The goal of the examination is to test your understanding of distributed frameworks and your capacity to manipulate them in an efficient way. This test will not be graded by the syntactic correctness of your code but rather by the demonstration that you can write (efficient) distributed code. For each question, you will be penalized if your answer uses Spark/Hadoop inefficiently (e.g. doing `collect` with too much data, shuffling more data than needed or doing a useless shuffle).

I need to understand what you are writing so please be clear, use the Python language (even for Map Reduce) and comment your code. You can use a mix of Python and Pseudo-code but be very clear on how this translate to Python. Here are several examples of what I expect if it was asked to compute how many times each input items appears (classic word count).

**Pythonic style with RDD**

```python
input RDD is R

R .map(lambda x: (x,1))
   .reduceByKey(lambda a,b: a+b) #don't hesitate to break lines!
```

**Pythonic style with RDD without anonymous functions**

```python
def mapper(x):
    return (x,1)

def reducer(a,b):
    return a+b

R.map(mapper).reduceByKey(reducer)
```

**Pseudo-code + Python style with Spark**

```python
R.map(x → (x,1)).reduceByKey(a,b → a + b)
```

**Dataframe code**

```python
input.groupBy("item").count()
```

**Dataframe (each Dataframe question can be answered with SQL code)** Table input has one column “item”.

```sql
SELECT COUNT(*) FROM input GROUP BY item
```

**Hadoop (provide only the mapper, reducer and eventually the combiner functions)**

```python
def mapper(x): # returns a list
    return [(x,1)]

def reducer(key, vals): # takes a key and a list of values
    return sum(vals) # returns an element or a list
```

Final note: please add the question number before each of your answer (e.g. 1.2 or 3.2).
1 Simple Maxima

In this section, you are asked to compute the maximum integer of a list of integers. The answer can either be a single python value or a collection (RDD/Dataframe/HDFS File) containing a single value.

1.1 RDD

Given an RDD $R$ containing integers, write the code computing the maximum element of $R$.

1.2 Dataframe

Given a Dataframe $R$ containing a column $value$ write the code computing the maximum of $value$ in $R$ (using only the Dataframe API).

1.3 Map Reduce

Write a pair of functions $map$ and $reduce$ computing the maximum in the style of Hadoop MapReduce. The input of the map will be a line of text containing a single integer and the output of all the reduce calls should be a single integer. The type of $map$ should be $\text{int} \rightarrow (\alpha \times \beta) \text{list}$ and the type of $reduce$ should be $\alpha \times (\beta\text{list}) \rightarrow \text{int list}$ (for some type $\alpha$ and $\beta$).

2 Key Maxima

In this section, you are given a list of pairs $(k,v)$ you are asked to compute for each $k$ the maximum $v$ associated with $k$. The answer should be a distributed collection (RDD/Dataframe/HDFS file) and not a python value (don’t collect data on the driver!).

2.1 RDD

Given an RDD $R$ containing pairs (string, integers), write the code computing the maximum integer for each string (using only the RDD API).

2.2 Dataframe

Given a Dataframe $R$ containing a column $value$ and a column $key$ write the code computing the maximum $value$ associated with each $key$ (using only the Dataframe API / SQL code).

2.3 Map Reduce

Write a pair of functions $map$ and $reduce$ computing the maximum for each key in the style of Hadoop MapReduce. The input of the map will be a line of text containing integer and then a string. You can the following function cut(l) to cut the string $l$ into a pair $(i,s)$ where $i$ is the integer and $l$ the key:

```python
def cut(l):
    # returns a pair
    return l.split(" ",1)
```
3 Recode basic RDDs functions

3.1 Implement filter with flatMap

Imagining that the filter operation was not defined in the Spark RDD API, how could perform myRDD.filter(f) using only the flatMap operation? Here myRDD is an RDD and f is function returning a boolean.

3.2 Implement GroupByKey

Imagining that the groupByKey operation was not defined in the Spark RDD API, how could perform myRDD.groupByKey()? Note, in this question you are not limited to flatMap and you can use a combination of several operators.

4 Optimize a Spark program

Let $A$ be an RDD containing triples $(d, c, n)$ where $d$ is an integer corresponding to a french department number, $c$ is a city in this department and $n$ is the number of people living in the city $c$. Consider the following program:

```python
def bestInDpt(cities):
    bestNb = -1
    bestCity = ""
    for (nb,city) in cities:
        if nb > bestNb:
            bestNb = nb
            bestCity = city
    return bestCity

A.map(lambda x: x[0],(x[1],x[2])).groupByKey().mapValues(bestInDpt)
```

4.1 Explain what the program computes

4.2 Optimize the program (using only the Spark RDD API)

If needed, you can suppose that there is a moderate number of department (e.g. $10^4$) but you cannot suppose the number of cities is moderate (e.g. $10^{10}$). If your program require such supposition, please state so and explain why.

4.3 Write an equivalent program in the dataframe API

Suppose that $A$ is now a Dataframe, how to compute the same answers?

