Logic Programming

1. Write a predicate sublists that relates a list of items to another list that contains every sublist of the first. For instance, the goal

   ```
   sublists([1, 2, 3], X)
   ```

   succeeds with

   ```
   X = [[], [3], [2], [2, 3], [1], [1, 3], [1, 2], [1, 2, 3]]
   ```

   You may find it useful to use `add_to_all` in your definition.

   Note: By default, the `swipl` interpreter will truncate output. Type in the `w` character to see the full output.

```
sublists([], [[]]).
% write the recursive rule here
```

2. Write a predicate `unzip` that succeeds if its first argument is the "zip" of the second and third arguments, meaning that it interleaves the items from the second and third arguments. The second argument is required to be exactly the same length as, or one item larger than, the third argument. For example:

   ```
   ?- unzip([a, b, c, d, e, f], Evens, Odds).
   Evens = [a, c, e],
   Odds = [b, d, f].

   ?- unzip([a, b, c, d, e], Evens, Odds).
   Evens = [a, c, e],
   Odds = [b, d] .
   ```

   Hint: You will likely find that you need two base cases, one for when the two lists have the same length and one for when the first list is one item larger.

```
% write your solution for unzip here
```

3. For the prime sieve in lecture, we wrote a `numbers` predicate that relates a number `N` to an ordered list of numbers from 2 to `N`. However, the implementation in lecture takes quadratic time, since it uses the `append` predicate. Write another version of `numbers` that has a "tail-recursive" structure so that it only takes linear time.

```
% write your solution for numbers here
```