Problem (Life Imitates Art, TAs Grade Homework). Suppose you are a TA for `CS381`[1]. You and your fellow TAs (there are $n \in \mathbb{N}$ TAs total) are very stressed because there is a backlog of $k \in \mathbb{N}$ homeworks that need to be graded. To grade these k homeworks, Professor Kent has split the TAs up into k groups to handle each homework. Crucially, each group has one leader, who handles the most work (delegation, rubric construction, etc.).

Unfortunately, he used ChatGPT, so some TAs are in more than one group. He decided that the best we could do was to enforce a 'fairness requirement', where no single person leads too many groups.

Formally, there are k homeworks and n TAs. Homework i for $i \in [k]$ is graded by TA group $S_i \subseteq [n]$ with some leader $j \in S_i$. The fairness requirement is that any TA $j \in [n]$ who is in groups $S_{j_1}, \ldots, S_{j_m}$, does not lead more than

$$\left\lceil \frac{1}{|S_{j_1}|} + \frac{1}{|S_{j_2}|} + \cdots + \frac{1}{|S_{j_m}|} \right\rceil = \left\lceil \sum_{j \text{ s.t. } i \in S_j} \frac{1}{|S_j|} \right\rceil$$

of those groups. Given the k groups $S_1, S_2, \ldots, S_k \subseteq [n]$ as input, give a polynomial time algorithm (the faster the better) that *always* finds a leader assignment following the fairness requirement, or show that a poly-time algorithm for this problem implies a poly-time algorithm for SAT.

[1] For me, this is very easy.

Problem Exercise 22.1**, part** 1**.** Recall the array-backed lists from Section 22.3, where we showed how to support `append` in $O(1)$ amortized time via the "doubling trick". We would like to support a `remove` operation with similar amortized time. Consider the following approach:

> After removing the element from the array and decreasing the size counter, if the size of the list is now less than half of the capacity of the array, allocate a new array of half the capacity and copy the elements of the list over.

Show that this does not achieve $O(1)$ amortized time. That is, give a sequence of n `append`/`remove` operations on which the data structure would take $\Omega(n^2)$ time.