```
Final Tests: Summary for anb122 of j1
-------------------------------------

  Public Tests:
    student-tests/hft-test/addP:         3 / 3
    student-tests/hft-test/mulP:         1 / 1
    student-tests/hft-test/sumP:         2 / 2
    student-tests/hft-test/prodP:        0 / 1
    student-tests/hft-test/diffT:        2 / 2
    student-tests/hft-test/intT:         2 / 2
    student-tests/hft-test/diffP:        1 / 1
    student-tests/hft-test/intP:         1 / 1
    student-tests/hft-test/diffE:        3 / 3
    student-tests/hft-test/intE:         8 / 16
    student-tests/hft-test/intESecret:   2 / 6
    hidden-tests/hft-test/addP:          3 / 3
    hidden-tests/hft-test/mulP:          1 / 1
    hidden-tests/hft-test/sumP:          2 / 2
    hidden-tests/hft-test/prodP:         0 / 1
    hidden-tests/hft-test/diffT:         2 / 2
    hidden-tests/hft-test/intT:          2 / 2
    hidden-tests/hft-test/diffP:         1 / 1
    hidden-tests/hft-test/intP:          1 / 1
    hidden-tests/hft-test/diffE:         3 / 3
    hidden-tests/hft-test/intE:          8 / 16
    hidden-tests/hft-test/intESecret:    2 / 6

Git Repo: git@gitlab.doc.ic.ac.uk:lab2324_spring/haskellfinaltest_anb122.git
Commit ID: 6b98a
```

