CSC 555: Mining Big Data

Project, Phase 2 (due Sunday June 16th)

Lavinia Wang 1473704

In this part of the project, you will execute queries using Hive, Pig and Hadoop streaming and develop a custom version of KMeans clustering. The schema is available below, but don't forget to change to the correct delimiter:
http://rasinsrv07.cstcis.cti.depaul.edu/CSC555/SSBM1/SSBM_schema_hive.sql

The data is available at (this is Scale1, the smallest denomination of this benchmark)
http://rasinsrv07.cstcis.cti.depaul.edu/CSC555/SSBM1/

In your submission, please note what cluster you are using. Please be sure to submit all code (pig, python and Hive). You should also submit the command lines you use and a screenshot of a completed run (just the last page, do not worry about capturing the whole output). An answer without corresponding code will not be counted.

I highly recommend creating a small sample input (e.g., by running head lineorder.tbl > lineorder.tbl.sample, you can create a small version of lineorder with a few lines) and testing your code with it. You can run **head -n 500 lineorder.tbl** to get a specific number of lines.

NOTE: the total number of points adds up to 70.

Part 1: Data Transformation (15 pts)

Transform part.tbl table into a ~-separated ('~') file: Use Hive, MapReduce with HadoopStreaming and Pig (i.e. 3 different solutions).

In all solutions you must switch the first two columns (i.e., switch the positions of columns 1 and 2). You do not need to transform the columns in any way, just switch them around. Note that this means you do not have to use SELECT TRANSFORM or python in your Hive solution.

Using my multi-node cluster

Hive

```
CREATE TABLE part (
  p_partkey     INT,
  p_name        VARCHAR(22),
  p_mfgr        VARCHAR(6),
  p_category    VARCHAR(7),
  p_brand1      VARCHAR(9),
  p_color       VARCHAR(11),
  p_type        VARCHAR(25),
  p_size        INT,
  p_container   VARCHAR(10)
)ROW FORMAT DELIMITED FIELDS
TERMINATED BY '|' STORED AS TEXTFILE;
```

LOAD DATA LOCAL INPATH '/home/ec2-user/part.tbl' OVERWRITE INTO TABLE part;

Python code (colSwitcher.py):

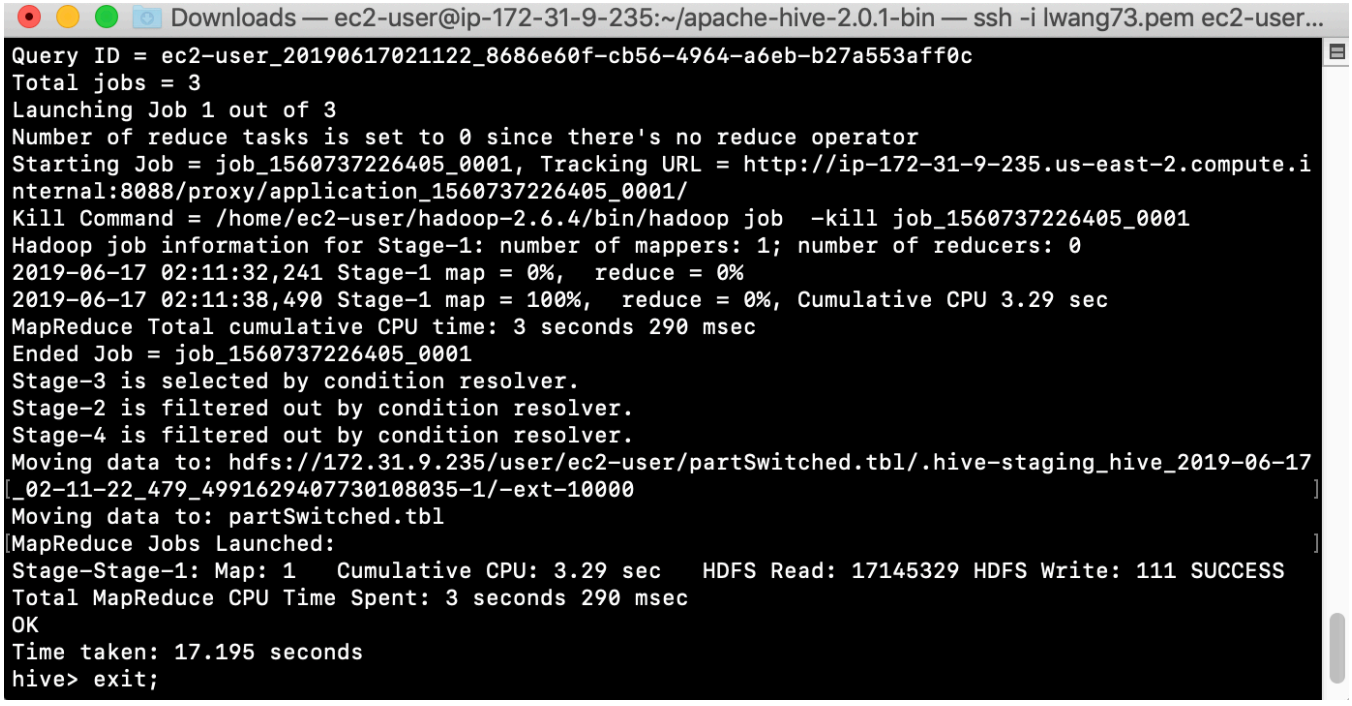
```
#!/usr/bin/python
import sys

for line in sys.stdin:
  line = line.strip().split('\t')
  print '~'.join([line[1], line[0], line[2], line[3], line[4], line[5], line[6], line[7], line[8]])
```

Commands:

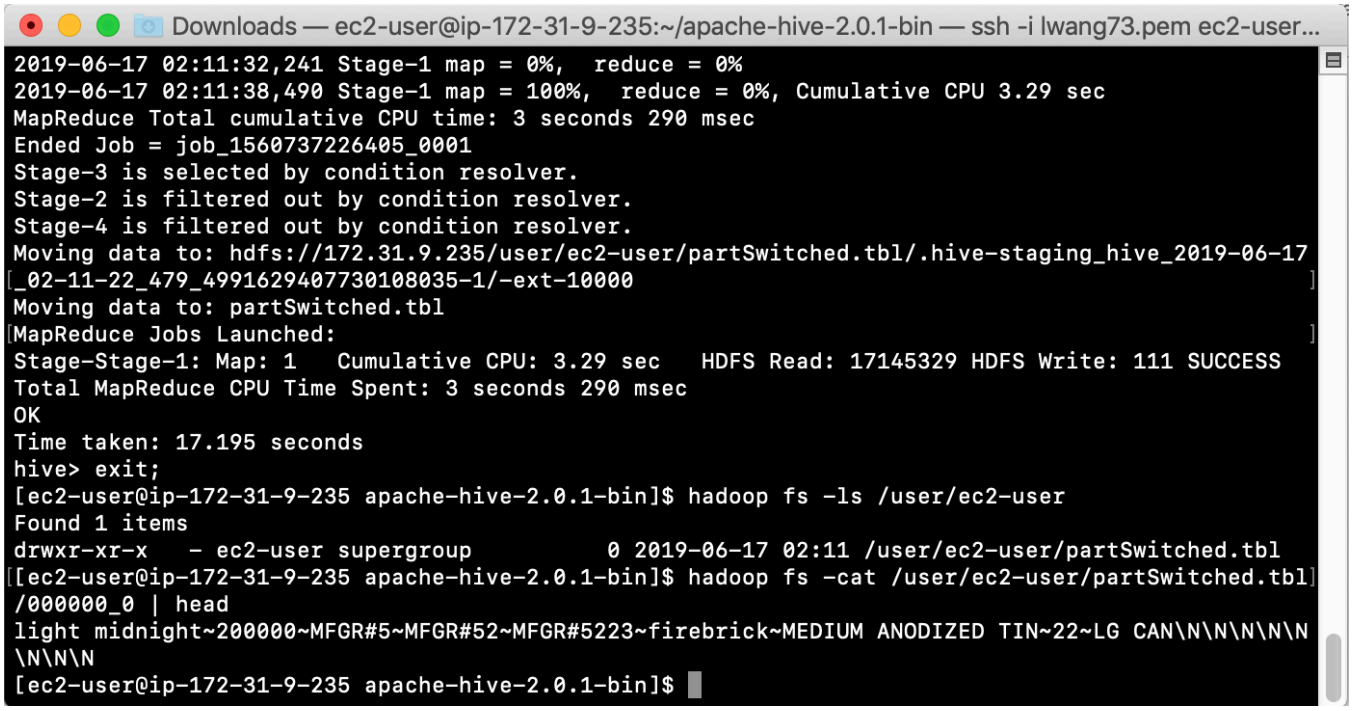
ADD FILE /home/ec2-user/colSwitcher.py;

