```haskell
import Test.Hspec
import Grad
import System.TimeIt



main :: IO ()
main = hspec $ do

  describe "Testing Gradient Descent" $ do
    it "Parallel - 100000 rows" $ do
        csvData <- getCSVData "data/test-5.csv"
        output <- timeItT $ (descendSteps "parallel" 64 csvData computeGradRowLinear [0,0] (1000::Int)
(0.00000000000000001::Double)) `seq` return ()
        let computeTime = fst output
        print $ computeTime
        computeTime `shouldSatisfy` (<=(300.0::Double))


    it "Parallel - 10000 rows" $ do
        csvData <- getCSVData "data/test-4.csv"
        output <- timeItT $ (descendSteps "parallel" 64 csvData computeGradRowLinear [0,0] (1000::Int)
(0.0000000000001::Double)) `seq` return ()
        let computeTime = fst output
        print $ computeTime
        computeTime `shouldSatisfy` (<=(10.0::Double))


    it "Sequential - 10000 rows" $ do
        csvData <- getCSVData "data/test-4.csv"
        output <- timeItT $ (descendSteps "sequential" 64 csvData computeGradRowLinear [0,0] (1000::Int)
(0.0000000000001::Double)) `seq` return ()
        let computeTime = fst output
        print $ computeTime
        computeTime `shouldSatisfy` (<=(300.0::Double))


    it "Parallel - 1000 rows" $ do
        csvData <- getCSVData "data/test-3.csv"
        output <- timeItT $ (descendSteps "parallel" 64 csvData computeGradRowLinear [0,0] (1000::Int) (0.0000000001::Double))
`seq` return ()
        let computeTime = fst output
        print $ computeTime
        computeTime `shouldSatisfy` (<=(10.0::Double))


    it "Sequential - 1000 rows" $ do
        csvData <- getCSVData "data/test-3.csv"
        output <- timeItT $ (descendSteps "sequential" 64 csvData computeGradRowLinear [0,0] (1000::Int)
(0.0000000001::Double)) `seq` return ()
        let computeTime = fst output
        print $ computeTime
        computeTime `shouldSatisfy` (<=(300.0::Double))


    it "Parallel - 100 rows" $ do
        csvData <- getCSVData "data/test-2.csv"
        output <- timeItT $ (descendSteps "parallel" 64 csvData computeGradRowLinear [0,0] (1000::Int) (0.0000001::Double))
`seq` return ()
        let computeTime = fst output
        print $ computeTime
        computeTime `shouldSatisfy` (<=(10.0::Double))


    it "Sequential - 100 rows" $ do
```

```haskell
        csvData <- getCSVData "data/test-2.csv"
        output <- timeItT $ (descendSteps "sequential" 64 csvData computeGradRowLinear [0,0] (1000::Int) (0.0000001::Double))
`seq` return ()
        let computeTime = fst output
        print $ computeTime
        computeTime `shouldSatisfy` (<=(10.0::Double))
```