22.5/30

```haskell
import IC.TestSuite

import Int
import Utilities
import Examples
import Types

main :: IO ()
main = runTests tests

tests :: [TestGroup]
tests =
  [ testGroup "addP" addPTests
  , testGroup "mulP" mulPTests
  , testGroup "sumP" sumPTests
  , testGroup "prodP" prodPTests
  , testGroup "diffT" diffTTests
  , testGroup "intT" intTTests
  , testGroup "diffP" diffPTests
  , testGroup "intP" intPTests
  -- PART II
  , testGroup "diffE" diffETests
  -- PART III
  , testGroup "intE" intETests
  , testGroup "intESecret" intESecretTests
  ]

addPTests :: [TestCase]
addPTests = [ pretty (addP p2 p2) --> "[2x]"
            , pretty (addP p3 p4) --> "[2x^2 + x - 2]"
            , pretty (addP p5 [(0, 0)]) --> "[2x^3 - 2x + 2]"
            ]

mulPTests :: [TestCase]
mulPTests = [ pretty (mulP p3 p4) --> "[8x^3 - 18x^2 + 13x - 3]"
            ]

sumPTests :: [TestCase]
sumPTests = [ pretty (sumP [[(0, 0)]]) --> "[0]"
            , pretty (sumP [p1, p2, p3, p4, p5]) --> "[2x^3 + 2x^2 + 5]"
            ]

prodPTests :: [TestCase]
prodPTests = [ pretty (prodP [p3, p5]) -->
                 "[4x^5 - 6x^4 - 2x^3 + 10x^2 - 8x + 2]"
             ]

diffTTests :: [TestCase]
diffTTests = [ diffT (1, 0) --> (0, 0)
             , diffT (2, 3) --> (6, 2)
             ]

intTTests :: [TestCase]
intTTests = [ intT (1, 0) --> (1, 1)
            , intT (2, 3) --> (1 / 2, 4)
            ]

diffPTests :: [TestCase]
diffPTests = [ pretty (simplify (P (diffP p3))) -->
                 -- without the `simplify . P`, this needs a `+ 0` on the end
                 "[4x - 3]"
             ]

intPTests :: [TestCase]
intPTests = [ pretty (intP p3) --> "[(2/3)x^3 + (-3/2)x^2 + x]"
            ]
```



```haskell

diffETests :: [TestCase]
diffETests = [ pretty (simplifiedDiff e3) --> "[4x - 3]"
             , pretty (simplifiedDiff e10) --> "[24x^2 - 36x + 13]"
             , pretty (simplifiedDiff e11) --> "[-1] . [x]^-2 . log[x] + [x]^-2"
             ]

intETests :: [TestCase]
intETests = [ fmap pretty (simplifiedInt e1) -->
                Just "[5x]"
            , fmap pretty (simplifiedInt e2) -->
                Just "[(1/2)x^2]"
            , fmap pretty (simplifiedInt e3) -->
                Just "[(1/6)] . [4x^3 - 9x^2 + 6x]"
            , fmap pretty (simplifiedInt e4) -->
                Just "[2x^2 - 3x]"
            , fmap pretty (simplifiedInt e5) -->
                Just "[(1/2)] . [x^4 - 2x^2 + 4x]"
            , fmap pretty (simplifiedInt e6) -->
                Just "[(1/6)] . [3x^4 + 4x^3 - 15x^2 + 18x]"
            , fmap pretty (simplifiedInt e7) -->
                Just "[(5/6)] . [3x^4 + 4x^3 - 15x^2 + 18x]"
            , fmap pretty (simplifiedInt e8) -->
                Just "[x] . ([-1] + log[x])"
            , fmap pretty (simplifiedInt e9) -->
                Just "log[x]"
            , fmap pretty (simplifiedInt e10) -->
                Just "[(1/2)] . [4x^4 - 12x^3 + 13x^2 - 6x + 1]"
            , fmap pretty (simplifiedInt e11) -->
                Just "[(1/2)] . (log[x])^2"
            , fmap pretty (simplifiedInt e12) -->
                Just "[2x^2 - 3x + 1] . ([-1] + log[2x^2 - 3x + 1])"
            , fmap pretty (simplifiedInt e13) -->
                Just "[(2/5)] . ([2x^2 - 3x + 1])^(5/2)"
            , fmap pretty (simplifiedInt e14) -->
                Just "[(2/25)] . ([5] . [2x^2 - 3x + 1])^(5/2)"
            , fmap pretty (simplifiedInt e15) -->
                Just "[(1/3)] . [x^3 - 3x] . ([-1] + log[x^3 - 3x])"
            , fmap pretty (simplifiedInt e16) -->
                Nothing
            ]

intESecretTests :: [TestCase]
intESecretTests = [ fmap pretty (simplifiedInt e17) -->
                      Just "[(2/125)] . ([5] . [2x^2 - 3x + 1])^(5/2)"
                  , fmap pretty (simplifiedInt e18) -->
                      Just "[(1/15)] . [x^3 - 3x] . ([-1] + log[x^3 - 3x])"
                  , fmap pretty (simplifiedInt e19) -->
                      Just "[(1/3)] . [x^3 - 3x]"
                  , fmap pretty (simplifiedInt e20) -->
                      Just "[(1/4)] . [x^4 - 6x^2]"
                  , fmap pretty (simplifiedInt e21) -->
                      Just "[(1/3)] . [x^3 - 3x] . ([-1] + log[x^3 - 3x])"
                  , fmap pretty (simplifiedInt e22) -->
                      Just "[(5/3)] . [x^3 - 3x] . ([-1] + log[x^3 - 3x])"
                  ]
```

```haskell
module Int where
import GHC.Real
import Data.List
import Data.Maybe
import Control.Applicative

import Types
import Utilities
import Examples

--
-- Universal assumptions/preconditions:
-- 1. All polynomials are in standard form with decreasing
--    powers of x
-- 2. 0 is represented by P [(0, 0)]; P [] is undefined for
--    the purposes of the exercise.
-- 3. All constants will be polynomials of the form
--    [(c, 0)], e.g. logarithms of constants and constant
--    powers will not appear.
-- 4. All computed integrals omit the constant of integration.
--

------------------------------------------------
-- Part I (13 marks)

-- addP p1 p2
--   =
--   where
--     sums = concat [ [ (coeff1 + coeff2, e2) | (coeff2, e2) <- p2, e2 == e1] | (coeff1, e1) <- p1]

--assumes terms of the same power
-- addTerm :: Term -> Term -> Term
-- addTerm (coeff1, e1) (coeff2, e2)
--   = (coeff1+coeff2, e1)

-- I made helpers to make the process more broken down
-- This helper finds whether a given power exists in a polynomial
-- If it does, it returns it, if not, it returns (0,0)
expVal :: Exponent -> Polynomial -> Term
expVal e1 [] = (0,0)
expVal e1 ((coeff, e2):ps)
  | e1 == e2 = (coeff, e2)
  | otherwise = expVal e1 ps

-- This helper computes the sum between two polynomials for a given power
-- This uses the above functions to find a corresponding power in both polynomials and sums them together
compSum :: Exponent -> Polynomial -> Polynomial -> Term
compSum i p1 p2
  | (coeff1, e1) == (0,0) = (coeff2, e2)
  | (coeff2, e2) == (0,0) = (coeff1, e1)
  | otherwise = (coeff2 + coeff1, e1)
  where
    (coeff1, e1) = expVal i p1
    (coeff2, e2) = expVal i p2

-- This calls the above helper and removes any remnant (0,0) terms that may exist because either power doesn't exist in both polynomials
addP :: Polynomial -> Polynomial -> Polynomial
addP p1 p2 = filter (/= (0,0)) (map (\a -> compSum a p1 p2) iterlist)
  where
    ((coeffs1, es1), (coeffs2, es2)) = (unzip p1, unzip p2)
    maxpow = max (head es1) (head es2)
    iterlist = reverse [0..maxpow]
```

