Scala Programs

1. Write the scala code to implement bubble sort algorithm.

```
def bubbleSort(arr: Array[Int]): Array[Int] = {
  for (i <- 0 until arr.length - 1) {
    for (j <- 0 until arr.length - i - 1) {
      if (arr(j) > arr(j + 1)) {
        val temp = arr(j)
        arr(j) = arr(j + 1)
        arr(j + 1) = temp
      }
    }
  }
  arr
}

// Example usage:
val arr = Array(64, 34, 25, 12, 22, 11, 90)
println(bubbleSort(arr).mkString(", "))
```

2. Write scala code to find the length of each word from the array.

```
def wordLengths(words: Array[String]): Array[Int] = {
  words.map(_.length)
}

// Example usage:
val words = Array("apple", "banana", "cherry")
println(wordLengths(words).mkString(", "))
```

OR

```
val words = Array("apple", "banana", "cherry")
for(i <- 0 until words.length){
  println(words(i).length)
}
```

OR

```
val words = Array("apple", "banana", "cherry")
val res = words.map(_.length)
res.foreach(println)
```

3. Write scala code to find the number of books published by each author, referring to the collection given below, with (authorName, BookName) { (‘ Dr. Seuss’: ‘How the Grinch Stole Christmas!’) ,(‘ Jon Stone’: ‘Monsters at the End of This Book’) , (‘ Dr. Seuss’: ‘The Lorax’) ,(‘Jon Stone’: ‘Big Bird in China’) (‘ Dr. Seuss’ :’ One Fish, Two Fish, Red Fish, Blue Fish’) }

```
val books = List(
("Dr. Seuss", "How the Grinch Stole Christmas!"),
("Jon Stone", "Monsters at the End of This Book"),
("Dr. Seuss", "The Lorax"),
("Jon Stone", "Big Bird in China"),
("Dr. Seuss", "One Fish, Two Fish, Red Fish, Blue Fish")
)
val booksByAuthor = books.groupBy(_._1).map { case (author, books) =>
(author, books.size) }
println(booksByAuthor)
```

OR

```
// Group books by author using an explicitly defined function
val groupedBooks = books.groupBy(tuple => tuple._1)
// Map the grouped values to their sizes (count of books per author)
val bookCount = groupedBooks.mapValues(bookList => bookList.size)
```

4. Write the program to illustrate the use of pattern matching in scala, for the following
   Matching on case classes.
   Define two case classes as below: abstract class Notification

case class Email(sender: String, title: String, body: String) extends Notification case class SMS(caller: String, message: String) extends Notification Define a function showNotification which takes as a parameter the abstract type Notification and matches on the type of Notification (i.e. it figures out whether it's an Email or SMS). In the case it's an Email(email, title, _) return the string: s"You got an email from $email with title: $title" In the case it's an SMS return the String: s"You got an SMS from $number! Message: $message"

```
abstract class Notification
case class Email(sender: String, title: String, body: String) extends
Notification
case class SMS(caller: String, message: String) extends Notification
def showNotification(notification: Notification): String = {
  notification match {
    case Email(sender, title, _) => s"You got an email from $sender with
title: $title"
    case SMS(caller, message) => s"You got an SMS from $caller! Message:
$message"
  }
}
// Example usage:
val someEmail = Email("john.doe@example.com", "Meeting Reminder",
"Don't forget our meeting at 10 AM.")
val someSMS = SMS("123-456-7890", "Hi, are you available for a call?")
println(showNotification(someEmail))
println(showNotification(someSMS))
```

5. Write the scala program using imperative style to implement quick sort algorithm.

```
def quickSortImperative(arr: Array[Int]): Array[Int] = {
  def swap(i: Int, j: Int): Unit = {
    val temp = arr(i)
    arr(i) = arr(j)
    arr(j) = temp
  }
```

```
  def partition(low: Int, high: Int): Int = {
    val pivot = arr(high)
    var i = low - 1
    for (j <- low until high) {
      if (arr(j) <= pivot) {
        i += 1
        swap(i, j)
      }
    }
    swap(i + 1, high)
    i + 1
  }

  def sort(low: Int, high: Int): Unit = {
    if (low < high) {
      val pi = partition(low, high)
      sort(low, pi - 1)
      sort(pi + 1, high)
    }
  }

  sort(0, arr.length - 1)
  arr
}

// Example usage:
val arr = Array(10, 7, 8, 9, 1, 5)
println(quickSortImperative(arr).mkString(", "))
```