INSERT OVERWRITE DIRECTORY 'partSwitched.tbl' SELECT TRANSFORM (p_partkey, p_name, p_mfgr, p_category, p_brand1, p_color, p_type, p_size, p_container) USING 'colSwitcher.py' AS (p_name, p_partkey, p_mfgr, p_category, p_brand1, p_color, p_type, p_size, p_container) FROM part;

Completed Run:



Output:



Hadoop Streaming

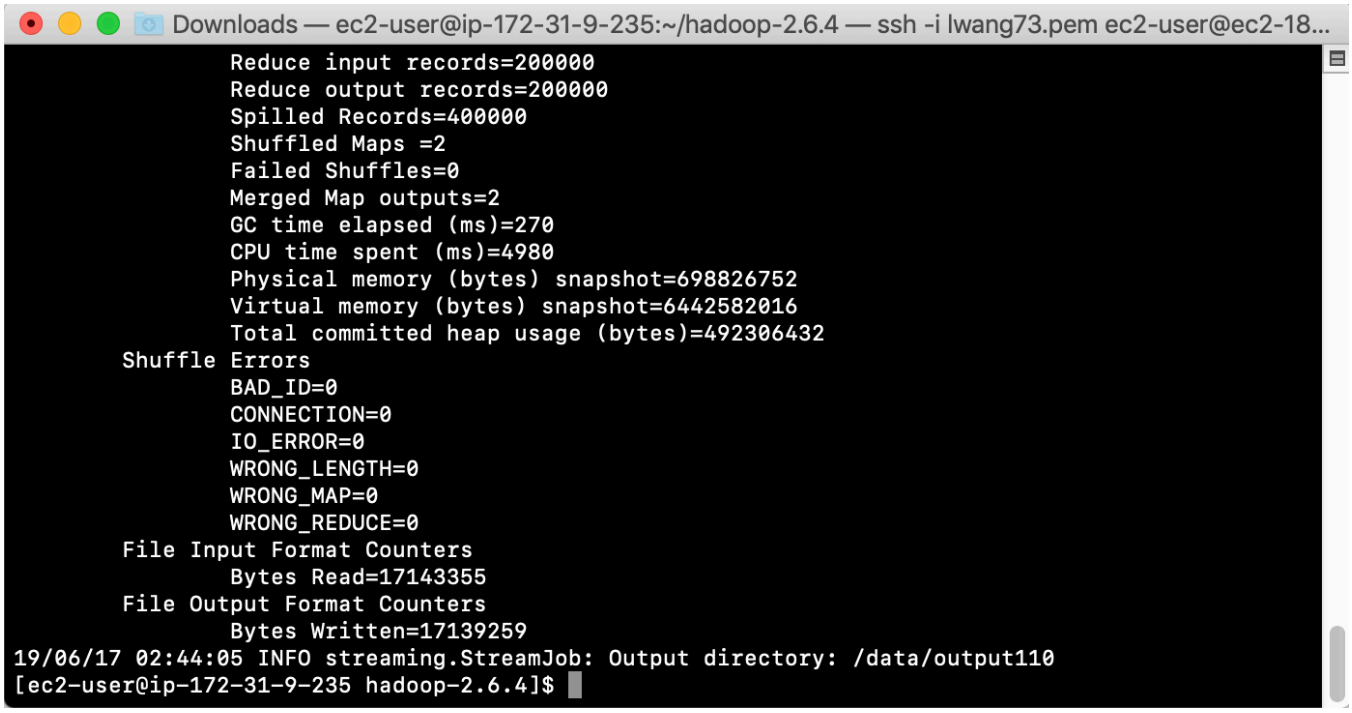
There is no need for a custom mapper for this exercise, so I used the linux cat function as the mapper. The reducer code is:

colSwitcherReducer.py

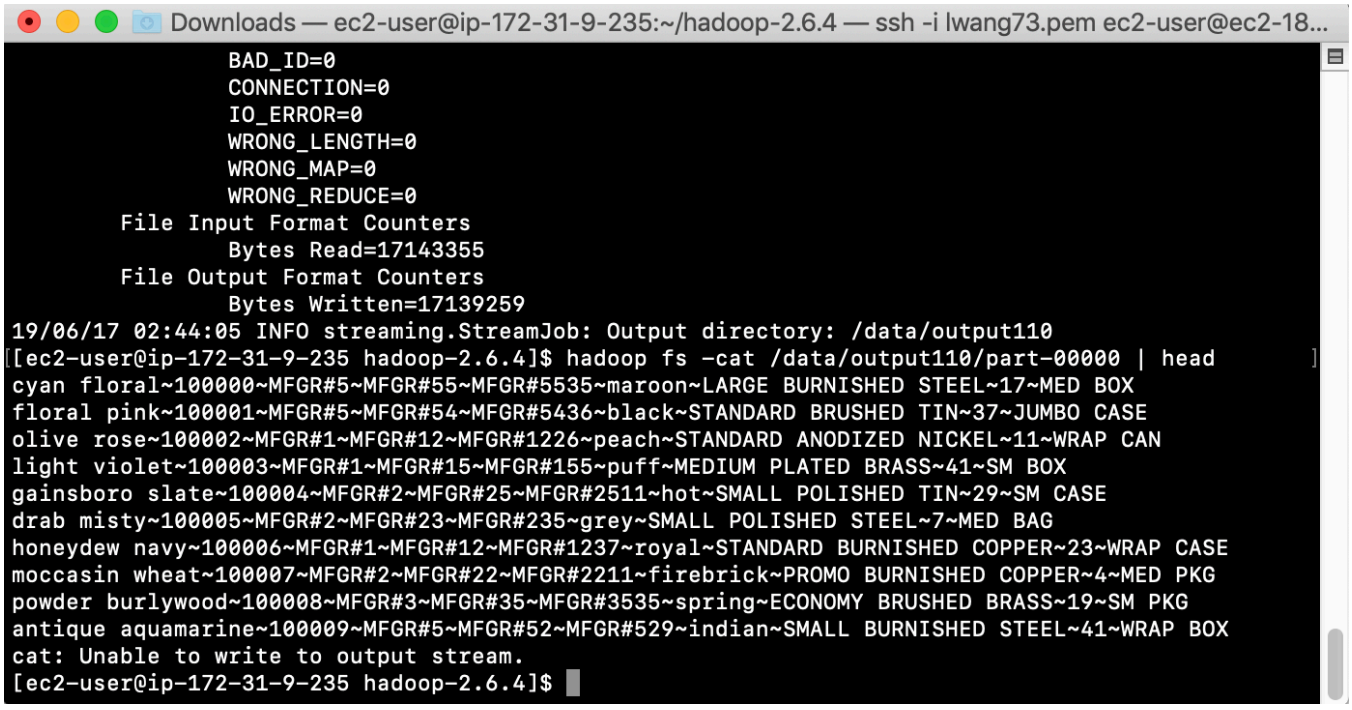
```
#!/usr/bin/python
import sys

for line in sys.stdin:
  line = line.strip().split('|')
  print "%s~%s~%s~%s~%s~%s~%s~%s~%s" %
(line[1],line[0],line[2],line[3],line[4],line[5],line[6],line[7],line[8])
```

Command:

hadoop jar hadoop-streaming-2.6.4.jar -input /user/ec2-user/ssbm/part.tbl -output /data/output110 -mapper /bin/cat -reducer colSwitcherReducer.py -file colSwitcherReducer.py



Output:



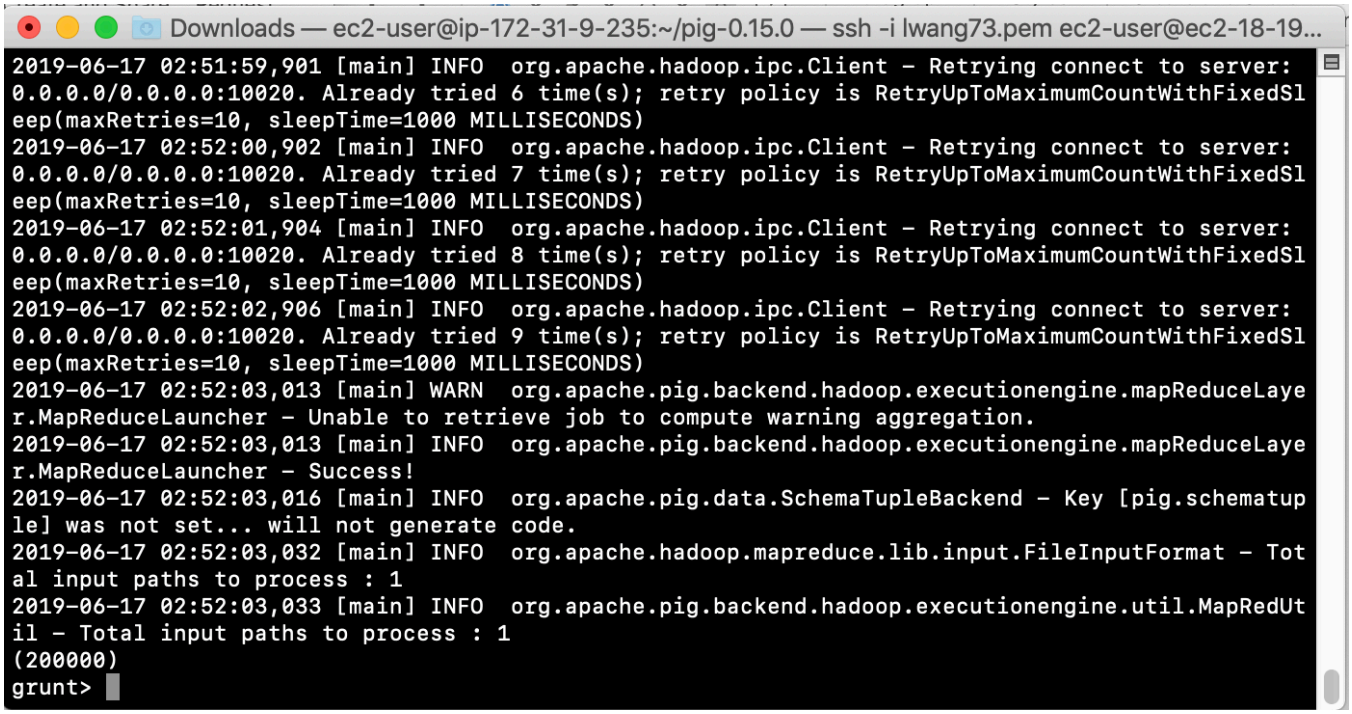
Pig

Load the Data:

PartData = LOAD '/user/ec2-user/ssbm/part.tbl' USING PigStorage('|') AS (p_partkey:int, p_name:chararray, p_mfgr:chararray, p_category:chararray, p_brand1:chararray, p_color:chararray, p_type:chararray, p_size:int, p_container:chararray);

Verify Data Loaded:

PartG = GROUP PartData ALL;
Count = FOREACH PartG GENERATE COUNT(PartData);
DUMP Count;

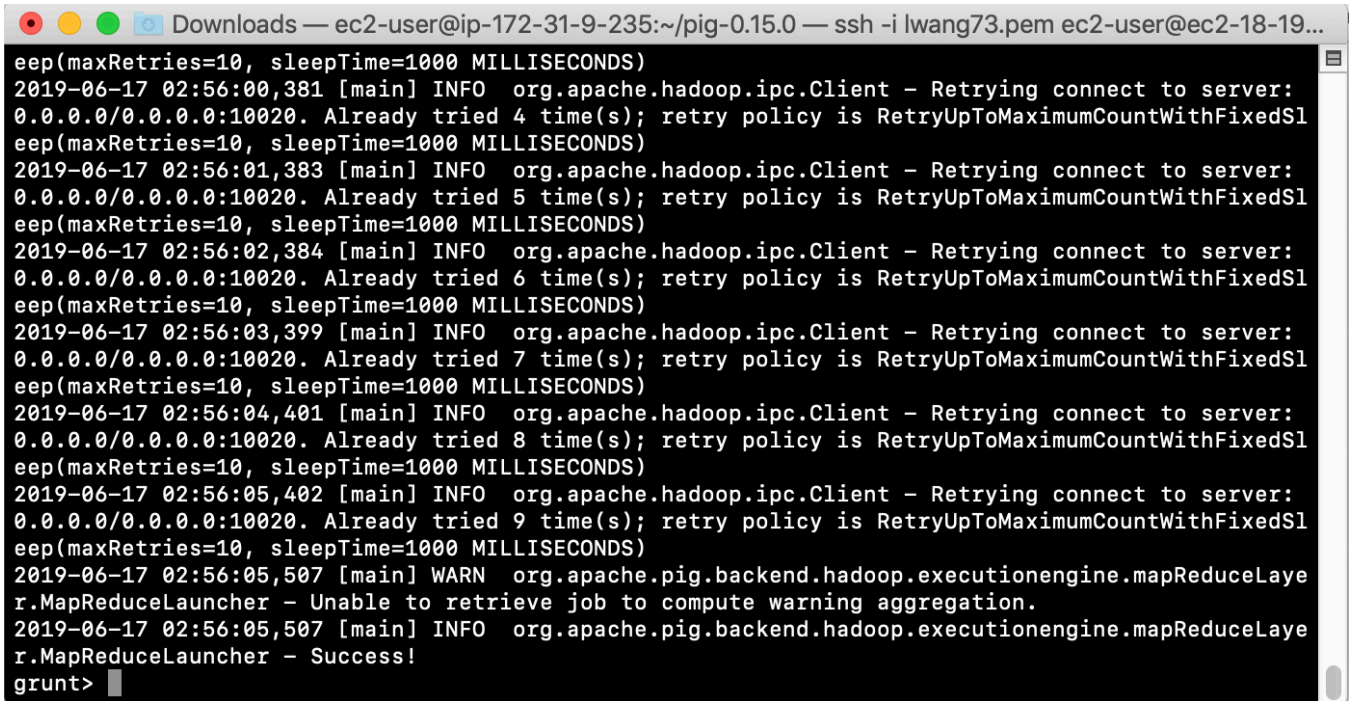


Switch Columns:

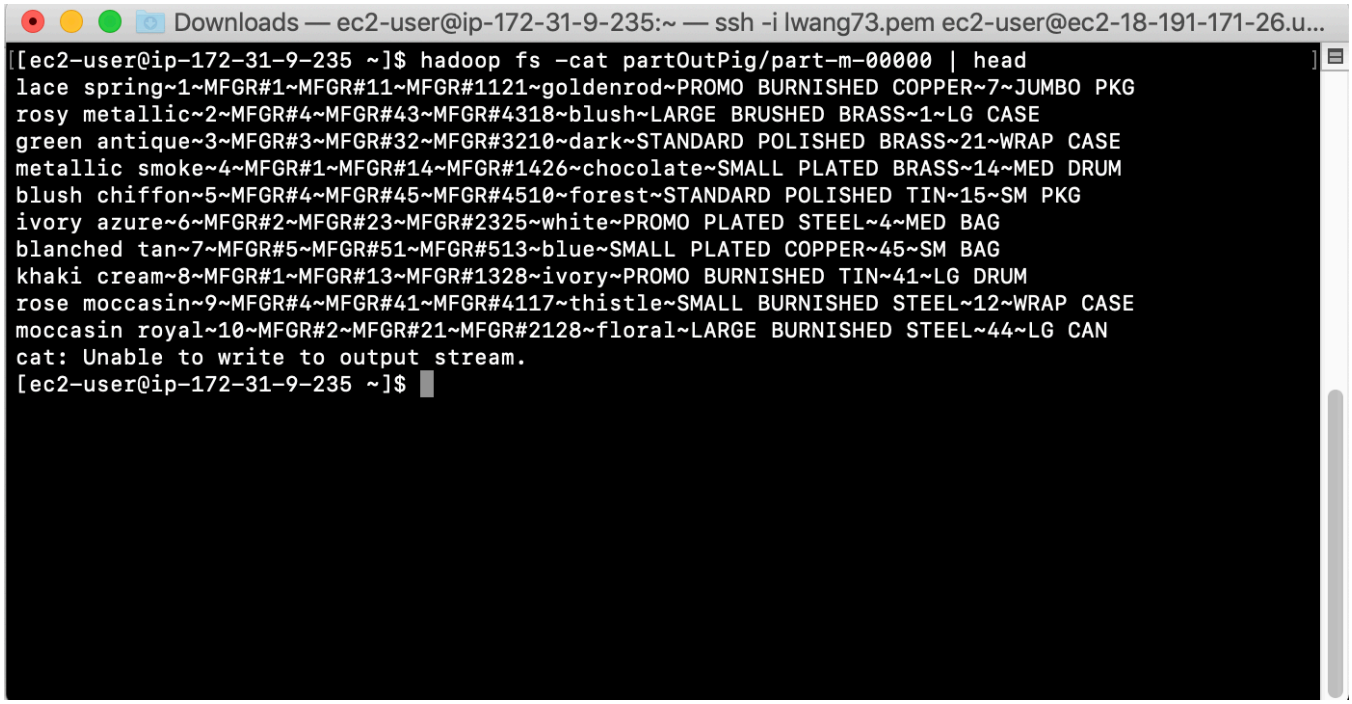
PartSwitchedPig = FOREACH PartData GENERATE p_name, p_partkey, p_mfgr, p_category, p_brand1, p_color, p_type, p_size, p_container;

Write to file:

STORE PartSwitchedPig INTO 'partOutPig' USING PigStorage('~');



Output:



Part 2: Querying (25 pts)

Implement the following query:

```
select c_nation, sum(lo_revenue)
from customer, lineorder
where lo_custkey = c_custkey
  and c_region = 'AMERICA'
  and lo_discount BETWEEN 4 and 6
group by c_nation;
```

using Hive, MapReduce with HadoopStreaming and Pig (i.e. 3 different solutions). I Hive, this merely requires pasting the query into the Hive prompt and timing it. In Hadoop streaming, this will require a total of 2 passes (one for join and another one for GROUP BY).

