Pada minggu ini, saya mempelajari banyak tentang fold dan map. Materi cukup challenging namun tidak memungkinkan untuk dikuasai. Selain itu saya juga mengerjakan hackerrank dan mempelajari banyak, terutama tentang cara memproses input.

3. Saya asumsi fold bebas digunakan, sehingga saya membuat helper function sum yang menggunakan fold.

- Map melakukan mapping ^2 kepada setiap elemen dari 1 sampai n.
- List comprehension melakukan hal yang sama, yaitu untuk setiap x dari 1 .. n, masukkan ke dalam list sebagai x ^ 2.

```
sum' :: (Num a) => [a] -> a
sum' = foldr (+) 0

sumOfSquares1 :: Int -> Int
sumOfSquares1 n = sum' (map (^ 2) [1 .. n])

sumOfSquares2 :: Int -> Int
sumOfSquares2 n = sum' [x ^ 2 | x <- [1 .. n]
```

7.

```
[x+y|x<-[1..4], y<-[2..4], x > y]
kalau dari yang saya mengerti, list comprehension mulai dari paling kiri.
[x_0 + y+0, x_0 + y_1, x_0 + y_2, ..., x_3 + y_3, x<-[1..4], y<-[2..4], x
> y]

jadi akan dibuat pasangan x + y for every pair x, y in x = [1,2,3,4] dan y
= [2, 3, 4]

tanpa kondisi x > y, maka akan hasilkan 4 * 3 = 12 elemen
[3,4,5,4,5,6,5,6,7,6,7,8]

namun karena ada kondisi tersebut, harus dicek terlebih dahulu
[1+2, 1+3, 1+4, 2+2, 2+3, 2+4, 3+2, 3+3, 3+4, 4+2, 4+3, 4+4]
   ^    ^   ^     ^    ^   ^          ^   ^               ^

elemen yang ditanda ^ tidak memenuhi x > y, maka akan dihapus
[3 + 2, 4 + 2, 4 + 3]

sehingga hasil akhir list comprehension adalah
[5, 6, 7]

dan benar setelah dicek dengan jalankan program :)
```

Ternyata nomor 9 yang aku kerjain ini soal tahun lalu, tapi aku tetep include aja buat semacam refleksi

9.

```
sieve (p:xs) = p : sieve [x | x <- xs, x `mod` p > 0]
```

jadi sieve menerima array. Asumsi array ini berisi angka-angka berurutan contoh angka tidak berurutan tidak berjalan sesuai ekspektasi, 9 dianggap prima karena tidak ada divisor 3.

```
Prelude> take 4 (sieve [2, 4, 6, 8, 9, 10, 11, 12, 13])
[2,9,11,13]
```

untuk setiap elemen pertama `p`, dimana `p` adalah bilangan prima, prima pertama adalah 2, akan dilakukan looping kepada `xs` yaitu sisa dari list, dan dilakukan list comprehension `[x | x <- xs, x` mod `p > 0]` untuk setiap prima. Kita coba gali lebih lanjut kutipan kode tersebut.

```
Prelude> [x | x <- [3..11], x `mod` 2 > 0]
[3,5,7,9,11]

return 2 : sieve [3,5,7,9,11]
```

terlihat list yang direturn adalah semua angka pada `[2..n]` yang tidak habis dibagi dengan `p`, atau pada kasus pertama ini p = 2. List ini lalu menjadi argumen dari pemanggilan rekursif.

```
return [2, 3]
Prelude> [x | x <- [5,7,9,11], x `mod` 3 > 0]
[5,7,11]

return 2 : 3 : sieve [5,7,11]

Prelude> [x | x <- [7,11], x `mod` 5 > 0]
[7,11]
return 2 : 3 : 5 : sieve [7,11]

Prelude> [x | x <- [11], x `mod` 7 > 0]
[11]
return 2 : 3 : 5 : 7 : sieve [11]

Prelude> [x | x <- [], x `mod` 1 > 0]
[]
return 2 : 3 : 5 : 7 : 11 : sieve []
```

yang direturn adalah [2, 3, 5, 7, 11]

tes program dulu

```
Prelude> take 10 (sieve [2..11])
[2,3,5,7,11*** Exception: <interactive>:9:1-53: Non-exhaustive patterns in
function sieve
```

benar!