Problem 1: On input an array $A$ of $n$ elements, each of which is an integer in $[0..n^2]$, describe a simple method for sorting $A$ in $O(n)$ time.

Solution: The idea is to write a number in $[0..n^2]$ in $n$-ary notation. In other words, if $0 \leq a \leq n^2$, then $a$ can be uniquely encoded as $(a_0, a_1)$ where $0 \leq a_0 \leq n$, and $a = a_0 + n \cdot a_1$.

You can sort the elements of $A$ by sorting them “lexicographically” on the two components (first over $a_1$ and then over $a_0$ to break the ties). By using COUNTING-SORT on each component, you have a $O(n)$ algorithm.

Problem 2: You are given $n$ samples of a chemical compound. While they look identical, some of them have in fact been contaminated. You have a testing machine that given two samples can detect if they are the same or not. You also know that most of the samples (a majority of them) are identical. Find one of those identical samples making no more than $n$ tests with your machine (a.k.a. comparisons).

Solution: Order the $n$ samples arbitrarily as $s_1, s_2, \ldots, s_n$. Then for each odd $i$, compare $s_i$ with $s_{i+1}$. If they are equal, then keep just one of the two samples. If they are different, drop both of them. Repeat the above process until you are left with just one element, and output that element as a member of the majority.

First, let’s prove that this algorithm terminates with less than $n$ comparisons. Note indeed that at each step the number of samples is reduced by at least half, so the number of comparisons $T$ is

$$T \leq \frac{n}{2} + \frac{n}{4} + \ldots + 1 \leq n$$

We need to prove that the algorithm is correct. Let $m$ be the number of ”majority samples” (m-samples in the rest), so $m > n/2$. The ”nonmajority samples” (nm-samples) then are $n - m < n/2$.

Note that at the end of each iteration those samples will still be the majority of the samples left. Let’s prove it for the first iteration: the others will follow analogously. To see that, let

- $a$ be the number of pairs composed of a nm-sample and an m-sample: those are both dropped from the pool;
- $b$ the number of pairs composed by both m-samples. Of those 1 sample remains in the pool;
- $c = n/2 - a - b$ the number of pairs composed of both nm-samples. We do not know exactly what happens in this case (depends if the pair is of identical elements or not) but we know that at most $c/2$ elements remain in the pool

The total number of elements left in the pool is at most $\frac{c + b}{2}$, with $b/2$ m-samples. Note that $b > c$ since the number of m-samples is $m = a + 2b$, while the number of nm-samples is $a + 2c$. So the m-samples are still the majority after each iteration. This implies that the last element left must be an m-sample.

Problem 3: Consider a phone book – its entries have several keys: last name, first name, address, and telephone number. They are sorted first by last name, then by first name, and finally by address. Design a variation of radix sort for sorting the entries in a phone book, assuming that you have available a procedure that sorts records on a single key. The running time of your algorithm should be of the same order as the running time of the procedure that sorts records on a single key.

Solution: Let $A$ be an algorithm that sorts by key $k$. Then if the algorithm is stable it is sufficient to call it 3 times in reverse order of importance of the keys: first on the address, then on the first names, and finally on the last names. Remember that an algorithm is called ”stable” if it leaves items in the same order as they arrived in the input if there is a tie.

If $A$ is not stable then one needs to first sort by last name and keep track of the ties. Then sort just the ties by first name. And then again sort the ”double ties” by address.

Notice that in either case the cost of sorting the whole data is at most 3 times the running time of $A$. 