1. **Review of Halting Problem**

   Ask your TA any question about the Halting Problem.

2. **Code reachability is impossible**

   Consider triplets (M, x, L) where

```
M is a java program
x is some input
L is an integer
```

   and the question of: if we execute $M(x)$, do we ever hit line L?

   Prove this problem is undecidable.

   Answer:

   Suppose we had a procedure that could decide the above. Consider the following program for deciding whether $M(x)$ halts:

```
line    0    ...
line    1    ... source code of M ...
line    2    ...

line  N+1    int main() {
line  N+2      run M(x);
line  N+3      foobar;
line  N+3    }
```

   Then we ask, is line $N+3$ ever executed?

   If so, it means that $M(x)$ halts. Otherwise, $M(x)$ infinite looped.

3. **Compute this**

   (a) Can you write a program that gets n (a natural number) as input and finds the shortest formula that computes n? A formula is a valid sequence consisting of decimal digits, the operators $+$, $\times$, ^ (raising to the power), and parentheses. The length of a formula is simply the number of characters you need to use to type it (i.e. each operator, decimal digit, or paranthesis counts as one character).

   Answer: Yes it is possible to write such a program. We already know one way to write a formula for n, which is to just write the number n (with no operators). Let the length of this formula in characters be l. In order to find the shortest formula we simply need to search among formulae that have length at most l. But there are a finite number of such formula and we can write a program that iterates over all of them (e.g. by treating each character as a byte or an 8-bit number, the whole formula becomes a binary integer of length at most $8l$, so we can simply iterate over all binary numbers up to 2^{8l} and for each one check if it is a valid formula). For each formula that we encounter we can compute its value in finite time (since there are no loop/control structures in formula). Therefore we can check whether it computes n, and then among those that do compute n we find the smallest one.

(b) Now assume that you want to write a computer program that given the input n (a natural number) finds another computer program (in a specific language, e.g. C or Python) that prints n. The program that is found has to have the minimum length plus execution time amongst all programs that print n, where length is measured by the number of characters in the source code and execution time is measured by a concrete number such as the number of CPU instructions executed. Can this be done?

Answer: Yes. Again it is possible to write such a program. Again given a number n there is one way we know that we can write n, which is to print its digits one by one (each using a print statement for example). So we already know of a program that prints n. Let the length plus running time of this program be l (where $l \simeq c \lg n$ for some constant c, but this does not matter). We only need to check programs that have a length of at most l and a running time of at most l (since otherwise their running time plus length would be bigger than l). One can again similarly to the previous part iterate over all programs of length at most l (by treating each one as a large binary integer and checking each one's validity by e.g. compiling it). One might be tempted to again run each program and check whether its output is n, but the problem is that we do not know how long each program runs for (if we knew we could have solved the halting problem which is impossible). But note that programs that have a running time of more than l can simply be discarded. So for each program, we run it for at most l steps. If it takes more time, we stop executing it and go to the next program, otherwise in at most l steps we see its output and we can check whether it is equal to n or not.

Now among all programs that have length at most l and execute for at most l steps and print n we find the one that has the shortest length plus execution time.

(c) Consider the set of programs (or functions) that take a single natural number n as input and output a natural number in at most $10^6 + 2^n$ steps (i.e. they always terminate after $10^6 + 2^n$ steps). Let this set be L. A member of L is called **thorough** if every natural number m can be produced as its output (by an appropriate input). As an example, a member of L that always returns values mod n (for some n in natural numbers) would not be thorough. Can you write a program that takes a member of L as input and determines whether that member is thorough? The given member of L is guaranteed to be in L, there is no need for your program to verify the membership.

(HINT: If you had such a program, could you somehow use it to solve the halting problem? If so, what would that mean?)

Answer: No, this is not possible. We will prove this by contradiction. So assume that this task was possible and we have a program P that takes as its input a member of L and determines whether it is thorough or not. We will show that by using P we can solve the halting problem which we know is impossible, therefore proving that P cannot exist.

To solve the halting problem, consider an input to the halting problem H. We need to determine whether H halts at some point or not. Let us construct a program G that takes a natural number n as its input and simulates H for at most n steps and then prints the number of steps H ran for in the output. The program G is thorough if and only if H never halts. This is because if H halts, then it stops after some finite number of steps l, and G never outputs any number larger than l. But if H does not halt, then on every input n, the program G simulates H for the whole n steps and outputs n. So G prints its input to the output and therefore is thorough.

Note that the program G is in L, because simulating H for n steps takes time at most a small constant times n (e.g. $10n$) which is always smaller than $10^6 + 2^n$. So now by feeding G into P and checking the ouput we can find out whether H was halting or not. This shows that with the help of P we can solve the halting problem. So it means that P cannot exist.