat on each mannequin head. Each hat is a distinct size, and no two heads are the same size. For each hat, there is exactly one head that the hat fits perfectly. I cannot directly compare two hats or two heads, nor can I inspect a single hat or a single head to determine its size.

   I can, however, try to place a hat on a head. If I do so, I find out one of the following pieces of information:

   - The hat fits the head perfectly
   - The hat is too small for the head
   - The hat is too big for the head

   We can then remove the hat from the head if we so choose.

   The goal of the Sorting Hat problem is to match each head with the hat that fits it perfectly while placing as few hats on heads as possible. Give an approach to do so. If there are n hats and n heads, express the number of times your approach will place a hat on a head in $\mathcal{O}$-notation and justify why that many such operations will happen. Clearly indicate if you are measuring *worst-case* or *expected case* (either is okay, but be sure to clearly indicate it).

   Good credit is available for any reasonably-efficient solution, clearly described, with more credit for more efficient solutions.

3. 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.