6. Write a scala function to convert the each word to capitalize each word in the given sentence.

```
def capitalizeWords(sentence: String): String = {
  sentence.split(" ").map(_.capitalize).mkString(" ")
}
```

```
// Example usage:
val sentence = "hello world from scala"
println(capitalizeWords(sentence))
```

7. Write scala code to show functional style program to implement quick sort algorithm.

```
def quickSortFunctional(arr: List[Int]): List[Int] = {
 if (arr.length <= 1) arr
 else {
  val pivot = arr(arr.length / 2)
  quickSortFunctional(arr.filter(_ < pivot)) ++ arr.filter(_ == pivot) ++
quickSortFunctional(arr.filter(_ > pivot))
 }
}

// Example usage:
val list = List(10, 7, 8, 9, 1, 5)
println(quickSortFunctional(list).mkString(", "))
```

8. For the below given collection of items with item-names and quantity, write the scala code for the given statement.
   Items = {(“Pen”:20), (“Pencil”:10), (“Erasor”:7), (“Book”:25), (“Sheet”:15)}
   i. Display item-name and quantity
   ii. Display sum of quantity and total number of items
   iii. Add 3 Books to the collection

   Add new item “Board” with quantity 15 to the collection

```
val items = List(("Pen", 20), ("Pencil", 10), ("Eraser", 7), ("Book", 25),
("Sheet", 15))

// i. Display item-name and quantity
items.foreach { case (name, quantity) => println(s"$name: $quantity") }
```

```
// ii. Display sum of quantity and total number of items
val totalQuantity = items.map(_._2).sum
val totalItems = items.size

// iii. Add 3 Books to the collection
val updatedItems = items.map {
  case ("Book", quantity) => ("Book", quantity + 3)
  case other => other
}

// Add new item "Board" with quantity 15 to the collection
val finalItems = updatedItems :+ ("Board", 15)
finalItems.foreach { case (name, quantity) => println(s"$name: $quantity") }
```

9. Develop a scala code to search an element in the given list of numbers. The function Search() will take two arguments: list of numbers and the number to be searched. The function will write True if the number is found, False otherwise.

```
def searchElement(numbers: List[Int], target: Int): Boolean = {
  numbers.contains(target)
}

// Example usage:
val numbers = List(1, 2, 3, 4, 5)
val target = 3
println(searchElement(numbers, target))
```

10.Illustrate the implementation by writing scala code to generate a down counter from 10 to 1.

```
(10 to 1 by -1).foreach(println)
```

11.Design a scala function to perform factorial item in the given collection. The arguments passed to the function are the collection of items. Assume the type of the argument for the function suitably. The return type is to be integer.

```
def factorial(n: Int): Int = {
 if (n <= 1) 1
 else n * factorial(n - 1)
}

// Example usage:
val items = List(2, 3, 4, 5)
val factorials = items.map(factorial)
println(factorials)
```

12.For the below given collection of items with item names and quantity, write the scala code for the given statement. Items = {(“Butter”:20), (“Bun”:10), (“Egg”:7), (“Biscuit”:25),(“Bread”:15)}

i. Display item-name and quantity
ii. Display sum of quantity and total number of items
iii. Add 3 Buns to the collection
iv. Add new item “Cheese” with quantity 12 to the collection

```
val items = Map("Butter" -> 20, "Bun" -> 10, "Egg" -> 7, "Biscuit" -> 25, "Bread" -> 15)

// i. Display item-name and quantity

items.foreach { case (name, quantity) => println(s"$name: $quantity") }

// ii. Display sum of quantity and total number of items
```

```
val totalQuantity = items.values.sum

val totalItems = items.size

// iii. Add 3 Buns to the collection

val updatedItems = items.updated("Bun", items("Bun") + 3)

// iv. Add new item "Cheese" with quantity 12 to the collection

val finalItems = updatedItems + ("Cheese" -> 12)

finalItems.foreach { case (name, quantity) => println(s"$name:
$quantity") }
```

