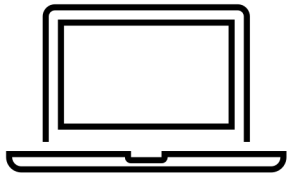
Week 4 Exercises

Once again, this includes some exercises presented in class and additional exercises to work on in your own time.

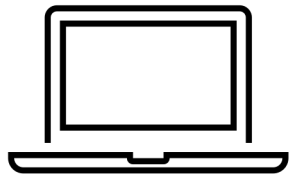
Some problems are considerably more challenging that others.

Give it a go!



1. Download trees.hs from Blackboard. Test it – confirm that balance works and prints the tree that you would expect.

Give it a go!



2. Write functions to return those properties of a BST:
    a. minimum
    b. maximum
    c. successor
    d. predecessor

Give it a go!

A pencil and paper exercise this time (or type it if you want):

3. Say I have this function definition:

```
sumwith :: Int -> [Int] -> Int
sumwith v [] = v
sumwith v (x:xs) = sumwith (v+x) xs
```

What is the sequence of steps (in which order are things calculated) when I evaluate:

```
> sumwith 4 [1,2,0]
```

Give it a go!

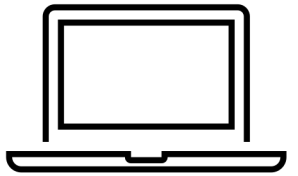
4. If my definition of sumwith was instead:

```
sumwith v [] = v
sumwith v (x:xs) = (sumwith $! (v+x)) xs
```

What is the sequence of steps (in which order are things calculated) when I evaluate:

```
> sumwith 4 [1,2,0]
```

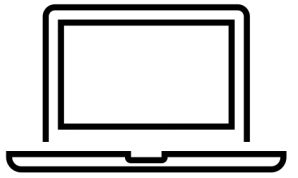
Give it a go!



5. [Hutton, ex 15.9] Define functions that are versions of the library functions repeat, take and replicate, except for binary trees, not lists.

Hint: start with take…

Give it a go!



6. [Hutton, ex. 15.6]

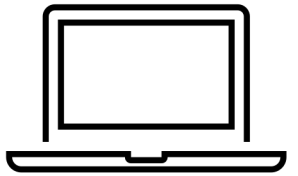
Newton's method for calculating the square root of a (non-negative) floating-point number `n` can be expressed as follows:

- start with an initial approximation to the result
- given the current approximation `a`, the next approximation is defined by the function `next a = (a + n/2) / 2`
- repeat the second step until the two most recent approximations are with some desired distance of one another, at which point the most recent value is returned as the result

Define a function `sqroot :: Double -> Double` that implements this procedure.

Hint: First produce an infinite list of approximations using the library function `iterate`. For simplicity, take the number `1.0` as the initial approximation, and `0.00001` as the distance value.

Give it a go!



7. Use the mergesort code from week 3 to try out using Debug.Trace
   a) First, try it as shown in the lecture slides, and test it with a simple list of numbers.
   b) Can you print different information that might be more useful?
   c) Can you make use of tracing in some of your other work to understand better what is happening when your code is evaluated?

Finished all these? Revisit exercises from previous weeks. Think about alternative solutions to the ones you wrote the first time round – can you produce more elegant solutions now?