CPSC 5330 – Big Data Analytics – Spring 2021
Midterm Exam

Nine questions – 10 points per question for the first 8, 20 points for #9
One sheet of notes
90 minutes. Exam starts at 6:00, ends at 7:30. There will be a 10-minute "late" period between 7:30 and 7:40, where submissions will be penalized 20%. No work will be accepted after 7:40.

Name: _____

Question	Score
1	
2	
3	
4	
5	
6	
7	
8	
9	

1. One of the most fundamental design principles in Hadoop is "bring the computation to the data." Explain what this means and how Hadoop implements this principle.

2. Cassandra implements a policy of "eventual consistency" whereas traditional DBMS systems guarantee strict consistency. Define the difference between eventual and strict consistency. What is a system gaining by relaxing strict consistency?

3. Explain the function of the NameNode in HDFS. What functions does it perform? Sketch the flow of a request from a client to read a record from a file stored in HDFS.

4. One of Sqoop's jobs is to move data from a relational database table or tables into HDFS. Suppose T is a table to be moved. What does Sqoop do to try to parallelize the data movement as much as possible?

5. Avro and schema evolution

Avro has the concept of "forward compatible" schema changes: If I have a v1 schema and I make certain changes to the schema to bring it to v2, an application written to process v1 data can still read data that is produced by the v2 schema without breaking or losing information, provided the changes are all forward compatible.

For the following changes indicate whether or not the change is forward compatible (YES) or is not forward compatible (NO) or can be forward compatible with additional restrictions on the v2 schema (SOMETIMES). If your answer is SOMETIMES, indicate restrictions on the change.

Change	**Fwd Compatible?**	**Restrictions**
Renaming a field		
Changing a field's default value		
Changing a field's type		
Adding an alias to a field name		
Adding a new field		
Deleting a field		
Adding a value to an enum		
Removing a value from an enum		

6. Explain how Avro represents the concept of an optional field. That is, I have a data table that has an integer field "age" that is optional. How is that represented in the Avro schema?

7. What does "speculative execution" mean in MapReduce, and why is it a good idea?

8. In MapReduce, between Map and Reduce, is the "shuffle and sort" step. What does the shuffle and sort step do?

9. Streaming MapReduce

You are processing city-to-city distance information from various data providers. Here is a sample input file showing the record format:

```
Seattle,Portland,188
Portland,Seattle,215
Tacoma,Seattle,35
Seattle,Portland,192
```

Each line has a city name, a comma character, a second city name, a comma character, then a distance (in miles).

You can assume that each line is properly formatted, the third argument is a positive integer, no record has the same city name twice, city names do not contain commas, and that every provider supplies city names the same way. That is, city names will not be misspelled and will be capitalized the same way. So every provider refers to Seattle as 'Seattle' for example. Records in the file are in no particular order.

Providers differ in that (a) different providers can report different distances for the same pair of cities, and (b) the cities in a pair may be provided in either order. Notice the first two lines, which might be from two different providers, which reverse the city order and quote different distances.

You may assume that the actual distance between cities is the same in either direction: that the distance from Seattle to Portland is the same as the distance from Portland to Seattle.

You will write a Map Reduce procedure that will read these records and produce average distance for each city pair, and for each pair of cities, produce *two* records with the same cities in both orders. For example the output file for the input file above would be:

```
Portland,Seattle,198
Seattle,Portland,198
Seattle,Tacoma,35
Tacoma,Seattle,35
```

The order of records in the output file is unimportant.

Write the Mapper and Reducer code that solves this problem, using Python and the Hadoop Streaming framework.

10a. Mapper Code Here

10b. Reducer Code Here

Map Reduce Code for Word Count

```python
#!/usr/bin/env python
"""mapper.py"""
import sys
# input comes from STDIN (standard input)
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # split the line into words
    words = line.split()
    # increase counters
    for word in words:
        # write the results to STDOUT (standard output);
        # what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py
        #
        # tab-delimited; the trivial word count is 1
        print '%s\t%s' % (word, 1)
```

```python
#!/usr/bin/env python
"""reducer.py"""

from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None

# input comes from STDIN
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()

    # parse the input we got from mapper.py
    word, count = line.split('\t', 1)

    # convert count (currently a string) to int
    try:
        count = int(count)
    except ValueError:
        # count was not a number, so silently
        # ignore/discard this line
        continue

    # this IF-switch only works because Hadoop sorts map output
    # by key (here: word) before it is passed to the reducer
    if current_word == word:
        current_count += count
    else:
        if current_word:
            # write result to STDOUT
            print '%s\t%s' % (current_word, current_count)
        current_count = count
        current_word = word
```

Average Word Length by First Letter Category

```
#!/usr/bin/env python
"""mapper.py"""

import sys
import string


for line in sys.stdin:
    for word in line.strip().split():
        lowered = word.lower()
        filtered = filter(lambda c: 97 <= ord(c) <= 122, lowered)
        if len(filtered) > 0:
            o = ord(filtered[0])
            if o <= 105:
                k = 'early'
            elif o <= 114:
                k = 'middle'
            elif o <= 122:
                k = 'late'
            else:
                raise "Bad ordinal!"
            print '%s\t%s' % (k, len(filtered))
```

```
#!/usr/bin/env python
import sys

current_label = None
current_sum = 0
current_count = 0
word = None

for line in sys.stdin:
    line = line.strip()
    label, fcount = line.split('\t', 1)
    try:
        fcount = int(fcount)
    except ValueError:
        continue

    if current_label == label:
        current_sum += fcount
        current_count += 1
    else:
        if current_label:
            # Multipy by 1.0 needed to force real-valued division
            print '%s\t%f' % (current_label, (1.0 * current_sum)/current_count)
        current_label = label
        current_sum = fcount
        current_count = 1

if current_label == label:
    print '%s\t%f' % (current_label, (current_sum * 1.0)/ current_count)
```