13. Implement function for binary search using recursion in Scala to find the number, given a list of numbers. The function will have two arguments: Sorted list of numbers and the number to be searched.

```
def binarySearch(arr: Array[Int], target: Int): Boolean = {
 def search(low: Int, high: Int): Boolean = {
  if (low > high) false
  else {
   val mid = (low + high) / 2
   if (arr(mid) == target) true
   else if (arr(mid) > target) search(low, mid - 1)
   else search(mid + 1, high)
  }
 }
 search(0, arr.length - 1)
}

// Example usage:
val sortedList = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
val numberToSearch = 5
println(binarySearch(sortedList, numberToSearch))
```

14. Write a function to find the length of each word and return the word with highest length .Ex for the collection of words = (“games”, “television”,”rope”,”table”) The function should return (“television”,10). The word with the highest length .Read the words from the keyboard.

```
def findLongestWord(words: Array[String]): (String, Int) = {
  words.map(word => (word, word.length)).maxBy(_._2)
}

// Example usage:
val words = Array("games", "television", "rope", "table")
val longestWord = findLongestWord(words)
println(longestWord)
```

Spark Programs

15. Analyze the application of fold() and aggregate() functions in Spark by considering a scenario where all the items in a collection are updated by a count of 100. Evaluate the efficiency, performance, and suitability of both.

Source code:

```
import org.apache.spark.{SparkConf, SparkContext}
object SumWithFoldAggregate {
 def main(args: Array[String]): Unit = {
 val conf = new
SparkConf().setAppName("SumWithFoldAggregate").setMaster("loc
al[*]")
 val sc = new SparkContext(conf)
 val data = List(1, 2, 3, 4, 5)
 val rc = sc.parallelize(data)
 // Using fold
 val updatedFold = rc.fold(0)((a, b) => a + b + 100)
 // Using aggregate
 val updatedAggr = rc.aggregate(0)((a, b) => a + b + 100, (a1, a2) =>
a1 + a2)
 println("Using fold: " + updatedFold)
 println("Using aggregate: " + updatedAggr)
 sc.stop()
 }
}
```

SBT file:

```
name := "SumWithFoldAggregate"
version := "0.1"
scalaVersion := "2.12.14"
libraryDependencies += "org.apache.spark" %% "spark-core" %
"3.2.0"
```

Result:

```
Using fold: 1315
Using aggregate: 51
```

16. Consider a text file text.txt. Develop Spark code to read the file and count the number of occurrences of each word using Spark RDD. Store the result in a file. Display the words which appear more than 4 times.

Source code

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkConf
import org.apache.spark.rdd.RDD
object wordcount {
 def main(args: Array[String]) {
 val pathToFile = "log.txt"
 val conf = new
SparkConf().setAppName("Wordcount1").setMaster("local[*]")
 val sc = new SparkContext(conf)

 val wordsRdd = sc.textFile(pathToFile).flatMap(_.split(" "))

 val wordCountInitRdd = wordsRdd.map(word => (word, 1))

 val wordCountRdd = wordCountInitRdd.reduceByKey((v1, v2) => v1
+ v2)

 val highFreqWords = wordCountRdd.filter(x => x._2 > 4)

 highFreqWords.saveToFile("newFolfer")
 sc.stop()
 }
}
```

SBT file:

```
name := "WordCountApp"
version := "0.1"
scalaVersion := "2.12.18"
libraryDependencies ++= Seq(
 "org.apache.spark" %% "spark-core" % "3.5.1"
)
fork in run := true
```

```
javaOptions in run ++= Seq(
 "--add-exports=java.base/sun.nio.ch=ALL-UNNAMED",
 "--add-opens=java.base/java.nio=ALL-UNNAMED",
 "--add-opens=java.base/java.lang.invoke=ALL-UNNAMED"
)
```

Log file:

