Question 1:

Suppose you have a fair coin that can be flipped independently for an unbounded number of times. Is it possible to design a procedure that samples a number uniformly random over {1, 2, 3, 4, 5, 6}. This means that we simulate a fair die. Consider the following cases with analysis (correctness, running time):

- An algorithm that never fails and outputs the correct distribution perfectly, but might not terminate

- An algorithm that might fail, but the output is the perfectly correct under the condition that the algorithm does not fail.

- An algorithm that does not fail and outputs almost correctly.

Suppose you can simulate a fair die using a fair coin (either perfectly or almost correctly). Argue that this implies you can generate random output over {1,2,3,4,5} as well (perfectly or almost correctly, respectively). The argument should be simple (i.e. using only one sentence).

Question 2:
Your simulated system may not have a perfectly fair coin where the probability of head is exactly 1/2. Suppose the simulated system only has a biased coin where Pr[head] = p for some $p \in [0\ 1]$, and that coin can be flipped independently for an unbounded number of times. Design a procedure that outputs a fair coin. You should also consider the following three cases, and everything should be argued with analysis.

- An algorithm that never fails and outputs the correct distribution perfectly, but might not terminate

- An algorithm that might fail, but the output is the perfectly correct under the condition that the algorithm does not fail.

- An algorithm that does not fail and outputs almost correctly.

In addition, what is the running time of your procedure? Does it depend on p?

Question 3:
Suppose a computer **only** has a fair coin that can be flipped for an unbounded number of times. Prove that there does not exist any algorithm that always terminates in a finite number of steps and outputs an element from {1,2,3,4,5} uniformly at random, i.e. the algorithm cannot output fail or anything outside the range {1,2,3,4,5}.

Hint: Given any positive number k, and an algorithm that tosses k coins, what is the sample space of the outcomes? Use a counting argument to prove the statement.

Question 4:
(a) Consider the set [n] = {1,2, …n}. We can generate a subset X of this set as follows: a fair coin is flipped independently for each element of the set [n]; if the coin is head, then the element is included in the set X, and otherwise it is not. Argue that the resulting set X is equally likely to be any one of the 2^n possible subsets. This gives you a simple way to generate a uniformly random subset of [n].

(b) Suppose that two sets X and Y are chosen independently and uniformly at random from all the 2^n subsets of [n]. Calculate the following probabilities: P(X $\subseteq$Y), and P(X $\cup$ Y = [n]).

Hint: (1) You can use the procedure (a) to calculate the probabilities. (2) The answers are simple.

Question 5:
We have a function F : {0,1,2,…n - 1} → {0,1,2,…m - 1}. We know that the function has some structure: for any x, y ∈ [n - 1],

$$F\big((x+y) mod n\big) = \big(F(x) + F(y)\big)\ mod\ m$$

This property is often known as “linear". Such a function can be stored by using a table of size n, e.g. the table is the array [F(0), F(1)…..F(n - 1)]. Suppose someone stores such a table inside a device and then sends that device to you. The specification of the device allows you to query some index “i” and returns F(i). However, the device was somewhat damaged by the time of delivery: 1/5 of the tables were corrupted, and you don't know which parts were corrupted. If an entry j is corrupted, the device will still return some value F’(j) to you upon query, but F’(j) might not be the original value F(j). Also the value F’(j) is a normal value in {0,1,2, …m - 1}, and it will not say “I am corrupted".

Problem: Describe a simple randomized algorithm that, given an input z, outputs a value that equals F(z) with probability at least 1/2. Your algorithm can query the device on whatever inputs in {0,1,2, …n - 1}, but it should query as fewer as possible. Also, your algorithm should work for every z no matter which parts of the tables were corrupted. Consider an algorithm that queries the device on the input z and outputs the answer from the device. This algorithm does not work, because if it does not work for every z. If the entry at z is corrupted, then the algorithm will output something wrong. Suppose you are allowed to repeat the initial algorithm three times, what should you do? What is the probability that your enhanced algorithm returns the correct answer?