3/4

Very overcomplicated and inefficient


```haskell
mulP :: Polynomial -> Polynomial -> Polynomial
mulP p1 p2
  = mulHelper basecalc basecalc []
  where
    basecalc
      = concat [ [ (coeff1 * coeff2, e2 + e1) | (coeff2, e2) <- p2]
        | (coeff1, e1) <- p1]

mulHelper :: Polynomial -> Polynomial -> [Exponent] -> Polynomial
mulHelper [] _ _ = []
mulHelper ((coeff, e):ps) baselist powers
  | notElem e powers
    =  sumList (matchList (coeff, e) baselist) : mulHelper ps baselist ([e] ++ powers)
  | otherwise = mulHelper ps baselist powers

-- take sum of all polynomials that are of matching power
-- Assumed that powers of all polynomials are the same
sumList :: Polynomial -> Term
sumList [(coeff1, e1)] = (coeff1, e1)
sumList ((coeff1, e1):ps)
  = (coeff1 + coeffout, e1)
  where
    (coeffout, eout) = sumList ps

-- Find a list of terms with matching powers
matchList :: Term -> Polynomial -> Polynomial
matchList t' [] =  []
matchList t'@(coeff, e) ((coeff1, e1):ps)
  | e == e1 = (coeff1, e1) : matchList t' ps
  | otherwise = matchList t' ps

-- groupBy :: Term -> Polynomial -> Polynomial
-- groupBy t@(coeff1, exp1) p'@((coeff2, exp2):ps)
--   |

sumP :: [Polynomial] -> Polynomial
sumP pl
  = foldr1(addP) pl

prodP :: [Polynomial] -> Polynomial
prodP pl
  = foldr1 (mulP) pl

diffT :: Term -> Term
diffT (coeff, 0)
  = (0,0)
diffT (coeff, ex)
  = (coeff * (fromIntegral ex), ex - 1)

intT :: Term -> Term
intT (coeff, ex)
  = (coeff * (1 :% (ex + 1)), ex + 1)

diffP :: Polynomial -> Polynomial
diffP pl
  = map diffT pl

intP :: Polynomial -> Polynomial
intP pl
  = map intT pl

------------------------------------------------
-- Part II (7 marks)

diffE :: Expr -> Expr
```

3/4

1/1

Why foldr1?

