Assignment #3

Question 1: Implement Q3 of HW1 (word size count for file Alice.txt) in Spark and compare the performance with Hadoop MapReduce.

```
val file = sc.textFile("alice.txt")
val counts = file.flatMap(line => line.split(" ")).map(word => (word, 1)).
    reduceByKey(_ + _)
counts.saveAsTextFile("alice_output")
```

Compared to MapReduce, Spark completed these commands in an instant. This is because Spark RDDs can handle multiple map operations at once. It processes data in real time and faster unlike Hadoop.

Question 2: Given a travel data with the following format, use Spark RDD manipulations to answer the questions.

```
Column1: City pair (Combination of from and to ) : String
Column2: From location : String
Column3: To Location : String
Column4: Product type : Integer (1=Air, 2=Car, 3=Air+Car, 4=Hotel,
5=Air+Hotel, 6=Hotel+Car, 7=Air+Hotel+Car)
Column5: Adults traveling : Integer
Column6: Seniors traveling : Integer
Column7: Children traveling : Integer
Column8: Youth traveling : Integer
Column9: Infant traveling : Integer
Column10: Booking Date : Date
Column11: Boarding Date : Date
```

1. Find top 20 destination people travel the most.

```
val traveldata = sc.textFile("traveldata.txt")
val splitOne = traveldata.map(lines => lines.split('\t')).map(x=>(x(2), 1)).
    reduceByKey(_+_)
val splitOne_ordered = splitOne.map(x => x.swap).sortByKey(false)
val quesOne = splitOne_ordered.map(x => x.swap).take(20)
```

2. Find top 20 locations from where people travel the most.

```
val splitTwo = traveldata.map(lines => lines.split('\t')).map(x=>(x(1),1)).
    reduceByKey(_+_)
val splitTwo_ordered = splitTwo.map(x => x.swap).sortByKey(false)
val quesTwo = splitTwo_ordered.map(x => x.swap).take(20)
```

3. Find top 20 cities that generate high airline revenues for travel, so that the site can concentrate on offering discount on booking to those cities to attract more bookings.

```
val airlinedata = traveldata.map(lines => lines.split('\t')).filter(x => {if ((
    x(3).matches(("1")))) true else false})
val splitThree = airlinedata.map(x => (x(2), 1)).reduceByKey(_+_)
val splitThree_ordered = splitThree.map(x => x.swap).sortByKey(false)
val quesThree = splitThree_ordered.map(x => x.swap).take(20)
```

Question 3: For the given employee data file (EMPSAL.txt), write a Scala program which calculates he minimum as well as the maximum salaries per department and saves the output in a file. (For the output, create any data with two columns, department and salary, for it.)

```
import org.apache.spark.sql.types._
import org.apache.spark.sql.Row

def moneyToFloat (money: String): Float = {
        money.replace("$", "").replace(",","").toFloat
}

def transformToRow (input: (String, (Float, Float))): Row = {
        Row(input._1, input._2._1, input._2._2)
}

val schema =
        StructType (
                StructField("Department", StringType, false)::
                StructField("MinimumSalary", FloatType, false)::
                StructField("MaximumSalary", FloatType, false) :: Nil)

val empsal = sc.textFile("empsal.txt")
val getMaxs = empsal.map(lines=>lines.split('\t')).map(x=>(x(1), moneyToFloat(x(5)))
    ).reduceByKey(math.max(_,_))
val getMins = empsal.map(lines=>lines.split('\t')).map(x=>(x(1), moneyToFloat(x(5)))
    ).reduceByKey(math.min(_,_))
val sortMaxs = getMaxs.sortByKey()
val sortMins = getMins.sortByKey()
val joined = sortMins.join(sortMaxs)
val together = joined.map(transformToRow)
val df = spark.createDataFrame(together, schema)

df.coalesce(1).write.format("com.databricks.spark.csv").option("header", "true").
    save("deptMinMaxSalary")
```

Question 4: Airports data file (airports.csv) contains the following fields:

Field	Description
Airport ID	Unique OpenFlights identifier for this airport.
Name	Name of airport. May or may not contain the City name.
City	Main city served by airport. May be spelled differently from Name.
Country	Country or territory where airport is located.
IATA/FAA	3-letter FAA code, for airports locate in “United States of America.” 3-letter IATA code, for all other airports. Blank if not assigned.
ICAO	4-letter ICAO code. Blank if not assigned.
Latitude	Decimal degrees, usually to 6 significant digits. Negative is South, positive is North.
Longitude	Decimal degrees, usually to 6 significant digits. Negative is West, positive is East.
Altitude	In feet.
Timezone	Hours offset from UTC. Fractional hours are expressed in decimals.
DST	Daylight savings time. One of E (Europe), A (US/Canada), S (South America), O (Australia), Z (New Zealand), N (None) or U (Unknown).
timezone	Timezone in “tz” (Olsen) format

Use SparkSQL analysis to answer the following questions:

```
val sqlcontext = new org.apache.spark.sql.SQLContext(sc)
val airports = sqlcontext.read.format("csv").option("header", "true").option("
    inferSchema", "true").load("airports.csv")
airports.createOrReplaceTempView("df")
```

1. Print the schema of the DataFrame created.

   ```
   airports.printSchema()
   ```

2. How many airports are there in South east part of the dataset.

   ```
   val q2 = spark.sql("SELECT AirportID FROM df WHERE Latitude < 0 AND Longitude >
       0")
   q2.show
   ```

3. How many unique cities have airports in each country?

   ```
   val q3 = spark.sql("SELECT Country, COUNT(*) City FROM df GROUP BY Country
       ORDER BY Country")
   q3.show
   ```

4. What is the average Altitude (in feet) of airports in each Country?

   ```
   val q4 = spark.sql("SELECT Country, AVG(Altitude) FROM df GROUP BY Country
       ORDER BY Country")
   q4.show
   ```

5. How many airports are operating in each timezone?

   ```
   val q5 = spark.sql("SELECT Timezone, COUNT(*) Airport FROM df GROUP BY Timezone
       ORDER BY Timezone")
   q5.show
   ```

6. Calculate average latitude and longitude for these airports in each country.

   ```
   val q6 = spark.sql("SELECT Country, AVG(Latitude), AVG(Longitude) FROM df GROUP
       BY Country ORDER BY Country")
   q6.show
   ```

7. How many different DSTs are there?

   ```
   val q7 = spark.sql("SELECT COUNT(DISTINCT DST) FROM df")
   q7.show
   ```

Question 6 from HW 2: In mathematics, the least common multiple (LCM) of two numbers is the smallest positive integer that can be divided by the two numbers without producing a remainder. LCM can be calculated as follows:

$$LCM(a.b) = \frac{a \cdot b}{GCD(a,b)}$$

where $GCD(a,b)$ is the greatest common divisor of a and b, i.e., the largest number that divides both of them without leaving a remainder. Write a Scala program to implement a function to calculate $LCM(a,b)$ using Higher Order Functions.

```
object Question6 extends App{

        println("The LCM of 10 and 49 is: " + lcm(10, 40))
        println("The LCM of 65 and 30 is: " + lcm(65, 30))
        println("The LCM of 3 and 5 is: " + lcm(3,5))
        println("The LCM of 6 and 3 is: " + lcm(6, 3))
        println("The LCM of 12 and 48 is: " + lcm(12, 48))

        def gcd(a: Int, b: Int): Int = {
                if(b == 0) a
                else gcd(b, a % b)
        }

        def lcm(a: Int, b: Int): Int = (a * b) / gcd(a, b)
}
```