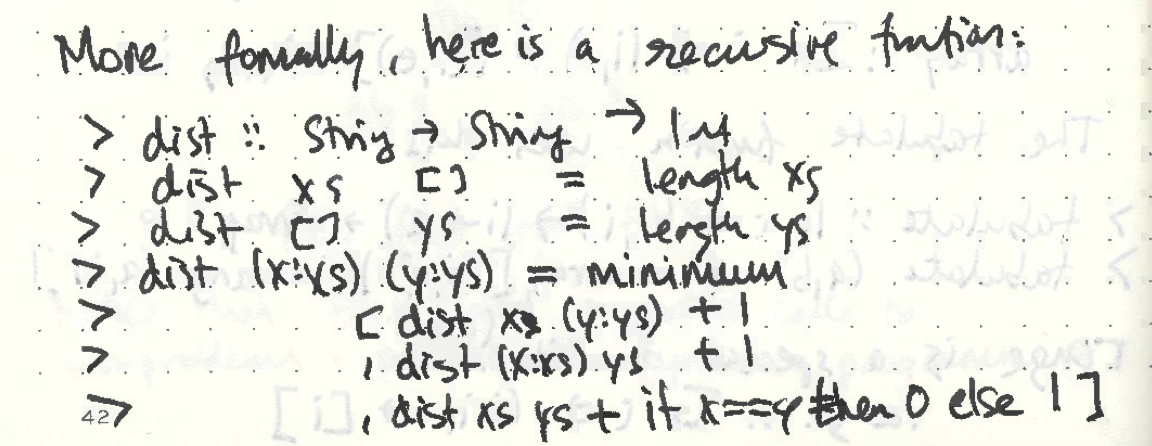
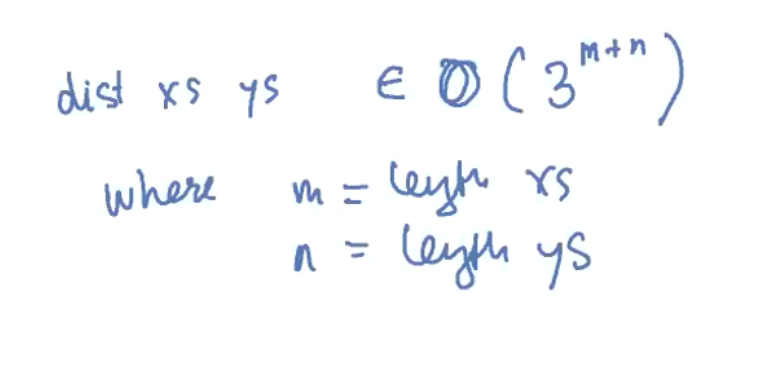
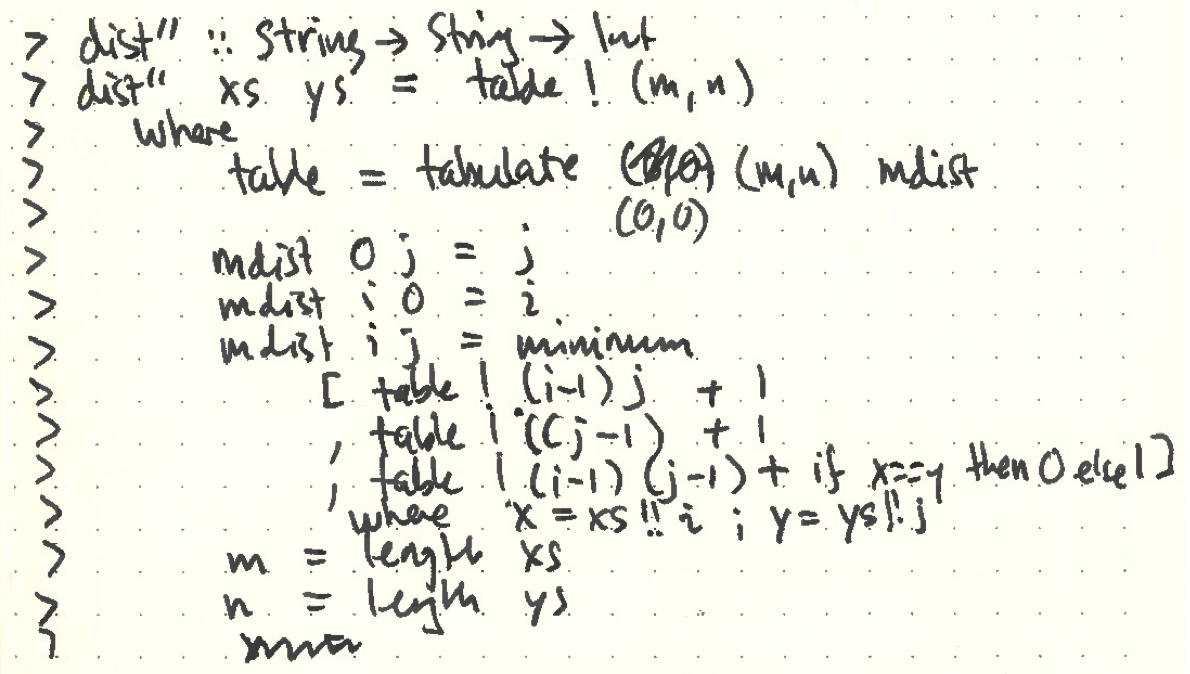
Lf[Panopto: <https://imperial.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=d7147440-6919-4337-952e-ab8400ce11ce>]

