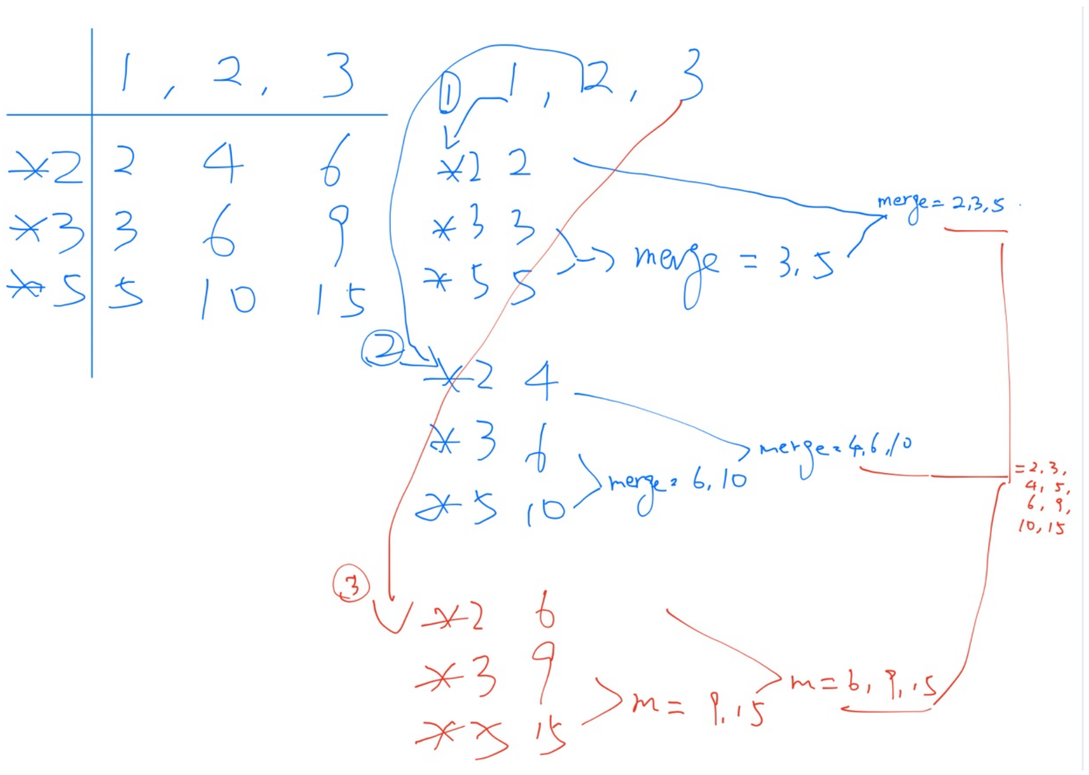
Coursework I

Task I.1

After the first 1, 2, and 3 elements have been printed, which will go through (map (*3) [1..]) (map (*5) [1..])) and (map (*2) [1..]). And the elements in map(*3) and map(*5) will be merged into a list, which will be merged with the map(*2) list. As the following figure:



Task I.2

As discussed, this evaluator has a weakness. Because if this is a spreadsheet written by a user who might make a mistake and write the definition that can not be evaluated. From my perspective, the results of the calculation should be packaged in Maybe.

Task I.3

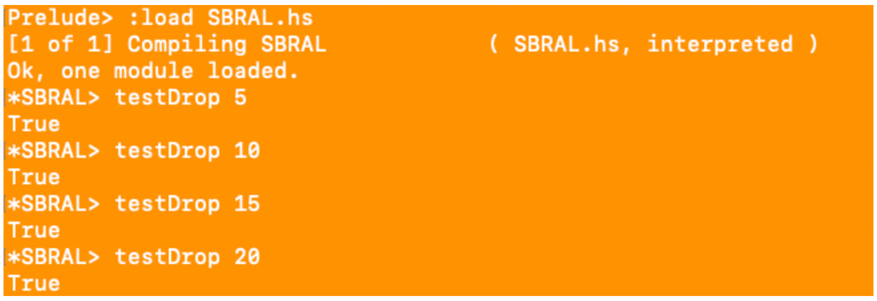
Explain how your implementation works:

Firstly, I define the drop function. In this function, I can drop n in the list. If (n >= w) drop (n-w) wts else (dropTree n w t) ++ wts. The dropTree contains I. drop 0 for lead; II. drop 0 for the node; remove left and drop right; drop left and keep right. Then I created the testDrop function to do some tests with the Bool. And I can check whether the drop function works well.

Explain why it has the desired time complexity:

Because the drop function discards half of the content that needs to be discarded each time, so its complexity is O(logn).