Hive:

Create and load tables:

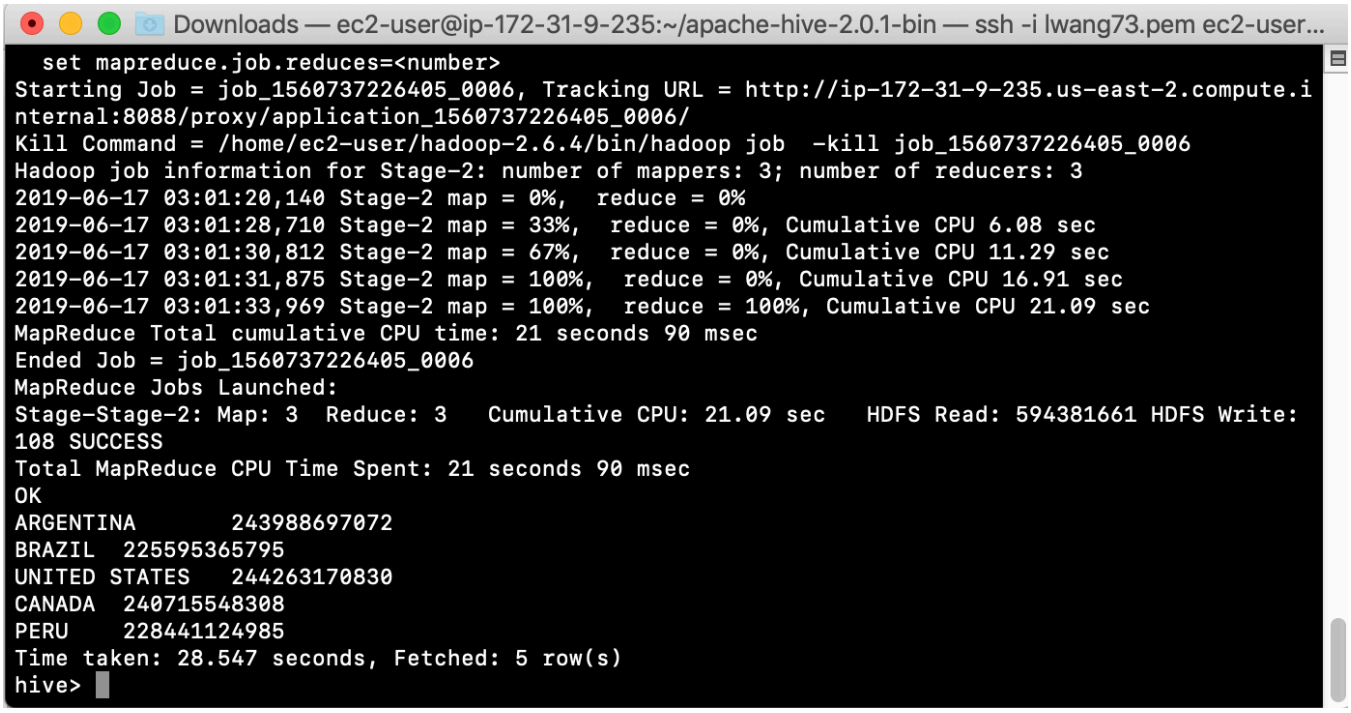
```
CREATE TABLE lineorder (
  lo_orderkey         INT,
  lo_linenumber       INT,
  lo_custkey          INT,
  lo_partkey          INT,
  lo_suppkey          INT,
  lo_orderdate        INT,
  lo_orderpriority    VARCHAR(15),
  lo_shippriority     VARCHAR(1),
  lo_quantity         INT,
  lo_extendedprice    INT,
  lo_ordertotalprice  INT,
  lo_discount         INT,
  lo_revenue          INT,
  lo_supplycost       INT,
  lo_tax              INT,
  lo_commitdate       INT,
  lo_shipmode         VARCHAR(10)
)
ROW FORMAT DELIMITED FIELDS
TERMINATED BY '|' STORED AS TEXTFILE;

LOAD DATA LOCAL INPATH '/home/ec2-user/lineorder.tbl' OVERWRITE INTO TABLE lineorder;
```

```
CREATE TABLE customer (
 c_custkey    INT,
 c_name       VARCHAR(25),
 c_address    VARCHAR (25),
 c_city       VARCHAR (10),
 c_nation     VARCHAR (15),
 c_region     VARCHAR (12),
 c_phone      VARCHAR (15),
 c_mktsegment  VARCHAR (10)
)
ROW FORMAT DELIMITED FIELDS
TERMINATED BY '|' STORED AS TEXTFILE;

LOAD DATA LOCAL INPATH '/home/ec2-user/customer.tbl' OVERWRITE INTO TABLE customer;
```

Execute sql statement:

```
select c_nation, sum(lo_revenue)
from customer, lineorder
where lo_custkey = c_custkey
 and c_region = 'AMERICA'
 and lo_discount BETWEEN 4 and 6
group by c_nation;
```

End of output with time taken:



Hadoop Streaming:

Join

lineCustMapJoin.py

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

# input comes from STDIN (standard input)
for line in sys.stdin:
   line = line.strip().split('|')
   if line[1].startswith('Customer#'):
      if line[5] == 'AMERICA': # Return on matching records
         print line[0], '\t', line[4], '\t', 'customer'
   # lineorder
   else:
      if 4 <= int(line[11]) <= 6: # Return on matching records
         print line[2], '\t', line[12], '\t', 'lineorder'
```

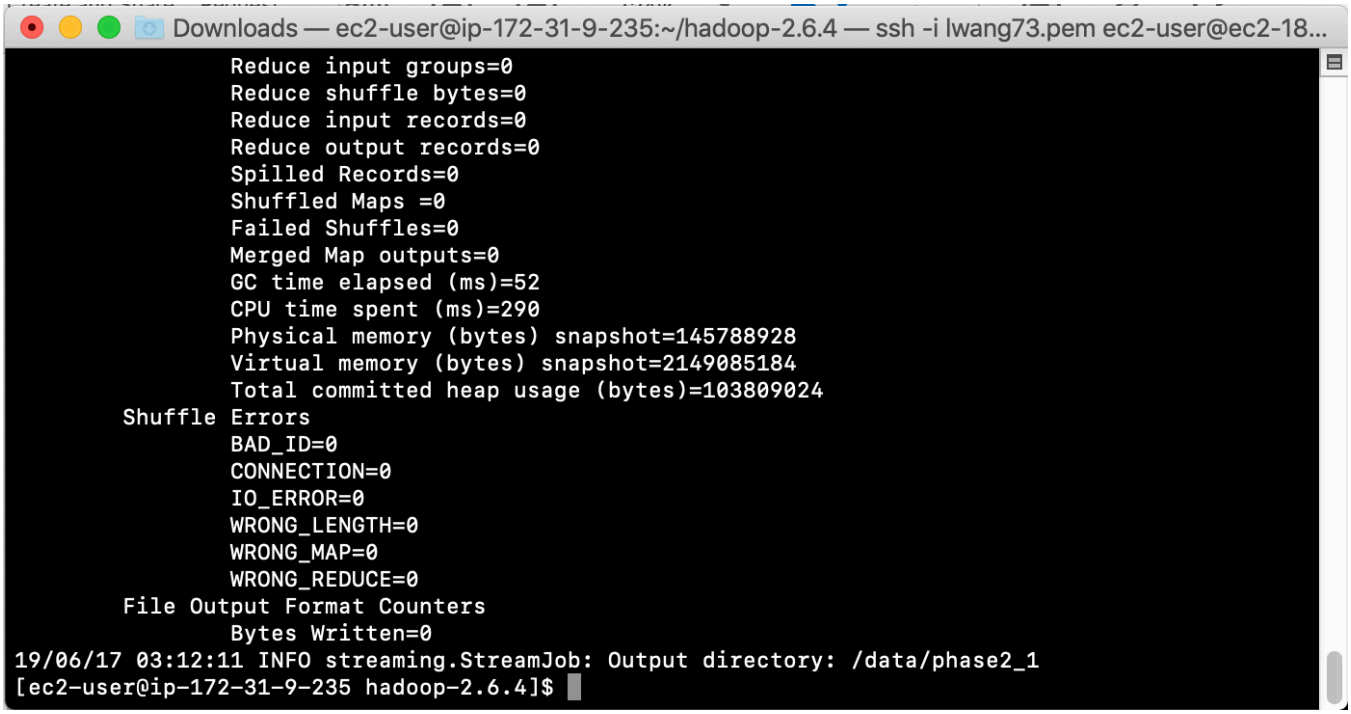
lineCustReduceJoin.py