```
This is it
it is This
This is nice
it is also nice
This is me
is it This
```

17. Consider the content of text file text.txt. Perform the counting of occurrences of each word using pair RDD.

Source code:

```
import org.apache.spark.{SparkConf, SparkContext}
object WordCount {
 def main(args: Array[String]): Unit = {
 val conf = new
SparkConf().setAppName("WordCount").setMaster("local[*]")
 val sc = new SparkContext(conf)
 // Path to the text file
 val pathToFile = "text.txt"
 // Read the text file and create an RDD of lines
 val linesRdd = sc.textFile(pathToFile)
 // Split each line into words and flatten the result
 val wordsRdd = linesRdd.flatMap(_.split("\\s+"))
 // Map each word to a tuple (word, 1) for counting
 val wordCountRdd = wordsRdd.map(word => (word, 1))
 // Reduce by key to get the count of each word
 val wordCounts = wordCountRdd.reduceByKey(_ + _)
 // Print the word counts
 wordCounts.foreach(println)
 sc.stop()
 }
```

```
}
```

SBT file:

```
name := "WordCount"
version := "0.1"
scalaVersion := "2.12.14"
libraryDependencies += "org.apache.spark" %% "spark-core" %
"3.2.0"
```

Text.txt file:

```
This is it
it is This
This is nice
it is also nice
This is me
is it This
```

18. Write the Spark Code to print the top 10 tweeters.
Tweet Mining: A dataset with the 8198 reduced tweets, reduced-tweets.json will be provided. The data contains reduced tweets as in the sample below:\
"id":"572692378957430785",
"user":"Srkian_nishu smile",
"text":"@always_nidhi @YouTube no idnt understand bti loved of this mve is rocking",
"place":"Orissa",
"country":"India"}
A function to parse the tweets into an RDD will be provided.

```
object tweetmining {

 // Create the spark configuration and spark context
 val conf = new SparkConf()
  .setAppName("User mining")
  .setMaster("local[*]")

 val sc = new SparkContext(conf)
```

```
  // Needs an argument which is the path to the tweets file in json
format.
  // e.g. "/home/sparkProjects/data/reduced-tweets.json"

  var pathToFile = ""

  def main(args: Array[String]) {

    if (args.length != 1) {
      println()
      println("But you have given me " + args.length +".")
      println("The argument should be path to json file containing a
bunch of tweets. desired.")
      System.exit(1)
    }

    pathToFile = args(0)

    // The code below creates an RDD of tweets. Please look at the
case class Tweet towards the end
    // of this file.

    val tweets =

sc.textFile(pathToFile).mapPartitions(TweetUtils.parseFromJson(_))
    val tweetsByUser = tweets.map(x => (x.user,x) ).groupByKey()

    val numTweetsByUser = tweetsByUser.map(x=> (x._1, x._2.size))

    val sortedUsersByNumTweets = numTweetsByUser.sortBy(_._2,
ascending=false)


    // Create an RDD of (user, Tweet).
    // TODO: Look at the tweet class. An object of type tweet will
have fields "id", "user", "userName"
```

```
   // etc. Collect all his tweets by the user. For this, you can use the
field "user" of an object
   // in the tweet RDD.
   // Hint: the Spark API provides a groupBy method
   // The code below should return RDD's with tuples (user, List of
user tweets).

   // TODO: For each user, find the number of tweets he/she has
made.
   // Hint: we need tuples of (user, number of tweets by user)

   // TODO: Sort the users by the number of their tweets.

