Code Jam to I/O for Women 2016 - Code Jam to I/O for Women

Analysis: Cody's Jams

With the regular and sale prices all jumbled together, how can you determine which prices are which? For the Small dataset, one approach is to consider all of the possibilities. Since there are only up to 8 prices, you can figure out all the different ways to choose half of them. Then, for each of those sets of half of the prices, you can assert that those prices are the sale prices, determine what the set of corresponding regular prices would be, and check the other half of the prices to see if they match that set. Eventually, you'll find the one set for which this works. This method won't work for the Large dataset, though, since it can have up to 2000 prices, and there may be far too many possible ways to choose half of them.

One possible strategy for the Large is to make deductions about the values. For example, a price of 9 cannot possibly be a regular price, because the sale price would be 6.75, which is not an integer. But sometimes the value alone won't give away what sort of price it is. For example, a 12 could be a regular price corresponding to a sale price of 9, or a sale price corresponding to a regular price of 16. There's no way to know without looking at other prices in the list, and you may need to look at other prices to classify *those* prices, and so on. It's possible to write code that does this recursively, but it's a headache. Is there a better way?

Let's call the smallest price in the list P_{min}. A key observation is that P_{min} *must* be a sale price. (Suppose that P_{min} were a regular price. Then the list would also have to include an even smaller 3/4 * P_{min} sale version of P_{min}... but then P_{min} wouldn't be the smallest price in the list anymore!) Then the regular price corresponding to P_{min} is just 4/3 * P_{min}, and you can find that price in the list. You can then remove both P_{min} and 4/3 * P_{min} from the list, and add P_{min} to your answer. Once you've done that, you can use the same strategy again: the smallest remaining price must be a sale price, and so on, until you've processed the entire list.

The approach above can be implemented in O($\mathbf{N}^2$) if you use linear search and removal, O(**N** log **N**) if you use a balanced tree structure to search and remove more efficiently (such as C++'s `set` or Java's `TreeSet`), and O(**N**) if you use chasing pointers (keep pointers to the last assigned price of each type, and only move those pointers forward when searching).