2/2

1/1

1/1

```haskell
diffE (P exp)
 = P (diffP exp)
diffE (Add exp1 exp2)
  = Add (diffE exp1) (diffE exp2)
diffE (Mul exp1 exp2)
  =  Add (Mul (exp1) (diffE exp2)) (Mul (diffE exp1) (exp2))
diffE (Pow exp r)
  = Mul (diffE exp) (Mul (toExpr (r)) ((Pow (exp) (r-1))))
diffE (Log exp)
  = Mul (diffE exp) (Pow exp (-1 % 1))

--
-- Given
--
toExpr :: Rational -> Expr
toExpr n = P [(n, 0)]

isConstant :: Expr -> Bool
isConstant (P [(_, 0)]) = True
isConstant _ = False

simplifiedDiff :: Expr -> Expr
simplifiedDiff = simplify . diffE

printDiff :: Expr -> IO ()
printDiff = prettyPrint . simplifiedDiff

-------------------------------------------------
-- Part III (10 marks)

-- type Coefficient = Rational
-- type Exponent = Integer
-- type Term = (Coefficient, Exponent)
-- type Polynomial = [Term]
-- data Expr = P Polynomial
--           | Add Expr Expr
--           | Mul Expr Expr
--           | Pow Expr Rational
--           | Log Expr
--          deriving (Eq, Ord, Show)

intE :: Expr -> Maybe Expr
intE (P exp)
  = Just (P (intP exp))
intE (Add exp1 exp2)
  = Just (Add (fromJust (intE exp1)) (fromJust (intE exp2)))
intE (Mul exp1 exp2)
  | isConstant exp1 = Just (Mul exp1 (fromJust (intE exp2)))
  | isConstant exp2 = Just (Mul (fromJust (intE exp1)) exp2)
  | otherwise = applyICR exp1 exp2
intE (Log exp1)
  = Just (Mul (exp1) (Add (Log exp1) (toExpr (-1 % 1))))
intE exp
  = applyICR (toExpr 1) exp


applyICR :: Expr -> Expr -> Maybe Expr
applyICR e1 e2
  | isConstant e1 || isConstant e2 = intE (P [(1,1)])
  | diffE e1 == e2 = intE (e1)
  | diffE e2 == e1 = intE (e2)
applyICR exp1 a'@(Add exp2 exp3)
  | exp1 == diffE exp2 || exp1 == diffE exp3 = intE a'
applyICR exp1 (Pow exp2 r)
  | diffE exp2 == exp1 = (intE (Pow exp2 r))
applyICR (Pow exp1 r) exp2
```

7/7

Too many parens!

What's the point of using Maybe if you just use fromJust?

