part1a

```
local O Generate TakeLazy Times Merge Ham in

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

   fun {Times N F}
      fun {$}
         (A#As) = {F} in
         ((N*A)#{Times N As})
      end
   end

   fun {Merge Xs Ys}
      fun {$}
         (A#As) = {Xs} (B#Bs) = {Ys} in
            if (A < B) then
               (A#{Merge As Ys})
            elseif (A == B) then
               (A#{Merge As Bs})
            else
               (B#{Merge Xs Bs})
            end
      end
   end

   fun {TakeLazy N F}
      (A#As) = {F} in
         if (N == 0) then
            nil
         else
            (A|{TakeLazy (N-1) As})
      end
   end

   fun {Ham}
      fun {$}
         local F = {Ham} in
            (1#{Merge {Times 2 F} {Merge {Times 3 F} {Times 5 F}}})
         end
      end
   end

   O = {TakeLazy 10 {Ham}}
   skip Browse O
end



  //fun lazy {Merge X Xs}
  //
  //end
```

part1b

```haskell
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

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

merge :: Ord a => Gen a -> Gen a -> Gen a
merge (G f) (G t) = let (a1, b1) = f ()
                        (a2, b2) = t ()
                        cmp = compare a1 a2 in
                            case cmp of EQ -> G(\_ -> (a1, merge b1 b2))
                                        GT -> G(\_ -> (a2, merge (G f) b2))
                                        LT -> G(\_ -> (a1, merge b1 (G t)))

hamming :: () -> Gen Int
hamming () = G(\_ -> (1, (merge (times 2 (hamming()))
                                (merge (times 3 (hamming())) 
                                    (times 5 (hamming ()))))))
```

```
local GateMaker AndG OrG NotG A B S IntToNeed Out MulPlex in

   fun {GateMaker F}
      fun {$ Xs Ys} GateLoop T in
         fun {GateLoop Xs Ys}
            case Xs of nil then nil
                     [] '|'(1:X 2:Xr) then
                        case Ys of nil then nil
                                 [] '|'(1:Y 2:Yr) then
                                    {F X Y}|{GateLoop Xr Yr}
                        end
            end
         end
         T = thread {GateLoop Xs Ys} end   // thread isn't (yet) a returnable expression
      T
      end
   end

   fun {NotG Xs} NotLoop T in
      fun {NotLoop Xs}
         case Xs of nil then nil
                  [] '|'(1:X 2:Xr) then ((1-X)|{NotLoop Xr})
         end
      end
      T = thread {NotLoop Xs} end          // thread isn't (yet) a returnable expression
      T
   end


//----------------------------------------
   //part b
   AndG = {GateMaker
               fun {$ X Y}
                  if (X==0) then 0  //short circuit occurs here
                  else if (Y==0) then 0
                  else 1
                  end
                  end
               end
   }
   OrG = {GateMaker
               fun {$ X Y}
                  if (X==1) then 1  //short circuit occurs here
                  else if (Y==1) then 1
                  else 0
                  end
                  end
               end
   }

    //part a
   fun {IntToNeed L}
      case L of nil then nil
         [] '|'(1:A 2:As) then F in
         byNeed proc {$ G} G=A end F   //used to make A a byNeed variable
         '|'(1:F 2:{IntToNeed As})
      end
   end

    //part c
   fun {MulPlex A B S}
      local X Y Z R in
         X = {NotG S}      //not S
         Y = {AndG X A}    //A and not S
         Z = {AndG S B}    //B and S
         R = {OrG Y Z}     //Or for and gates
         R
      end
   end

   //d.1
    /*
    bits 0 2 4 and 5 are not needed for a
    bits 1 and 3 are not needed for b
    this can be seen by outputting lists a and b
    */
   //d.2
    /*
    Needed: 213
    Needed: 265
    Needed: 275
    Needed: 326
    Needed: 347
    Needed: 403

    Out : [ 1  1  1  0  1  0 ]

    ((210), 1),
    ...
    ((265, 249, 46, 44, 29, 22), 1),
    ...
    ((275, 270, 100, 98, 80, 72), 1),
    ...
    ((326, 321, 56, 54, 33, 24), 0),
    ...
    ((347, 342, 110, 108, 84, 74), 1),
    ...
    ((403, 380, 115, 113, 86, 75), 0),

    These values match up with my predictions in 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}



   // run a loop so the MulPlex threads can finish before displaying Out
   local Loop in
      proc {Loop X}
         if (X == 0) then skip Basic
         else {Loop (X-1)} end
      end
     {Loop 1000}
   end

   skip Browse Out
end
```

part2