Test your solution: provide some evidence of testing with your answer:



```
Prelude> :load SBRAL.hs
[1 of 1] Compiling SBRAL            ( SBRAL.hs, interpreted )
Ok, one module loaded.
*SBRAL> testDrop 5
True
*SBRAL> testDrop 10
True
*SBRAL> testDrop 15
True
*SBRAL> testDrop 20
True
```

Task I.4

Provide the required instances, along with a brief explanation:

t1 = 1 +/- 0.5 == (Ivl 0.5 1.5)

t2 = Ivl 0 0 + Ivl 1 2 == Ivl 1 2

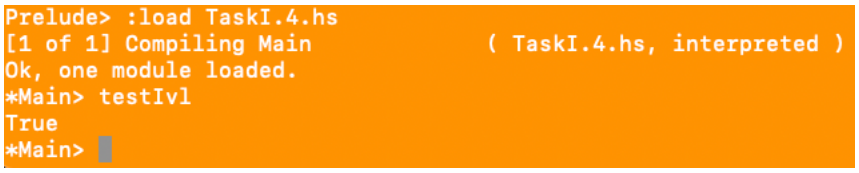
t3 = Ivl 2 3 - Ivl (-1) 4 == Ivl (-2.0) 4

1 +/- 0.5 means 1-0.5 and 1+0.5 equals 0.5,1.5

0+1 and 0+2 equals Ivl 1 2

2-4 and 3-(-1) equals -2.0 4

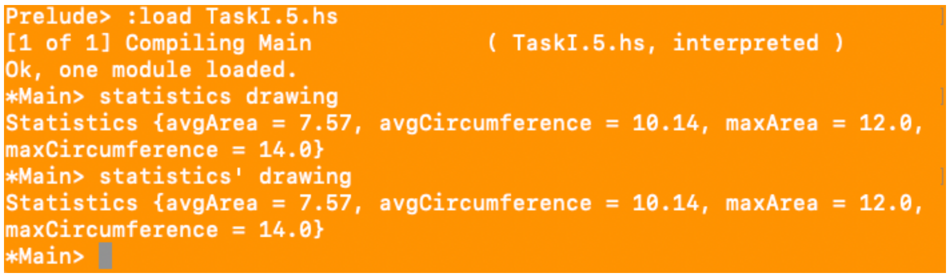
Test your solution: provide some evidence of testing with your answer:



```
Prelude> :load TaskI.4.hs
[1 of 1] Compiling Main             ( TaskI.4.hs, interpreted )
Ok, one module loaded.
*Main> testIvl
True
*Main> 
```

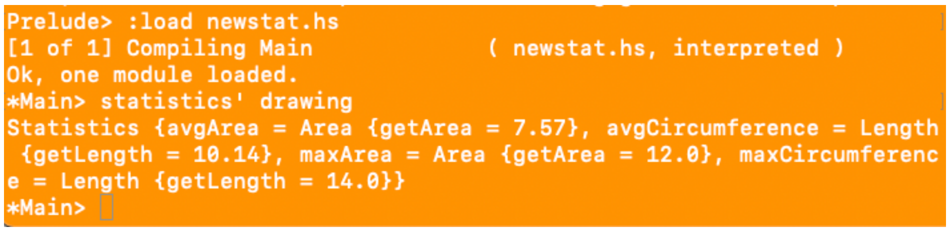
Task I.5

The function of ‘circumference’ is used to calculate the circumference of a rectangle and a circle. And ‘addAccumStats” is used to add the features of two objects which includes asCount, asSumArea, asSumCircumference, asMaxArea, asMaxCircumference(cnt, a, c, ma, mc).



```
Prelude> :load TaskI.5.hs
[1 of 1] Compiling Main             ( TaskI.5.hs, interpreted )
Ok, one module loaded.
*Main> statistics drawing
Statistics {avgArea = 7.57, avgCircumference = 10.14, maxArea = 12.0,
maxCircumference = 14.0}
*Main> statistics' drawing
Statistics {avgArea = 7.57, avgCircumference = 10.14, maxArea = 12.0,
maxCircumference = 14.0}
*Main> 
```

The new statistics’ :



```
Prelude> :load newstat.hs
[1 of 1] Compiling Main             ( newstat.hs, interpreted )
Ok, one module loaded.
*Main> statistics' drawing
Statistics {avgArea = Area {getArea = 7.57}, avgCircumference = Length
 {getLength = 10.14}, maxArea = Area {getArea = 12.0}, maxCircumferenc
e = Length {getLength = 14.0}}
*Main> 
```