1.5

4.5/10

3


```haskell
  | diffE exp1 == exp2 = (intE (Pow exp1 r))
applyICR exp1 (Log exp2)
  | diffE exp2 == exp1 = intE (Log exp2)
applyICR (Log exp1) exp2
  | diffE exp1 == exp2 = intE (Log exp1)



--
-- Given...
--
simplifiedInt :: Expr -> Maybe Expr
simplifiedInt = fmap simplify . intE

printInt :: Expr -> IO ()
printInt e = maybe (putStrLn "Fail") prettyPrint (simplifiedInt e)
```

```haskell
module Examples where

import Types
import Data.Ratio ((%))

p1, p2, p3, p4, p5 :: Polynomial
p1 = [(5,0)]
p2 = [(1,1)]
p3 = [(2,2),(-3,1),(1,0)]
p4 = [(4,1),(-3,0)]
p5 = [(2,3),(-2,1),(2,0)]

x :: Expr
x = P [(1,1)]

-- Basic polynomials
e1, e2, e3, e4, e5 :: Expr
e1 = P p1
e2 = P p2
e3 = P p3
e4 = P p4
e5 = P p5

-- Addition of polynomials
e6 :: Expr
e6 = Add e3 e5

-- Multiplication by constant
e7 :: Expr
e7 = Mul e1 e6

-- Simple functions of x
e8, e9 :: Expr
e8  = Log e2
e9  = Pow e2 (-1)

-- Inverse chain rule, id
e10, e11 :: Expr
e10 = Mul e4 e3
e11 = Mul (Pow x (-1)) (Log x)

-- Inverse chain rule, others
e12, e13 :: Expr
e12 = Mul e4 (Log e3)
e13 = Mul (Pow e3 (3/2)) e4

-- Now with a constant factor
e14, e15 :: Expr
e14 = Mul e4 (Pow (Mul e1 e3) (3/2))
e15 = Mul (P [(1,2), (-1,0)]) (Log (P [(1,3), (-3,1)]))

-- No integral to be found
e16 :: Expr
e16 = Mul (Log e3) (Pow e4 (1/2))

-- Secret testing...
e17, e18, e19, e20, e21, e22 :: Expr
e17 =
  Mul (P [(4 % 5,1),((-3) % 5,0)]) (Pow (Mul (P [(5 % 1,0)]) (P [(2 % 1,2),((-3) % 1,1),(1 % 1,0)])) (3 % 2))
e18 =
  Mul (P [(1 % 5,2),((-1) % 5,0)]) (Log (P [(1 % 1,3),((-3) % 1,1)]))
-- d/dx has factor of 3 => multiply by 1/3
e19 = P [(1,2), (-1,0)]
e20 = P [(1,3), (-3,1)]
e21 = Mul e19 (Log e20)
```



```haskell

-- As above but making 5/3
e22 = Mul (Mul e1 e19) (Log e20)
```

```
    1: -------- Test Output --------
    2: copying hft.cabal from solution
    3: copying cabal.project from solution
    4: copying src/Utilities.hs from solution
    5: Resolving dependencies...
    6: Build profile: -w ghc-9.4.8 -O1
    7: In order, the following will be built (use -v for more details):
    8:  - hft-0.1.0.0 (lib) (first run)
    9: Configuring library for hft-0.1.0.0..
   10: Preprocessing library for hft-0.1.0.0..
   11: Building library for hft-0.1.0.0..
   12: [1 of 4] Compiling Types            ( src/Types.hs, /tmp/d20240125-38-5rjabu/dist-newstyle/build/x86_64-linux/ghc-9.4.8/hft-0.1.0.0/build/Types.o, ⬀
/tmp/d20240125-38-5rjabu/dist-newstyle/build/x86_64-linux/ghc-9.4.8/hft-0.1.0.0/build/Types.dyn_o )
   13: [2 of 4] Compiling Examples         ( src/Examples.hs, /tmp/d20240125-38-5rjabu/dist-newstyle/build/x86_64-linux/ghc-9.4.8/hft-0.1.0.0/build/Examples.o, ⬀
/tmp/d20240125-38-5rjabu/dist-newstyle/build/x86_64-linux/ghc-9.4.8/hft-0.1.0.0/build/Examples.dyn_o )
   14: [3 of 4] Compiling Utilities        ( src/Utilities.hs, /tmp/d20240125-38-5rjabu/dist-newstyle/build/x86_64-linux/ghc-9.4.8/hft-0.1.0.0/build/Utilities.o, ⬀