1ai) (from notes)

aii) O(3^(m+n))



Explanation: Our dist function looks at three different cases: removing a letter from xs, removing a letter from ys and continuing on with comparing the rest of the string. We can bound each of the base case operations by n + m (n = length xs, m = length ys). Our inductive case causes us to check three further scenarios each inductive step meaning that in general, our complexity should be 3^(m + n).

bi) 

c)

dists :: String -> String -> [String]

dists = go

where

go [] ys = inits ys

go xs [] = tails xs

go (x : xs) (y : ys)

| x == y = map (x :) (go xs ys)

| otherwise = (x : xs) : ss

where

ss = minimumBy (compare `on` length) [ go xs (y : ys)

, map (y:) $ go (x : xs) ys

, map (y:) $ go xs ys]

Guess this works...

2ai) O(2^n)

Explanation: Generally, we have to make 2 calls per fib call. Our fib calls decrement linearly from n to 0 so our power is linear n.

aii)

fib’ :: Int -> Int

fib’ n = fibR n 1 0

Where

fibR :: Int -> Int -> Int -> Int

fibR 0 \_ n = n

fibR cnt x n = fibR (cnt - 1) (x + n) x

aiii)

gr **::** **Double**

gr **=** (1 **+** sqrt 5) **/** 2

fib **::** **Int** **->** **Integer**

fib n **=** round (gr' n **/** sqrt 5)

**where**

gr' **::** **Int** **->** **Double**

gr' 0 **=** 1

gr' 1 **=** gr

gr' p **=** x **\*** x **\*** **if** p **`mod`** 2 **==** 0 **then** 1 **else** gr

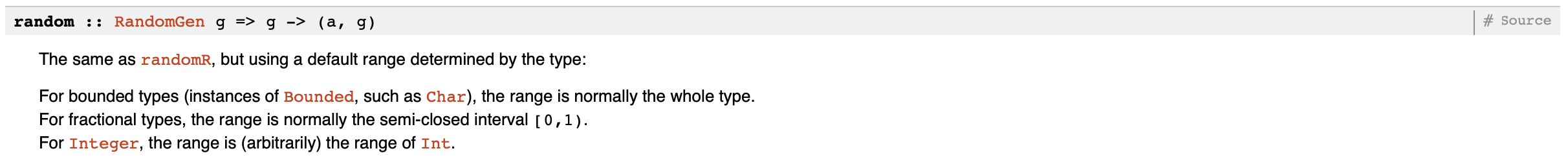
**where**

x **=** gr' (p **`div`** 2)

bi)

