Rudolf Chrispens
Marvin Klaus 3486809
Daniela

Problem 1

(a)

$1 - (1 - p)^n$

(b)

$\frac{n}{1-(1-p)^n}$

(c)

If *(b)* is added n times, you get the solution of *(a)*

Problem 2

(a)

join(): Applied on a dataset with tuples, *join* pairs all the same keys with their value. Input should be a dataset with *(K, V)* and *(K, W)* tuples and the output is a dataset with *(K, (V, W))* tuples. Example under *(a)*. *sort()*: I couldn't find a *sort()* Transformation. But I describe *sortByKey()* instead. This Transformation gets an dataset of *(K, V)* tuples and sort them by key. You can sort them descending or ascending. *groupBy()*: I couldn't find the *groupBy()* Transformation and describe the *groupByKey()* Transformation instead. This Transformation gets a dataset of tuples and groups values of the same key together. So the input tuples are *(K, V)* and the output is *(K, Iterable¡V¿)*.

```
// Example data
rdd1 =  sc.parallelize([("foo", 1), ("bar", 2), ("baz", 3)])
rdd2 =  sc.parallelize([("foo", 4), ("bar", 5), ("bar", 6)])

// Example join:
rdd1.join(rdd2)

// Example sortByKey:
rdd1.sortByKey()

// Example groupByKey:
rdd1.groupByKey()
```

(b)

Problem 3

Problem 4

(a)

Broadcast provides a fast way to send data to each desired *node* once. In this context the *node* is one dataset of the data we want to map.
Broadcast is a better solution than to just *join* the data. Once its send to a machine with Broadcast it stays cached on this machine and you can access the Broadcast values on the *nodes*. But be careful, don't modify

Problem 4 [(a)] continued on next page...

the Broadcast data.
A *task* is the function we want to execute in the map step.

(b)

Accumulators are used to count something up. For example (in the video) we need this to count failures in the application.
If we want to build a custom accumulator we have to implement the type of the accumulator. For example a Vector class to build an accumulator of type Vector.
The accumulator can only be used in the Master (Exception on workers), the reduce() function gathers data from all tasks.

(c)

The *join()*, *mapValues()* and *reduceByKey()* all results in partitioned RDD's. With partitioning there is less traffic over networks what makes it much faster.
In the pageRank example the links have a partitioner so that links with the same hash are on the same node. To build a custom one you have to implement a class that extends from Partitioner. The class need the variable *numPartitions* and the functions *getPartition()* and *equals()*.