/tmp/d20240125-38-5rjabu/dist-newstyle/build/x86_64-linux/ghc-9.4.8/hft-0.1.0.0/build/Utilities.dyn_o )
   15: [4 of 4] Compiling Int              ( src/Int.hs, /tmp/d20240125-38-5rjabu/dist-newstyle/build/x86_64-linux/ghc-9.4.8/hft-0.1.0.0/build/Int.o, ⬀
/tmp/d20240125-38-5rjabu/dist-newstyle/build/x86_64-linux/ghc-9.4.8/hft-0.1.0.0/build/Int.dyn_o )
   16: Resolving dependencies...
   17: Build profile: -w ghc-9.4.8 -O1
   18: In order, the following will be built (use -v for more details):
   19:  - hft-0.1.0.0 (lib) (configuration changed)
   20:  - hft-0.1.0.0 (test:hft-test) (first run)
   21: Configuring library for hft-0.1.0.0..
   22: Preprocessing library for hft-0.1.0.0..
   23: Building library for hft-0.1.0.0..
   24: Configuring test suite 'hft-test' for hft-0.1.0.0..
   25: Preprocessing test suite 'hft-test' for hft-0.1.0.0..
   26: Building test suite 'hft-test' for hft-0.1.0.0..
   27: [1 of 2] Compiling IC.TestSuite     ( test/IC/TestSuite.hs, ⬀
/tmp/d20240125-38-5rjabu/dist-newstyle/build/x86_64-linux/ghc-9.4.8/hft-0.1.0.0/t/hft-test/build/hft-test/hft-test-tmp/IC/TestSuite.o )
   28: [2 of 2] Compiling Main             ( test/Tests.hs, ⬀
/tmp/d20240125-38-5rjabu/dist-newstyle/build/x86_64-linux/ghc-9.4.8/hft-0.1.0.0/t/hft-test/build/hft-test/hft-test-tmp/Main.o )
   29: [3 of 3] Linking /tmp/d20240125-38-5rjabu/dist-newstyle/build/x86_64-linux/ghc-9.4.8/hft-0.1.0.0/t/hft-test/build/hft-test/hft-test
   30: Resolving dependencies...
   31: Build profile: -w ghc-9.4.8 -O1
   32: In order, the following will be built (use -v for more details):
   33:  - hft-0.1.0.0 (lib) (configuration changed)
   34:  - hft-0.1.0.0 (test:hft-test) (configuration changed)
   35: Configuring library for hft-0.1.0.0..
   36: Preprocessing library for hft-0.1.0.0..
   37: Building library for hft-0.1.0.0..
   38: Configuring test suite 'hft-test' for hft-0.1.0.0..
   39: Preprocessing test suite 'hft-test' for hft-0.1.0.0..
   40: Building test suite 'hft-test' for hft-0.1.0.0..
   41: Running 1 test suites...
   42: Test suite hft-test: RUNNING...
   43: addP: 3 / 3
   44:
   45: mulP: 1 / 1
   46:
   47: sumP: 2 / 2
   48:
   49:  Failure in prodP (test/Tests.hs:44):
   50:  expected: "[4x^5 - 6x^4 - 2x^3 + 10x^2 - 8x + 2]"
   51:   but got: "[4x^5 - 2x^3 + 10x^2 - 6x^4 - 8x + 2]"
   52:
   53: prodP: 0 / 1
   54:
   55: diffT: 2 / 2
```

```
intT: 2 / 2

diffP: 1 / 1

intP: 1 / 1

diffE: 3 / 3

 Failure in intE (test/Tests.hs:91):
 expected: Just "log[x]"
  but got: Just "[(1/2)x^2]"

 Failure in intE (test/Tests.hs:93):
 expected: Just "[(1/2)] . [4x^4 - 12x^3 + 13x^2 - 6x + 1]"
  but got: src/Int.hs:(186,1)-(199,40): Non-exhaustive patterns in function applyICR


 Failure in intE (test/Tests.hs:95):
 expected: Just "[(1/2)] . (log[x])^2"
  but got: src/Int.hs:(186,1)-(199,40): Non-exhaustive patterns in function applyICR


 Failure in intE (test/Tests.hs:97):
 expected: Just "[2x^2 - 3x + 1] . ([-1] + log[2x^2 - 3x + 1])"