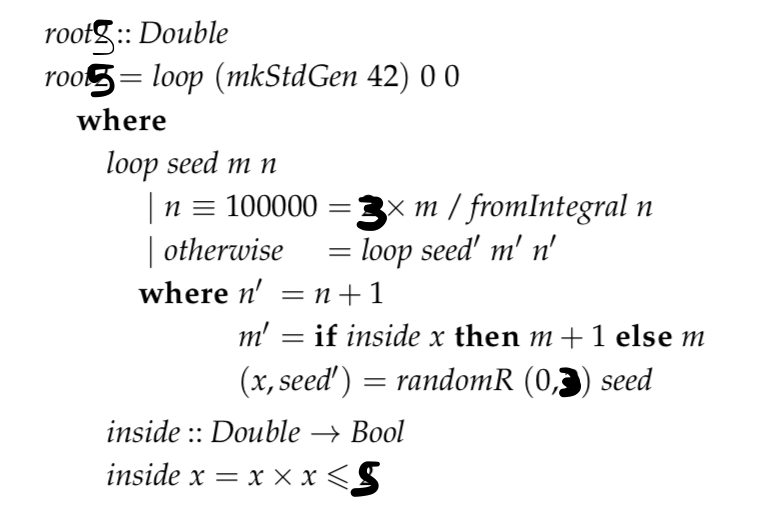
mkStdGen :: Int -> StdGen - Initialises a RNG

Random :: Random a => StdGen -> (a, StdGen)



RandomR :: Random a => (a, a) -> StdGen -> (a, StdGen): Produces a random number using the given RNG within the range of (x, y), returning this number along with a new RNG to continue ‘randomisation’

bii) (from full notes)

ho

Reason the range of (0, 3) is used: [https://piazza.com/class/k0r 3c44f879b3?cid=207](https://piazza.com/class/k0r  3c44f879b3?cid=207)

3ai) Instance Stack [Int] where

empty = []

Push x xs = x:xs

Pop [] = []

Pop (x:xs) = xs

Peek [] = Nothing

Peek (x:xs) = Just x

(look default implementation inherited from class definition)

Push, Pop and Peek are all O(1)

ii) Look implementation (same implementation provided used for lists and stacks) is of time complexity O(n) (or O(k) if needs to be stricter) when a list/stack is indexed for position k. This is because we must go from the start of the list and move through each element of the list until we reach position k.

bi)

*data* *StackArray* *=* StackArray (*Array* *Int* *Int*) *Int* *Int*

*-- first Int is number of elements we have*

*-- second Int is capacity*

bii)

*instance* *Stack* *StackArray* *where*

empty *=* StackArray (fromList []) 0 0

pop (StackArray a 0 n) *=* StackArray a 0 n

pop (StackArray a m n) *=* StackArray a (m *-* 1) n

peek (StackArray a 0 n) *=* Nothing

peek (StackArray a m n) *=* Just (a *!* (n *-* m))

look i (StackArray a m n)

*|* i *<* m *=* Just (a *!* (n *-* m *+* i))

*|* otherwise *=* Nothing

push x (StackArray a m n)

*|* m *<* n *=* StackArray (modify a (n *-* m *-* 1) x) (m *+* 1) n

*|* otherwise *=* StackArray (fromList (replicate n x *++* toList a)) (m *+* 1) (2 *\** n) *-- double the size of the array*

biii)

To prove that push has amortised complexity of constant time, we show that

C <= A + S(xs0) - S(xsn) is true, where A is some constant value.

Cpush = if (m<n) then 1 else n

Apush = 3

Size(StackArray axs m n) = 2m – n

By case analysis

**Case m<n:**

Size(StackArray axs m n) = 2m – n

Size after push

Size(StackArray axs’ m’ n’) = 2m’ - n’

Where m’ = m+1 and n’= n

= 2(m+1) - n

So Size(before) - Size(after) = (2m – n) - (2(m+1) - n)

= 2m – n – 2m – 2 + n

= -2

So Cpush <= Apush –2

<=> 1 <= 3-2

<=> True so Apush in O(1)

**Case m == n:**

Size(StackArray axs m n) = 2m – n

= 2m – m (as m = n)

= m

Size after push

Size(StackArray axs’ m’ n’) = 2m’ - n’

Where m’ = m + 1 and n’ = 2n

= 2(m+1) - 2n

= 2m + 2 – 2n

= 2m + 2 – 2m (as m ==n)

= 2

So Size(before) - Size(after) = m - 2

So Cpush <= Apush + m – 2

<=> m <= 3 + m – 2

<=> m <= m + 1

<=> True so Apush in O(1)