```
#!/usr/bin/python

import sys

currentKey = None
revenue = []
nation = ''

# input comes from STDIN
for line in sys.stdin:

    split = line.strip().split('\t')
    key = split[0] # key is customer id
    value = '\t'.join(split[1:])

    if currentKey == key: # Same key
        if value.endswith('lineorder'):
            revenue.extend([split[1]])
        if value.endswith('customer'):
            nation = split[1]
    else:
        # Do not print anything until all records
        # for a key have been seen, this is signaled
        # by currentKey != key
        # Check for values and then iterate results
```

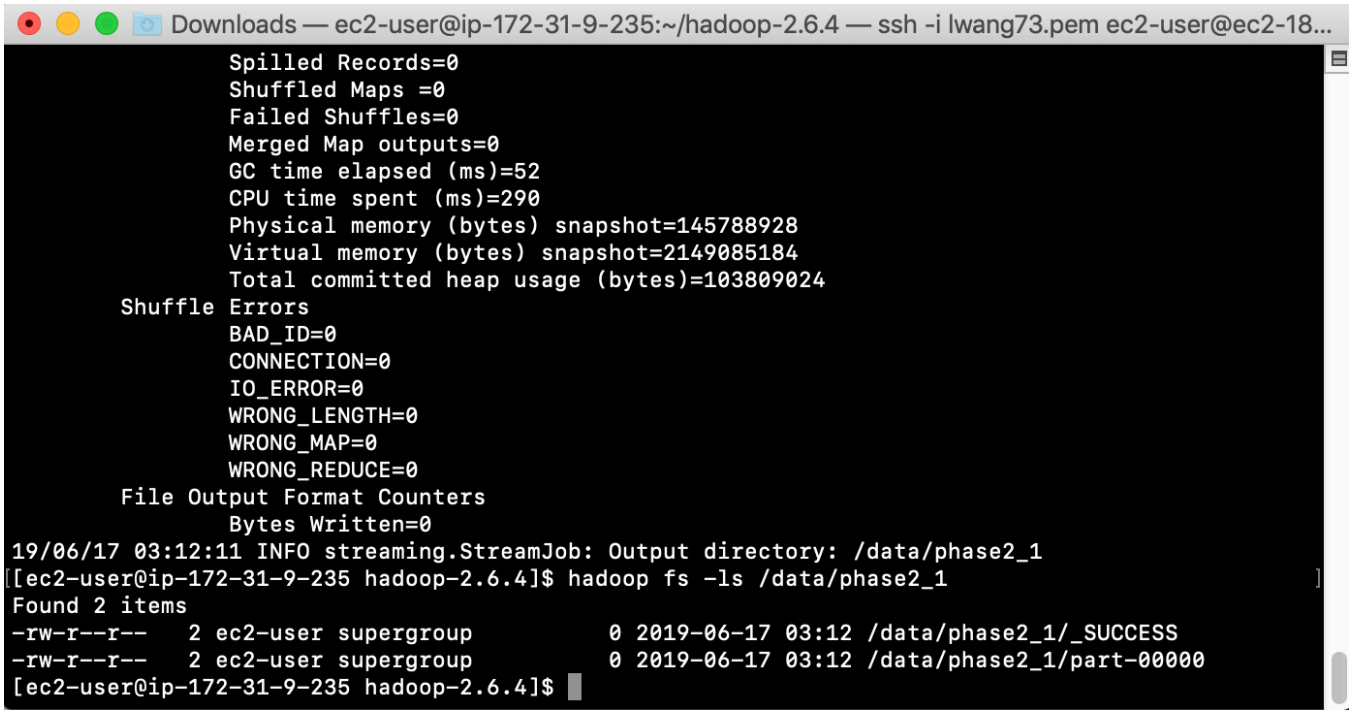
```
                lenNation = len(nation)
                if (lenNation > 0):
                    i = 0
                    while i < lenNation:
                        print nation, '\t', revenue[i]
                        i += 1

                # reset values
                revenue = []
                nation = ''

                if value.endswith('lineorder'):
                    revenue.extend([split[1]])
                if value.endswith('customer'):
                    nation = split[1]

            # set the current key at the end of each iteration
            currentKey = key
```

Commands:

hadoop jar hadoop-streaming-2.6.4.jar -input /user/ec2-user/phase2 -output /data/phase2_1 -mapper lineCustMapJoin.py -reducer lineCustReduceJoin.py -file lineCustMapJoin.py -file lineCustReduceJoin.py



hadoop fs -ls /data/phase2_1



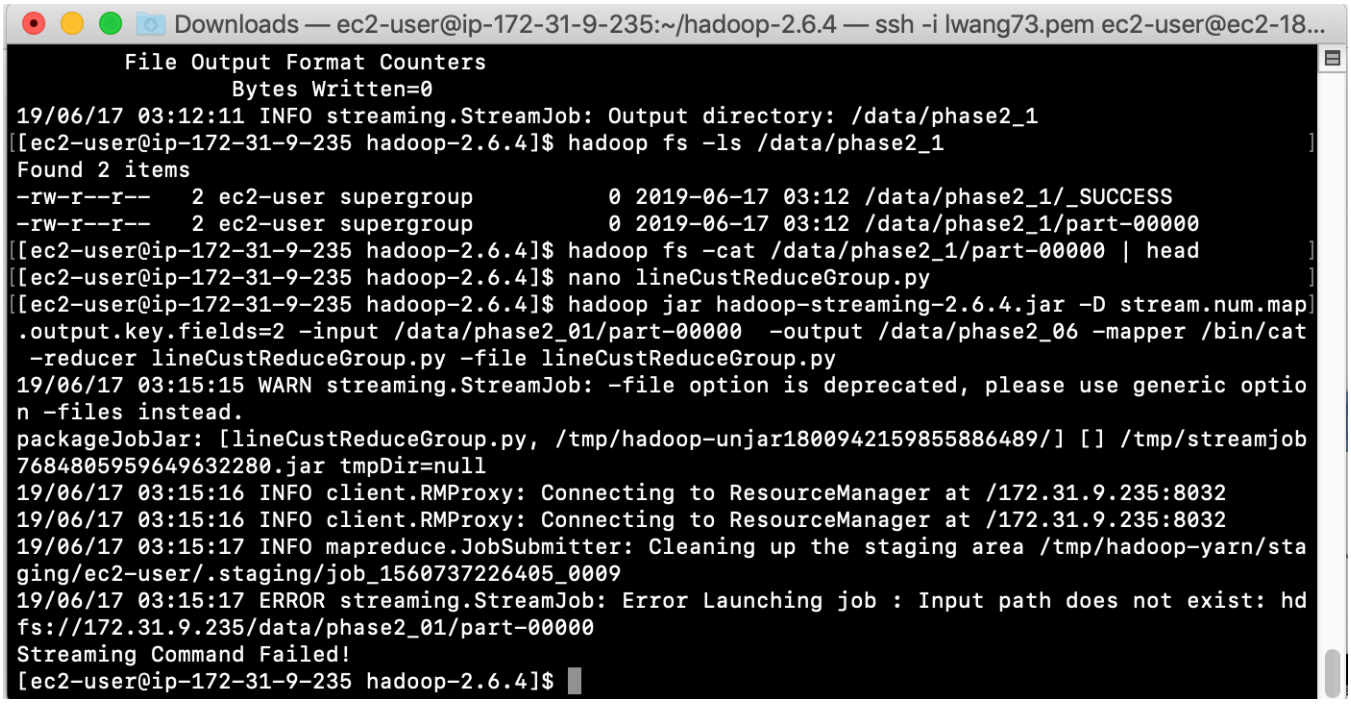
Group:

lineCustReduceGroup.py

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

curr_id = None
curr_tot = 0
id = None

# The input comes from standard input (line by line)
for line in sys.stdin:
    # parse the line and split it by '\t'
    line = line.strip().split('\t')

    # grab the key
    # values include some whitespace, removing here
    id = line[0].strip() + '\t' + line[1].strip()

    # grab the value (int)
    val = int(line[2])

    if curr_id == id:
        curr_tot += val

    else:
```

```
            if curr_id: # output the sum, single key completed
                print '%s\t%d' % (curr_id, curr_tot)
            curr_tot = val

        # set curr_id to id at end of each iteration
        curr_id = id

    # output the last key
    if curr_id == id:
        print '%s\t%d' % (curr_id, curr_tot)
```

hadoop jar hadoop-streaming-2.6.4.jar -D stream.num.map.output.key.fields=2 -input /data/phase2_01/part-00000 -output /data/phase2_06 -mapper /bin/cat -reducer lineCustReduceGroup.py -file lineCustReduceGroup.py