  but got: src/Int.hs:(186,1)-(199,40): Non-exhaustive patterns in function applyICR


 Failure in intE (test/Tests.hs:99):
 expected: Just "[(2/5)] . ([2x^2 - 3x + 1])^(5/2)"
  but got: src/Int.hs:(186,1)-(199,40): Non-exhaustive patterns in function applyICR


 Failure in intE (test/Tests.hs:101):
 expected: Just "[(2/25)] . ([5] . [2x^2 - 3x + 1])^(5/2)"
  but got: src/Int.hs:(186,1)-(199,40): Non-exhaustive patterns in function applyICR


 Failure in intE (test/Tests.hs:103):
 expected: Just "[(1/3)] . [x^3 - 3x] . ([-1] + log[x^3 - 3x])"
  but got: src/Int.hs:(186,1)-(199,40): Non-exhaustive patterns in function applyICR


 Failure in intE (test/Tests.hs:105):
 expected: Nothing
  but got: src/Int.hs:(186,1)-(199,40): Non-exhaustive patterns in function applyICR


intE: 8 / 16

 Failure in intESecret (test/Tests.hs:110):
 expected: Just "[(2/125)] . ([5] . [2x^2 - 3x + 1])^(5/2)"
  but got: src/Int.hs:(186,1)-(199,40): Non-exhaustive patterns in function applyICR


 Failure in intESecret (test/Tests.hs:112):
 expected: Just "[(1/15)] . [x^3 - 3x] . ([-1] + log[x^3 - 3x])"
  but got: src/Int.hs:(186,1)-(199,40): Non-exhaustive patterns in function applyICR


 Failure in intESecret (test/Tests.hs:118):
```

```
  expected: Just "[(1/3)] . [x^3 - 3x] . ([-1] + log[x^3 - 3x])"
   but got: src/Int.hs:(186,1)-(199,40): Non-exhaustive patterns in function applyICR


 Failure in intESecret (test/Tests.hs:120):
 expected: Just "[(5/3)] . [x^3 - 3x] . ([-1] + log[x^3 - 3x])"
  but got: src/Int.hs:(186,1)-(199,40): Non-exhaustive patterns in function applyICR


intESecret: 2 / 6


Test suite hft-test: FAIL
Test suite logged to:
/tmp/d20240125-38-5rjabu/dist-newstyle/build/x86_64-linux/ghc-9.4.8/hft-0.1.0.0/t/hft-test/test/hft-0.1.0.0-hft-test.log
0 of 1 test suites (0 of 1 test cases) passed.
copying test from solution
copying src/Examples.hs from solution
Resolving dependencies...
Build profile: -w ghc-9.4.8 -O1
In order, the following will be built (use -v for more details):
 - hft-0.1.0.0 (lib) (configuration changed)
 - hft-0.1.0.0 (test:hft-test) (configuration changed)
Configuring library for hft-0.1.0.0..
Preprocessing library for hft-0.1.0.0..
Building library for hft-0.1.0.0..
Configuring test suite 'hft-test' for hft-0.1.0.0..
Preprocessing test suite 'hft-test' for hft-0.1.0.0..
Building test suite 'hft-test' for hft-0.1.0.0..
Resolving dependencies...
Build profile: -w ghc-9.4.8 -O1
In order, the following will be built (use -v for more details):
 - hft-0.1.0.0 (lib) (configuration changed)
 - hft-0.1.0.0 (test:hft-test) (configuration changed)
Configuring library for hft-0.1.0.0..
Preprocessing library for hft-0.1.0.0..
Building library for hft-0.1.0.0..
Configuring test suite 'hft-test' for hft-0.1.0.0..
Preprocessing test suite 'hft-test' for hft-0.1.0.0..
Building test suite 'hft-test' for hft-0.1.0.0..
Running 1 test suites...
Test suite hft-test: RUNNING...
addP: 3 / 3

mulP: 1 / 1

sumP: 2 / 2

 Failure in prodP (test/Tests.hs:44):
