**Q 2-1, word count per word/doc combination**

Class MAPPER

Method Map(docset s)

For all docid a, doc d ϵ docset s do

For all term t ϵ doc d do

EMIT((docid a, term t), count 1)

Class REDUCER

Method REDUCE((docid a, term t), counts [c1, c2, …]

Sum <- 0

For all count c ϵ counts [c1, c2, …] do

Sum <- sum + c

EMIT((docid a, term t), sum)

**Q 2-2, total number of words per doc**

Class MAPPER

Method Map((docid a, term t), count c)

EMIT(docid a, count c)

Class REDUCER

Method REDUCE(docid a, counts [c1, c2, …]

Sum <- 0

For all count c ϵ counts [c1, c2, …] do

Sum <- sum + c

EMIT(docid a, sum)

**Q 2-3, total number of docs per word**

Class MAPPER

Method Map((docid a, term t), count c)

EMIT(term t, count 1)

Class REDUCER

Method REDUCE(term t, counts [c1, c2, …]

Sum <- 0

For all count c ϵ counts [c1, c2, …] do

Sum <- sum + c

EMIT(term t, sum)

**Q 2-4, TF-IDF**

Class MAPPER

Method Map((docid a, term t), count c, idf i)

For all (term t2, freq f) ϵ idf I do

If t == t2 then EMIT((docid a, term t), (count c, freq f))

Class REDUCER

Method REDUCE(((docid a, term t), (count c, freq f)), numberOfDocs n)

EMIT((docid a, term t), count c \* log2(numberOfDocs n/freq f))

**Q3**

**package ICE5  
import breeze.numerics.log2  
import org.apache.spark.{SparkConf, SparkContext}  
object ICE5 {  
 def main(args : Array[String]){  
 // administration  
 System.*setProperty*("hadoop.home.dir", "C:\\winutils")  
 val config = new SparkConf()  
 .setAppName("ICE5")  
 .setMaster("local[\*]")  
 val sc = new SparkContext(config)  
  
  
 val d1\_wordCount = sc.textFile("src/main/scala/ICE5/docs/d1.txt").flatMap(line => line.split(" ")).map(word => (*List*("d1",word), 1))  
 .reduceByKey(\_ + \_)  
 val d2\_wordCount = sc.textFile("src/main/scala/ICE5/docs/d2.txt").flatMap(line => line.split(" ")).map(word => (*List*("d2",word), 1))  
 .reduceByKey(\_ + \_)  
 val d3\_wordCount = sc.textFile("src/main/scala/ICE5/docs/d3.txt").flatMap(line => line.split(" ")).map(word => (*List*("d3",word), 1))  
 .reduceByKey(\_ + \_)  
 val wordCounts = d1\_wordCount.union(d2\_wordCount).union(d3\_wordCount)  
  
 val d1\_totalCount = d1\_wordCount.map(x => ("d1", x.\_2)).reduceByKey(\_ + \_)  
 val d2\_totalCount = d2\_wordCount.map(x => ("d2", x.\_2)).reduceByKey(\_ + \_)  
 val d3\_totalCount = d3\_wordCount.map(x => ("d3", x.\_2)).reduceByKey(\_ + \_)  
  
 val docsPerWord = wordCounts.map(x => (x.\_1(1),1)).reduceByKey(\_ + \_)  
 val idf = docsPerWord.map(x => (x.\_1, log2(3.0/x.\_2)))  
  
 val tfidf\_d1 = wordCounts.map(*identity*).join(idf.map(x => (*List*("d1",x.\_1),x.\_2))).map(y => (y.\_1,(y.\_2.\_1 \* y.\_2.\_2)))  
 val tfidf\_d2 = wordCounts.map(*identity*).join(idf.map(x => (*List*("d2",x.\_1),x.\_2))).map(y => (y.\_1,(y.\_2.\_1 \* y.\_2.\_2)))  
 val tfidf\_d3 = wordCounts.map(*identity*).join(idf.map(x => (*List*("d3",x.\_1),x.\_2))).map(y => (y.\_1,(y.\_2.\_1 \* y.\_2.\_2)))  
 val tfidf = tfidf\_d1.union(tfidf\_d2).union(tfidf\_d3)  
  
  
 tfidf.saveAsTextFile("src/main/scala/ICE5/output")  
 }  
}**