5 Comparing programs

5.1 Comparing Dataframe programs

Let $D_1$ be a Dataframe containing three columns: rating, movieId, userId and $D_2$ be a Dataframe containing two columns: movieId and title. Which of the two following programs is the most efficient? If your answer depends on constraints over the content of $D_1$ or $D_2$ (such as the number of ratings per movies or that a movieId corresponds to a unique title) please state so and explain why.
Program 1

```java
D1.join(D2,"movieId")
   .filter("title='Toy Story 1'")
   .groupBy("movideId")
   .avg("rating")
   .filter("avg>3.5")
```

Program 2

```java
D1.groupBy("movideId")
   .avg("rating")
   .filter("avg>3.5")
   .join(D2,"movieId")
   .filter("title='Toy Story 1'")
```

5.2 Comparing RDD programs

Let carSold be an RDD containing triplets (price, model of car, color) describing cars sold in the last year. The first column contains the price, an integer, the second column the model of the car, a string, and the third column contains an integer ranging from 0 to $NB\_COLORS - 1$. Which of the two following programs is the most efficient? If your answer is conditional on some constraints please state so and explain why.

Program 3

```python
colorPrice = carSold.map(lambda x: (x[2],x[0]))
colorTotalPrice = colorPrice.reduceByKey(lambda x,y: x+y)  # sum the price for each color
cTP = colorTotalPrice.collect()  # collect on the driver
cTP.sort(key=lambda x:x[1])  #sort by total price
cTP[-1]  # returns color with best total price
```

Program 4

```python
def combine(a,b):
    res = []
    for color in range(NB\_COLORS):
        res.append(a[color],b[color])
    return res

def ofColor(color,price):
    res = [0]*NB\_COLORS
    res[color]+=price
    return res

TP = carSold.map(lambda x: ofColor(x[2],x[0])).reduce(combine)
best = 0
for i in range(len(TP)):
    if TP[best]<TP[i]:
        best = i
```
6 Erdős number

The Erdős number of a person is its collaborative distance to Erdős:

- Erdős has an Erdős number of 0,
- a person who has collaborated with Erdős has an Erdős number of 1,
- generally, a person has an Erdős number of \( i + 1 \) when she has collaborated with a person with an Erdős number of \( i \) but has not collaborated with a person with an Erdős number \( j < i \).

In this section we suppose that Collaboration is an RDD containing pairs \((a, b)\) meaning that \(a\) collaborated with \(b\). The RDD is symmetric: if \((a, b)\) belongs to the RDD then \((b, a)\) also belongs to the RDD.

We define \(erdos(n)\) as the RDD containing all the pairs \((p, k)\) such that \(p\) is a string representing a person and \(k \leq n\) is its Erdős number. \(erdos(n)\) contains exactly one entry for each person that has an Erdős number less than \(n\).

For instance \(erdos(0)\) can be defined the following way:

\[
erdos0 = \text{sparkContext.parallelize}(["Erdős", 0])
\]

6.1 Collaborate With

Complete the following function so that it returns an RDD will all the people collaborating with at least one person in the RDD \(R\):

```python
def collaborateWith(R):
    ...
    return resRDD
```

6.2 \(erdos(n)\)

Complete the function below. The function takes one argument \(n\) and should return an RDD containing pair \((p, k)\), \(k\) is an integer less than \(n\), \(p\) is a string and the pair describes that the person \(p\) has Erdős \(k\).

```python
def erdos(n):
    if n == 0:
        return erdos0
    else:
        erdosN = ...
        return erdosN
```
7 Computing the $k$-th element

The goal of this section is to create an algorithm using the RDD API to compute the $k$-th element of an RDD containing integers. The idea of the algorithm is now described.

We will compute recursively $KElement(R, N, K)$ where $R$ is an RDD of integers, $N$ is the size of $R$ and $K < N$ is number of the element we are interested:

- When $N < 10^4$, we collect everything, sort the resulting array and return the $K$-th element of this array.
- Otherwise we sample $R$ without replacement and with probability $\frac{10^4}{N}$. We then obtain an RDD with roughly $10^4$ elements that we collect on the driver. We sort the resulting list on the driver and we obtain a list $v_0 \ldots v_k$ with $v_i \leq v_{i+1}$.
- The values $v_0 \ldots v_k$ define $k + 2$ intervals: $] - \infty; v_0]$, $[v_0; v_1]$, $\ldots$, $[v_k; + \infty]$. We count how many items of $R$ fall into each of these intervals.
- We collect the number of items in each interval on the driver and we use this information to determine into which interval the $k$-th element of $R$ belong.
- Finally we recurse with $KElement(I, S, K - B)$ where $I$ is the RDD corresponding to items in the interval into which $k$ belongs, $S$ it the size of $I$ and $B$ is the number of elements in intervals before $I$.

7.1 Translate this algorithm into Spark RDD code

7.2 Estimate the efficiency of this code in terms of total operations performed and total data transfer.