FUNCTIONAL PROGRAMMING 2017

MOISE GABRIEL

QUESTION 2

(a)

```
> merge :: [Integer] -> [Integer] -> [Integer]
> merge (x:xs) (y:ys)
>   | x == y    = x : merge xs ys
>   | x < y     = x : merge xs (y:ys)
>   | otherwise = y : merge (x:xs) ys
```

(b)

```
> hamming :: [Integer]
> hamming = h
>     where h = 1 : merge (map (2*) h) (merge (map (3*) h) (map (5*) h))
```

(c) The list we created has a cyclic structure as it is created starting from its first element, 1, and, due to lazy evaluation, each time a new element is created, it gets mapped in the three map calls and then it is added to the lists which are about to get merged. The first list is [2,4,6,...] and it grows once new elements are added to h (through comparisons at merge) and the other one is [3,5,6,9,10,12,...], which is being formed from [3,6,...] and [5,10,...].

(d) Since after one element is added to the "hamming" list, it is mapped and the results put in each of the three lists to be merged, the complexity of the function is linear, so it takes O(n) time to find the first n elements of hamming.

At (c) I think I lack a proper argument for proving the list's cyclicity and I don't know exactly how each element is produced and during what step it appears, so my explanation for (d) is particularly intuitive. Can you please make me understand it properly?

(e)

```
> hamming' :: [Integer] -> [Integer]
> hamming' as = h
>     where h = 1 : foldr1 merge [map (p*) h | p <- as]
```