 expected: "[4x^5 - 6x^4 - 2x^3 + 10x^2 - 8x + 2]"
  but got: "[4x^5 - 2x^3 + 10x^2 - 6x^4 - 8x + 2]"

prodP: 0 / 1

diffT: 2 / 2

intT: 2 / 2

diffP: 1 / 1

intP: 1 / 1
```

```
178:
179: diffE: 3 / 3
180:
181:  Failure in intE (test/Tests.hs:91):
182:  expected: Just "log[x]"
183:   but got: Just "[(1/2)x^2]"
184:
185:  Failure in intE (test/Tests.hs:93):
186:  expected: Just "[(1/2)] . [4x^4 - 12x^3 + 13x^2 - 6x + 1]"
187:   but got: src/Int.hs:(186,1)-(199,40): Non-exhaustive patterns in function applyICR
188:
189:
190:  Failure in intE (test/Tests.hs:95):
191:  expected: Just "[(1/2)] . (log[x])^2"
192:   but got: src/Int.hs:(186,1)-(199,40): Non-exhaustive patterns in function applyICR
193:
194:
195:  Failure in intE (test/Tests.hs:97):
196:  expected: Just "[2x^2 - 3x + 1] . ([-1] + log[2x^2 - 3x + 1])"
197:   but got: src/Int.hs:(186,1)-(199,40): Non-exhaustive patterns in function applyICR
198:
199:
200:  Failure in intE (test/Tests.hs:99):
201:  expected: Just "[(2/5)] . ([2x^2 - 3x + 1])^(5/2)"
202:   but got: src/Int.hs:(186,1)-(199,40): Non-exhaustive patterns in function applyICR
203:
204:
205:  Failure in intE (test/Tests.hs:101):
206:  expected: Just "[(2/25)] . ([5] . [2x^2 - 3x + 1])^(5/2)"
207:   but got: src/Int.hs:(186,1)-(199,40): Non-exhaustive patterns in function applyICR
208:
209:
210:  Failure in intE (test/Tests.hs:103):
211:  expected: Just "[(1/3)] . [x^3 - 3x] . ([-1] + log[x^3 - 3x])"
212:   but got: src/Int.hs:(186,1)-(199,40): Non-exhaustive patterns in function applyICR
213:
214:
215:  Failure in intE (test/Tests.hs:105):
216:  expected: Nothing
217:   but got: src/Int.hs:(186,1)-(199,40): Non-exhaustive patterns in function applyICR
218:
219:
220: intE: 8 / 16
221:
222:  Failure in intESecret (test/Tests.hs:110):
223:  expected: Just "[(2/125)] . ([5] . [2x^2 - 3x + 1])^(5/2)"
224:   but got: src/Int.hs:(186,1)-(199,40): Non-exhaustive patterns in function applyICR
225:
226:
227:  Failure in intESecret (test/Tests.hs:112):
228:  expected: Just "[(1/15)] . [x^3 - 3x] . ([-1] + log[x^3 - 3x])"
229:   but got: src/Int.hs:(186,1)-(199,40): Non-exhaustive patterns in function applyICR
230:
231:
232:  Failure in intESecret (test/Tests.hs:118):
233:  expected: Just "[(1/3)] . [x^3 - 3x] . ([-1] + log[x^3 - 3x])"
234:   but got: src/Int.hs:(186,1)-(199,40): Non-exhaustive patterns in function applyICR
235:
236:
237:  Failure in intESecret (test/Tests.hs:120):
238:  expected: Just "[(5/3)] . [x^3 - 3x] . ([-1] + log[x^3 - 3x])"
```

```
  but got: src/Int.hs:(186,1)-(199,40): Non-exhaustive patterns in function applyICR


intESecret: 2 / 6


Test suite hft-test: FAIL
Test suite logged to:
/tmp/d20240125-38-5rjabu/dist-newstyle/build/x86_64-linux/ghc-9.4.8/hft-0.1.0.0/t/hft-test/test/hft-0.1.0.0-hft-test.log
0 of 1 test suites (0 of 1 test cases) passed.

-------- Test Errors --------
Error: cabal: Tests failed for test:hft-test from hft-0.1.0.0.

Error: cabal: Tests failed for test:hft-test from hft-0.1.0.0.

```