The following exercises should be completed in the Racket programming language [1]. Remember to plan your work and make regular commits to your repository. The instructions for submitting your work are given on the Moodle page. Note that "from scratch" means using only `cons`, `car`, `cdr`, `define`, `lambda`, `if`, `null`, `null?`, `cond`, `map`, `=` and the basic numerical operators (`+`, `-`, `*`, `/`, `modulo`). Other basic functions may be allowed, but please confirm their use with the lecturer.

1. Write, from scratch, a function in Racket that uses a brute-force algorithm that takes a single positive integer and return true if the number is a prime and false otherwise. Call the function `decide-prime`.

2. Write, from scratch, a function in Racket that takes a positive integer n_0 as input and returns a list by recursively applying the following operation, starting with the input number.

$$n_{i+1} = \begin{cases} 3n_i + 1 & \text{if } n_i \text{ is odd} \\ n_i \div 2 & \text{otherwise} \end{cases}$$

   End the recursion when (or if) the number becomes 1. Call the function `collatz-list`. So, `collatz-list` should return a list whose first element is n_0, the second element is n_1, and so on. For example:

```
> (collatz-list 5)
'(5 16 8 4 2 1)
> (collatz-list 9)
'(9 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1)
> (collatz-list 2)
'(2 1)
```

3. Write, from scratch, two functions in Racket. The first is called `lcycle`. It takes a list as input and returns the list cyclically shifted one place to the left. The second is called `rcycle`, and it shifts the list cyclically shifted one place to the right.

   For example:

```
> (lcycle (list 1 2 3 4 5))
'(2 3 4 5 1)
> (rcycle (list 1 2 3 4 5))
'(5 1 2 3 4)
```

4. Write a function `sublsum` in Racket that takes a list (of integers) as input and returns a list of sublists of it that sum to zero. For this problem, you can use the `combinations` built-in function. Note the order of the sublists and their elements doesn't matter. For example:

```
> (sublsum (list 1 2 3 4 -5))
'((2 3 -5) (-5 1 4))
> (sublsum (list 1 2 3 4 5))
'()
```

5. Write a function `hamming-weight` in Racket that takes a list l as input and returns the number of non-zero elements in it. For example:

```
> (hamming-weight (list 1 0 1 0 1 1 1 0))
5
```

6. Write a function `hamming-distance` in Racket that takes two lists and returns the number of positions in which they differ. For example:

```
> (hamming-distance (list 1 0 1 0 1 1 1 0) (list 1 1 1 1 0 0 0 0))
5
```

7. Write a function `maj` in Racket that takes three lists x, y and z of equal length and containing only 0's and 1's. It should return a list containing a 1 where two or more of x, y and z contain 1's, and 0 otherwise. For example:

```
> (maj (list 0 0 0 0 1 1 1 1) (list 0 0 1 1 0 0 1 1) (list 0 1 0 1 0 1 0 1))
'(0 0 0 1 0 1 1 1)
```

8. Write a function `chse` in Racket that takes three lists x, y and z of equal length and containing only 0's and 1's. It should return a list containing the elements of y in the positions where x is 1 and the elements of z otherwise. For example:

```
> (chse (list 0 0 0 0 1 1 1 1) (list 0 0 1 1 0 0 1 1) (list 0 1 0 1 0 1 0 1))
'(0 1 0 1 0 0 1 1)
```

9. Write a function `sod2` in Racket that takes three lists x, y and z of equal length and containing only 0's and 1's. It should return a list containing a 1 where the number of 1's in a given position in x, y and z contains an odd nubmer of 1's, and 0 otherwise. For example:

```
> (sod2 (list 0 0 0 0 1 1 1 1) (list 0 0 1 1 0 0 1 1) (list 0 1 0 1 0 1 0 1))
'(0 1 1 0 1 0 0 1)
```

10. Write a function `lstq` in Racket that takes as arguments two lists l and m of equal length and containing numbers. It should return d, the distance given by the sum of the square residuals between the numbers in the lists:

$$d = \sum_i (l_i - m_i)^2$$

This means take the i^{th} element of m from the i^{th} element of l and square the result for all i. Then add all of those to get d. For example:

```
> (lstq (list 4.5 5.1 6.2 7.8) (list 1.1 -0.1 6.1 3.8))
54.61
```

References

[1] PLT Inc. Racket – a programmable programming language.