**Part1**

**(a)**

Times = fun {$ N H }

fun {$}

(X#G) = {H} in

((X\*N)#{Times N G})

end

end

Merge = fun {$ Xs Ys}

fun {$}

(X#G) = {Xs}

(Y#Z) = {Ys} in

if (X < Y) then

(X#{Merge G Ys})

else

if (X > Y) then

(Y#{Merge Xs Z})

else

(X#{Merge G Z})

end

end

end

end

Generate = fun {$ N}

fun {$} (N#{Generate (N+1)}) end

end

Hamming = fun {$ N}

fun {$}

(X#G) = {N} in

(1#{Merge {Times 2 {Hamming G}} {Merge {Times 3 {Hamming G}} {Times 5 {Hamming G}}}})

end

end

Take = fun {$ N F}

if (N == 0) then

nil

else

(X#G) = {F} in

(X| {Take (N-1) G})

end

end

X = {Generate 1}

Y = {Hamming X}

V = {Take 10 Y}

skip Browse V

A picture containing text, whiteboard

Description automatically generated

**(b)**

**data *Gen* a = G (() -> (a, *Gen* a))**

**generate :: *Int* -> *Gen* *Int***

**generate n = G (\\_ -> (n, generate (n+1)))**

**gen\_take :: *Int* -> *Gen* a -> [a]**

**gen\_take 0 \_ = []**

**gen\_take n (G f) = let (x,g) = f () in x : gen\_take (n-1) g    *-- What's the type of f here? --  f will be (Int, Gen Int)***

**times :: *Int* -> *Gen* *Int* -> *Gen* *Int***

**times n (G f) = let (x,g) = f () in G(\\_ -> ((n\*x),times n g))**

merge :: *Gen* *Int* -> *Gen* *Int* -> *Gen* *Int*

merge (G f) (G p) = let (x,g) = f () in let (y,k) = p() in

    if x < y then G (\\_ -> (x,merge g (G p)))

    else if y < x then G (\\_ -> (y,merge k (G f)))

    else  G (\\_ -> (x, merge g k))

hamming :: *Gen* *Int* -> *Gen* *Int*

hamming (G f) = let (x,g) = f () in G (\\_ -> (1,merge (times 2 (hamming g)) (merge (times 3 (hamming g)) (times 5 (hamming g)))))

**Part2**

(a)

IntToNeed = fun {$ L}

case L of [] '|'(1:X 2:Xr) then

local Y in

({ByNeed proc{$ A} A = X end Y}|{IntToNeed Xr})

end

nil then nil

end

end

(b)

AndG = {GateMaker fun{$ X Y} if (X == 0) then 0 else (X\*Y) end end} // Use GateMaker

OrG = {GateMaker fun{$X Y} if (X == 1) then 1 else (X+Y)-(X\*Y) end end} // Use GateMaker

(c)

fun {MulPlex A B S}

K L M in

K = {AndG {NotG S} A}

L = {AndG S B}

M = {OrG k L}

M

end

(d)

(1)

A = {IntToNeed [0 1 1 0 0 1]}  
B = {IntToNeed [1 1 1 0 1 0]}  
S = [1 0 1 0 1 1]  
Out = {MulPlex A B S}

In this case here, A [0], A[2], A[4],A[5],B[1],B[3], will not be needed. Because in andGate, when the first value is 0, the result is 0, and no need to check the second value.

(2)

Yes, they did match up with what I got in d(1)