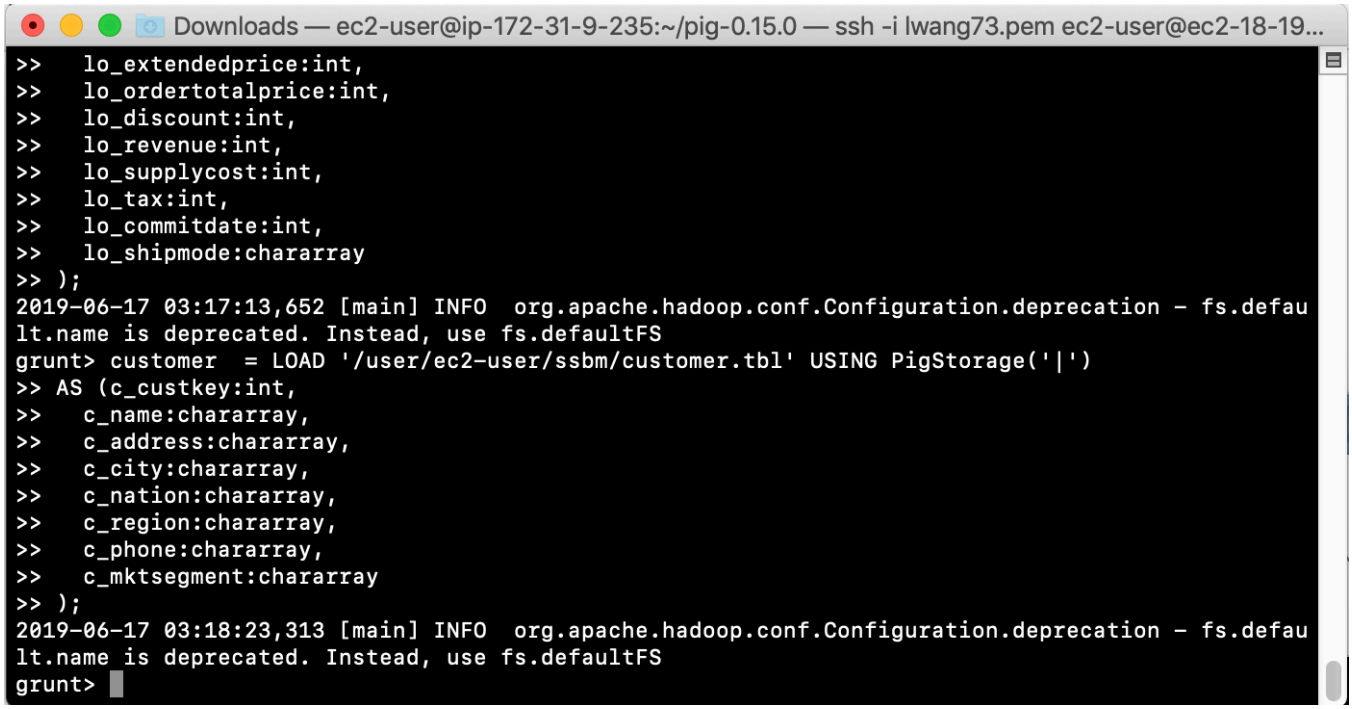
This one didn't work.

Pig:

Load Tables:

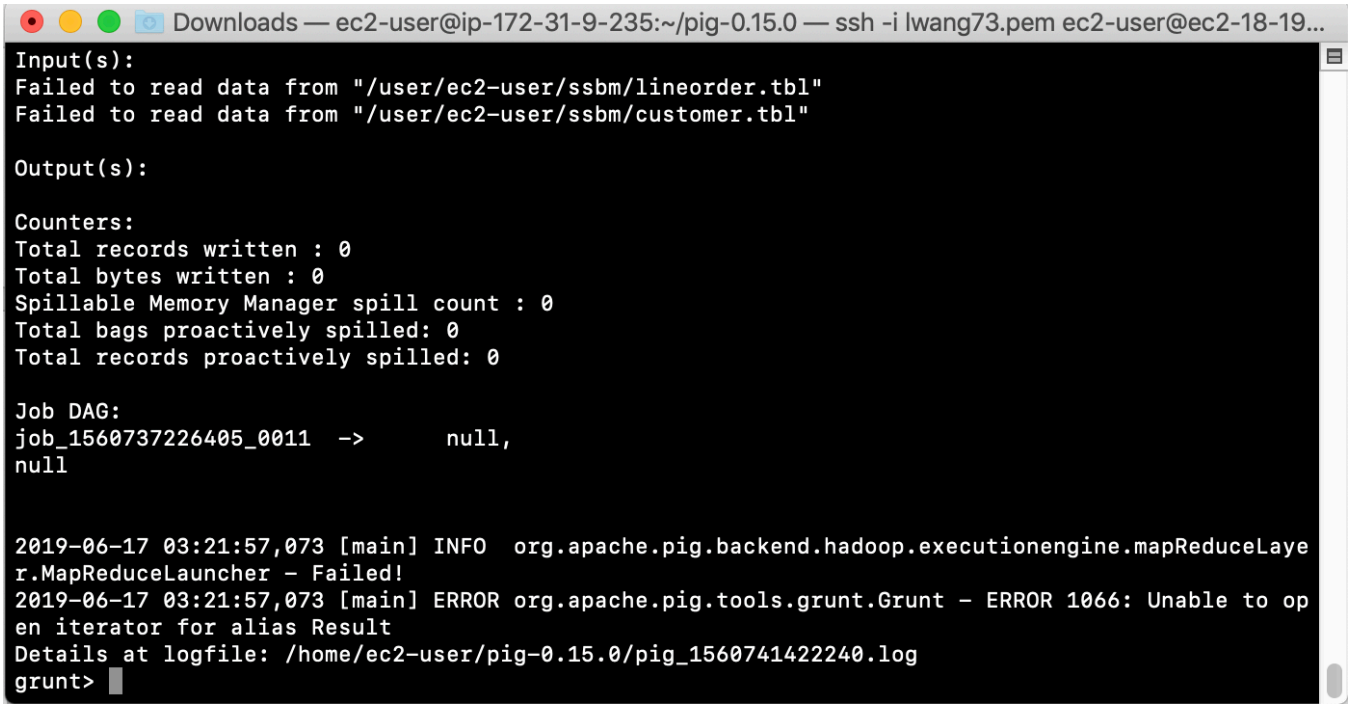
```
lineorder = LOAD '/user/ec2-user/ssbm/lineorder.tbl' USING PigStorage('|')
AS (lo_orderkey:int,
  lo_linenumber:int,
  lo_custkey:int,
  lo_partkey:int,
  lo_suppkey:int,
```

```
 lo_orderdate:int,
 lo_orderpriority:chararray,
 lo_shippriority:chararray,
 lo_quantity:int,
 lo_extendedprice:int,
 lo_ordertotalprice:int,
 lo_discount:int,
 lo_revenue:int,
 lo_supplycost:int,
 lo_tax:int,
 lo_commitdate:int,
 lo_shipmode:chararray
);

customer  = LOAD '/user/ec2-user/ssbm/customer.tbl' USING PigStorage('|')
AS (c_custkey:int,
 c_name:chararray,
 c_address:chararray,
 c_city:chararray,
 c_nation:chararray,
 c_region:chararray,
 c_phone:chararray,
 c_mktsegment:chararray
);
```



Execution steps:

```
FilteredLineorder = FILTER lineorder BY lo_discount >= 4 AND lo_discount <= 6;
FilteredCustomer = FILTER customer BY c_region == 'AMERICA';
JoinedData = JOIN FilteredLineorder BY (lo_custkey), FilteredCustomer BY (c_custkey);
GroupedData = GROUP JoinedData BY (c_nation);
Result = FOREACH GroupedData GENERATE group, SUM(JoinedData.lo_revenue) as rev;
DUMP Result;
```