   // Find the Top 10 twitterers and print them to the console.

   sortedUsersByNumTweets.take(10).foreach(println)

   val selectedTweets= sortedUsersByNumTweets.take(10)
//    selectedTweets.saveAsTextFile("tweetsDir")

  }
}

import com.google.gson._

object TweetUtils {
   case class Tweet (
            id : String,
            user : String,
            userName : String,
            text : String,
            place : String,
            country : String,
            lang : String
            )
```

```
  def parseFromJson(lines:Iterator[String]):Iterator[Tweet] = {
        val gson = new Gson
        lines.map(line => gson.fromJson(line, classOf[Tweet]))
  }
}
```

19. Simulate the following scenario using Spark streaming. There will be a process which will be streaming lines of text to a unix port using socket communication. The process we can use for this purpose is netcat. It will stream lines typed on the console to a unix socket. The spark application needs to read the lines from the specified port, and it needs to produce the word counts on the console. A batch interval of 5 second can be used.

spark-shell

```
scala> import org.apache.spark.streaming.{StreamingContext, Seconds}
scala> val ssc =new StreamingContext(sc, Seconds (2))
scala> val streams= ssc.socketTextStream("localhost", 4444, org.apache.spark.storage.StorageLevel.MEMORY_ONLY)
scala> val wordcount =streams.flatMap(_.split(" ")).map(w=> (w,1)).reduceByKey(_+_)
scala> wordcount.print
scala> ssc.start
```

Terminal 2:

```
nc -lk 4444
```

20. Develop the spark code to find the average of marks using the combineByKey() operation.
Sample Input format: Array(("Joe", "Maths", 83), ("Joe", "Physics", 74), ("Joe", "Chemistry", 91), ("Joe", "Biology", 82), ("Nik", "Maths", 69), ("Nik ", "Physics", 62), ("Nik ", "Chemistry", 97), ("Nik ", "Biology", 80))

```
import org.apache.spark.sql.SparkSession
```

```
object AverageMarks {
  def main(args: Array[String]): Unit = {
    val spark = SparkSession.builder()
      .appName("AverageMarks")
      .master("local[*]")
      .getOrCreate()

    // Sample input data
    val data = Array(
      ("Joe", "Maths", 83),
      ("Joe", "Physics", 74),
      ("Joe", "Chemistry", 91),
      ("Joe", "Biology", 82),
      ("Nik", "Maths", 69),
      ("Nik", "Physics", 62),
      ("Nik", "Chemistry", 97),
      ("Nik", "Biology", 80)
    )

    // Convert data to RDD
    val rdd = spark.sparkContext.parallelize(data)

    // Use combineByKey to calculate sum and count
    val sumCount = rdd.combineByKey(
      (marks: Int) => (marks, 1),                    // createCombiner: (marks)
=> (marks, 1)
      (acc: (Int, Int), marks: Int) => (acc._1 + marks, acc._2 + 1),  //
mergeValue: (acc, marks) => (acc._1 + marks, acc._2 + 1)
      (acc1: (Int, Int), acc2: (Int, Int)) => (acc1._1 + acc2._1, acc1._2 +
acc2._2)  // mergeCombiners: (acc1, acc2) => (acc1._1 + acc2._1,
acc1._2 + acc2._2)
    )

    // Calculate average
    val averageMarks = sumCount.mapValues { case (sum, count) =>
sum.toDouble / count }
```

```
    // Collect the results and print
    val result = averageMarks.collect()
    result.foreach { case (student, average) =>
      println(s"Student: $student, Average Marks: $average")
    }

    spark.stop()
  }
}
```

21. Consider the Employee table with the fields as (EmpID, Dept, EmpDesg). Design the Spark program to partition the table using Dept and construct the hashed partition of 4.

```
import org.apache.spark.sql.SparkSession
import org.apache.spark.HashPartitioner

// Step 1: Define Employee case class
case class Employee(EmpID: Int, Dept: String, EmpDesg: String)

object PartitionExample {
  def main(args: Array[String]): Unit = {
    // Step 2: Create SparkSession
    val spark = SparkSession.builder()
      .appName("PartitionExample")
      .master("local[*]")  // Change to appropriate cluster URL in
production
      .getOrCreate()

    // Step 3: Read data into RDD or DataFrame
    // For demonstration, creating RDD from a local collection
    val data = Seq(
      Employee(1, "IT", "Developer"),
      Employee(2, "HR", "Manager"),
      Employee(3, "IT", "Analyst"),
      Employee(4, "Admin", "Clerk"),
```

