1. ***Losing counts.*** Remember our old friend DFA? "Hey, how's going? Not bad, not bad, you know, just staying at home and spending my day deciding strings and all. It's different considering everything is remote! Yesterday I was told to keep track of the different packages visiting the router. You know, I love those cute little fellas, but I keep lost count of them! They just, you know... come and go! Without my reliable RAM at work this is kind of tough. Anything that helps?" You suggested them to use the old portable calculator that was sitting in their drawers since high school. "Ah that might work. But... how am I supposed to keep the count of each type separate using a single calculator?" You scratched your head and reached out to your pocket, and found some coins (along with a piece of candy that you had no idea where it came from). How can you help your friend not to lose their (remote) job?

   Consider an incoming string w of length n. Every symbol in w has a *number* from 0 to k, signifying the *type*; and the plus-minus sign suggests whether to *add* or *subtract* from the count. For example, if the incoming string w is

   ```
   +2-7-1+8+2-8-1+8+2+8-4-5-9+0-4+5+2-3-5-3+6-0+2+8-7+4-7-1-3+5+2
   ```

   then the total count for $\langle 0, \ldots, 9 \rangle$ is $\langle 0, -3, 6, -3, -1, 0, 1, -3, 3, -1 \rangle$, respectively.

   For the machine model, you don't have general purpose memory but a single *counter*, that can store any number of length at most $O(\log n)$. (Feel free to assume the constant C in the big-O notation is as big as you wish, but always strictly smaller than to the number of different types of symbols.)

   Design and analyze a randomized algorithm to decide with high probability[1], after reading w in *one-single pass*, if the count ends up at *exactly zero* for each type (in other words, whether the number of symbols of each type with a positive sign + is the same as the ones with negative sign -).

2. ***Solving* SAT *by flipping coins.*** Consider the following very simple randomized algorithm:

   ```
   RANDOMkSAT($\phi$):
     input: $k$-CNF formula $\phi$
     choose a uniform random assignment $x = \langle x_1, \ldots, x_n \rangle$
     repeat for $n^2$ steps:
         return yes if $x$ satisfies $\phi$
         choose an arbitrary unsatisfied clause $C$ from $\phi$
         choose a variable $x_i$ in $C$ uniformly at random
         flip the variable $x_i$ in assignment $x$
     return no
   ```

   Assuming that ϕ is a satisfiable formula. Let x^* be an arbitrary satisfying assignment of ϕ. We can measure the *distance* between x^* and any assignment x by the number of variables that have different values. Observe that the distance between x^* and any x is at most the number of variables n in ϕ, and $x^* = x$ if and only if their distance is zero.

   (a) Let C be any unsatisfied clause chosen by RANDOMkSAT. What is the probability that flipping a variable uniformly at random decreases the distance between x^* and x?

[1] that is, the error probability is less than $1/\operatorname{poly} n$

(b) Imagine an abstract version of tug-of-war: Let P be a path with n edges and $n+1$ vertices, labelled 0 to n according to the order on P. At any moment at an interior vertex i, we have 50% chance moving to vertex $i-1$ and 50% chance moving to vertex $i+1$. If we ever move to vertex 0 we win; if we visit vertex n then we lose.

For each i, compute the probability of winning when starting at vertex i.

(c) Prove that the algorithm RANDOM2SAT solves 2SAT in polynomial time with high probability. *[Hint: How does the tug-of-war game relate to* RANDOM2SAT*?]*

⋆(d) Why doesn't RANDOM3SAT solve 3SAT in polynomial time? Can you modify the algorithm so that it solves 3SAT (even no longer in polynomial time)?