# Frisco Bytes fall 2018 How to

**A: Fish Tank**
We are given a set of points describing a convex polygon that represents the front of a fish tank. The back of the tank is identical to the front. We must locate the two points that represent the base and the two points that represent the top of the tank. Then we need a list of (x,y) points corresponding to the tank vertices in increasing y-order. This list comprises interleaved values from the left and right edges of the tank.

We can then iterate up this list calculating the area of the polygon so far, as we go up (using The Shoe Lace Algorithm), and hence the volume of the tank up to that point. The objective is to find the four points that span the surface level of the water, two on the left edge and two on the right edge, or we may get lucky if the surface of the water has height exactly equal to one or two of the y-values). If not we solve for the height of the water by a simple linear equation based on those four points.

To speed things up, use binary search to find the four points that span the height of the water.

**B: Order Up**
Declare a class, comparableString, for a String with a comparator function that uses the first two characters of the Strings. Store the strings in an array of these comparableStrings and call Java's Arrays.sort. It will use Mergesort b/c you have declared a comparator. Mergesort is a stable sorting algorithm.

**C: Don't Ever, Ever Gamble**
It's easy to compute the probability of an outcome within the range of [the sum of the smallest face-values, the sum of the largest face values].

For example, with two six-sided die we get sums of all of the possible two die faces as follows:

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 6 | 7 | 8 | 9 | 10 | 11 | 12 |

From this table count the occurrences of 1's, 2's, etc. to yield p(2) = 1/36, p(3) = 2/36, p(4) = 3/36, . . ., p(7) = 6/36, etc. But you don't have to compute this table - it's not too difficult to develop equations for these values based on the numbers of faces on the two die.

**D: Smallest Multiple**
You have to compute the LCM (lowest common multiple) of the input numbers. The LCM of two integers $a, b$ is

$$\frac{ab}{GCD(a,b)}$$

and $LCM(a, b, c) = LCM(LCM(a, b), c)$

Alternatively, list the prime factors of the component numbers and multiply together the factors with the largest exponents: LCM(27, 16, 7, 9, 12):
Prime factors: $(3^3), (2^4), (7), (3^2), (3 * 2^2)$:
LCM $= 2^4 * 3^3 * 7 = 16 * 27 * 7 = 3024$

The BigInteger package includes a GCD function and deals with the range of numbers implied.

**E: Prime Hopping**
Do a binary search on the earliest time to consolidate all of the frogs to see if they are all in the correct place,

Check the leftmost and rightmost frog. If they are in the same section, then all frogs in between will have to be as well.

To find what section a frog is in use binary search again.

**Solitaire Mark X**
Represent the board in bitwise notation where a 'o' is a 1 and '-' is a 0. Then use brute force by looking for two cases where a piece can be removed 'oo-' and '-oo'. The results of these two cases are '–o' and 'o–' respectively.

Since the board size is 23, you only needed to look at 21 positions. Once you can't make any moves, the number of pieces left is the number of bits, and the solution with the least number of bits is the solution.

**G: Mail Train**
A greedy solution works where, at each station, you eliminate those packages that are now at their correct destination, then unload everything else on the train, mix the new packages from that station with those unloaded, and load them in increasing order of package weight.