```
    Employee(5, "HR", "Assistant"),
    Employee(6, "IT", "Tester"),
    Employee(7, "Admin", "Manager")
   )

   import spark.implicits._

   // Create RDD from the sequence
   val rdd = spark.sparkContext.parallelize(data)

   // Step 4: Hash partition by Dept into 4 partitions
   val partitionedRDD = rdd.map(emp => (emp.Dept,
emp)).partitionBy(new HashPartitioner(4)).map(_._2)

   // Optionally, you can perform further operations on
partitionedRDD
   partitionedRDD.foreach(println)  // For demonstration, printing
each record

   // Step 5: Write partitioned data to a destination
   // Example:
partitionedRDD.toDF().write.mode("overwrite").csv("output_path")

   // Stop SparkSession
   spark.stop()
  }
}
```

22. Consider a collection of 100 items of type integer given in the csv file. Write the Spark code to find the average of these 100 items.

```
  // Step 2: Read text file into an RDD
   val textFilePath = "path/to/your/input.txt"  // Replace with your
actual text file path
   val rdd = spark.sparkContext.textFile(textFilePath)
    .flatMap(line => line.split("\\s+"))  // Split each line by
whitespace to get individual numbers
```

```
  .map(_.toInt)  // Convert each string to integer

// Step 3: Calculate average using Spark aggregation functions
val count = rdd.count()
val sum = rdd.reduce(_ + _)
val average = sum.toDouble / count

// Step 4: Display or save the result
println(s"Average of the items: $average")
```

23. Consider a collection with items as (11,34,45,67,3,4,90).

i. Illustrate how spark context will construct the RDD from the collection, assuming number of partitions to be made is 3.

ii. Using mapPartitionsWithIndex return content of each partition along with partition index and apply a function, that increments the value of each element by 1, and returns an array.

```
import org.apache.spark.{SparkConf, SparkContext}
object RDDFromCollection {
def main(args: Array[String]): Unit = {
// Step 1: Create SparkContext
val conf = new
SparkConf().setAppName("RDDFromCollection").setMaster("local[
*]")
val sc = new SparkContext(conf)


// Step 2: Create the collection
val data = Array(11, 34, 45, 67, 3, 4, 90)


// Step 3: Create RDD from the collection with 3 partitions
val rdd = sc.parallelize(data, 3)


// Print RDD partitions
```

```
println("RDD partitions:")
rdd.glom().collect().foreach(arr => println(arr.mkString(", ")))

// Stop SparkContext
sc.stop()
}
}
```

24. Consider the Item object.
Item = Map{(“Ball”:10), (“Ribbon”:50), (“Box”:20), (“Pen”:5), (“Book”:8), (“Dairy”:4),(“Pin”:20)}
Design the spark program to perform the following
   i. Find how many partitions are created for the collection Item?
   ii. Display the content of the RDD Display the content of each partition separately

25. The table 1b provides the distributed data of the scala object
Item = Map{(“Ball”:10), (“Ribbon”:50), (“Box”:20), (“Pen”:5), (“Book”:8), (“Dairy”:4),(“Pin”:20)}

Partition1
{(“Ball”:10) (“Ribbon”:50) (“Box”:20)
Partition2
(“Pen”:5) (“Book”:8)
Partition3
(“Dairy”:4) (“Pin”:20)

Table 1b: Data distribution
Design the spark program to perform the following
   i. Find how many partitions are created for the collection Item?

ii. Display the content of the RDD

iii. Display the content of each partition separately.

26. Consider the text file words.txt as shown in the figure 1a.
Write the spark code to perform the following.

i) Count the number of occurrences of each word.

ii) Arrange the word count in ascending order based on Key.

iii) Display the words that begin with ‘s’.

She sells sea shells by the seashore

And the shells she sells by the seashore

Are sea shells for sure

Figure 1. a

27. Illustrate the application of combineByKey to combine all the values of the same key in the following collection.
((“coffee”,2),(“cappuccino”,5),(“tea”,3),(“coffee”,10),(“cappuccino”, 15))