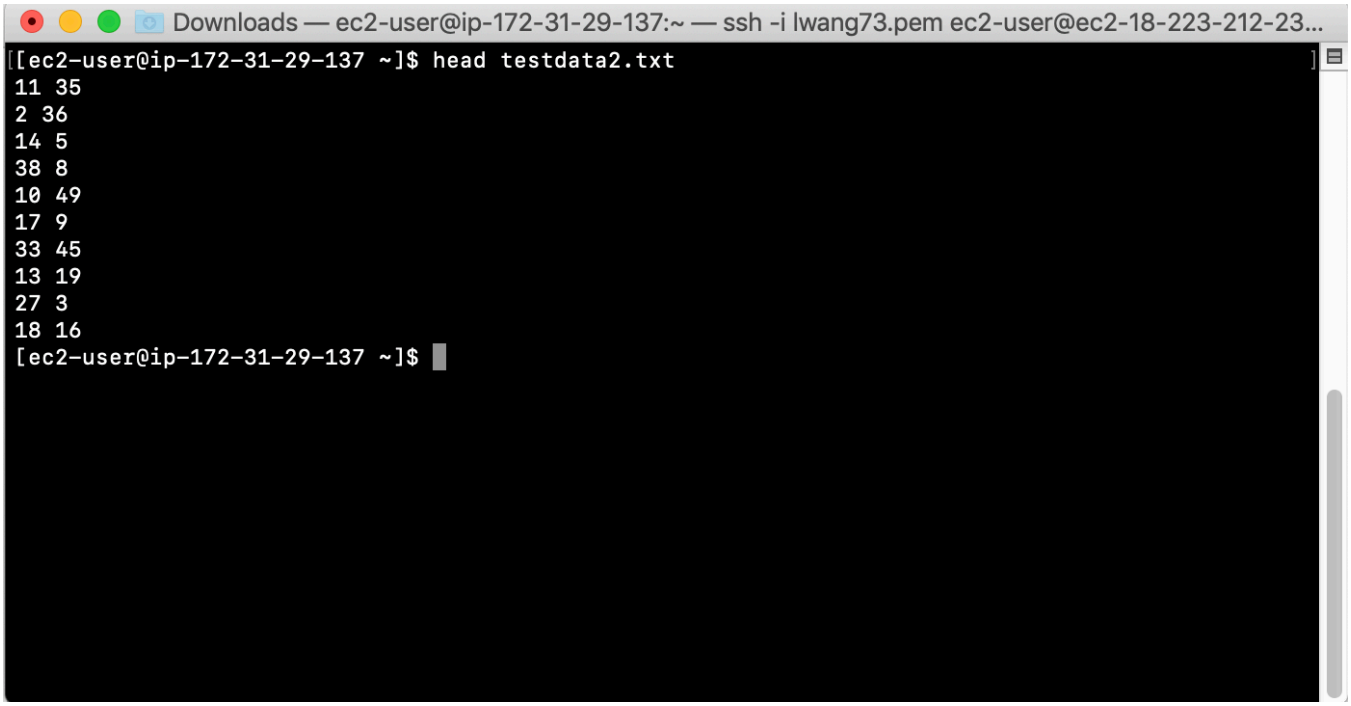
Part 3: Clustering (30 pts)

Create a new numeric file with 125,000 rows and 3 columns, separated by space – you can generate numeric data as you prefer, but submit the code that you have used.

A. (5 pts) Using Mahout synthetic clustering as you have in a previous assignment on sample data. This entails running the **same** clustering command, but substituting your own input data and the right number of clusters.

   Note, I used the single-node Hadoop instance for this exercise.

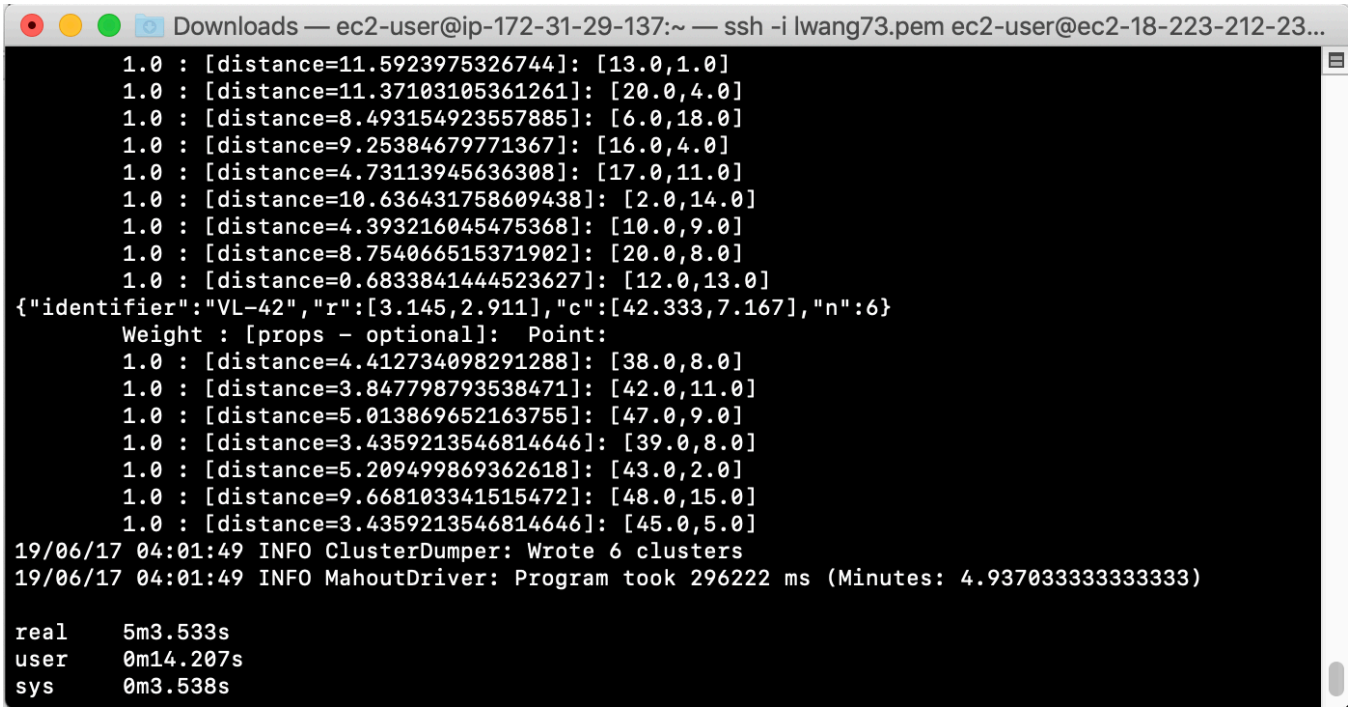
   First, I used an online random sequence generator to generate 100 x and y variables. Here's a screen shot of the first 10 records. The full list is at the end of this document.



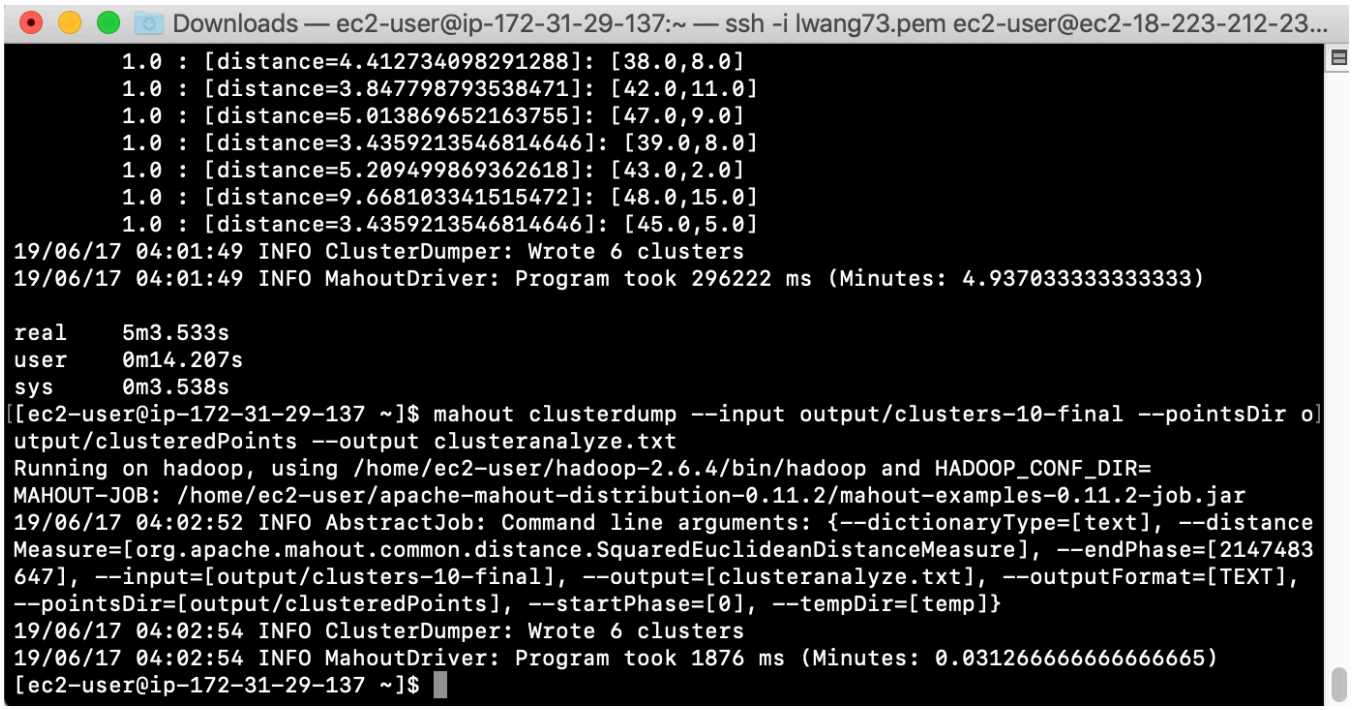
Commands:

hadoop fs –put testdata2.txt testdata/

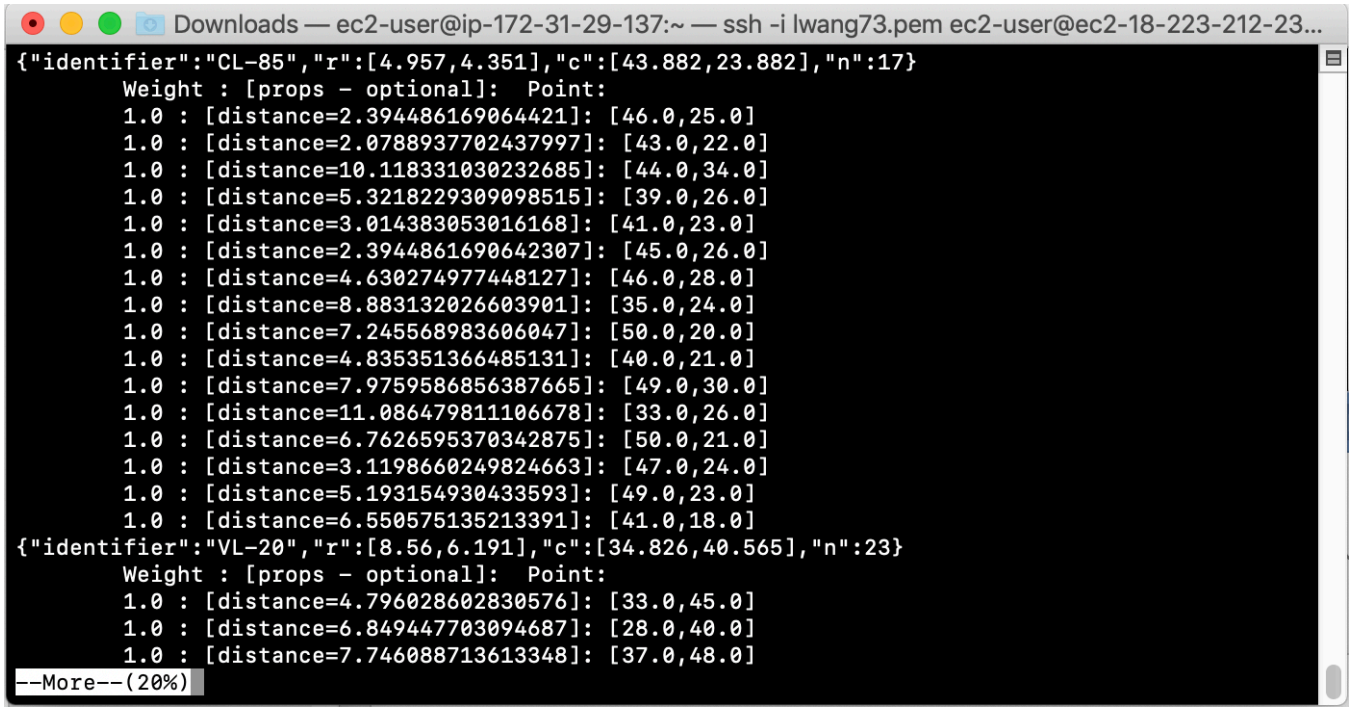
time mahout org.apache.mahout.clustering.syntheticcontrol.kmeans.Job



mahout clusterdump --input output/clusters-10-final --pointsDir output/clusteredPoints --output clusteranalyze.txt



more clusteranalyze.txt

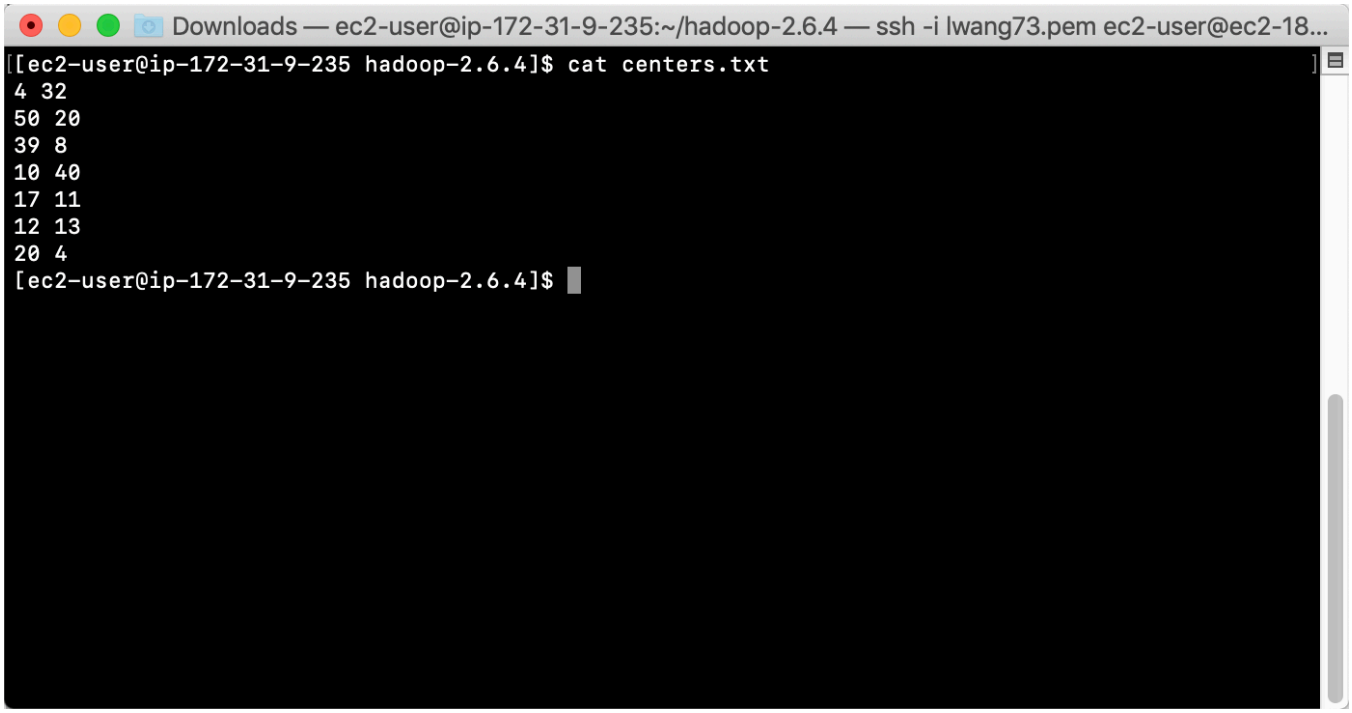


B. (25 pts) Using Hadoop streaming perform two iterations manually **using 7 centers** (initially with randomly chosen centers). As discussed in class, this would require passing a text file with cluster centers using -file option, opening the centers.txt in the mapper with open('centers.txt', 'r') and assigning a key to each point based on which center is the closest to each particular point. Your reducer would then compute the new centers, and at that point the iteration is done and the output of the reducer with new centers can be given to the next pass of the same code.

The only difference between first and second iteration is that in first iteration you have to pick the initial centers. In the 2nd iteration, the centers will be given to you by a previous pass of KMeans.

Note: I used the multi-node Hadoop instance for this exercise.

Using the same 100 points from exercise 3.A, I randomly picked seven starting centers from the list:



I used the same testdate2.txt file for this exercise as I did for part 3.A.

Code:

kmeansMapper.py

```
#!/usr/bin/python
```

```
import sys
import math

fd = open('centers.txt', 'r')
centers = []

for line in fd:
    line = line.strip()
    vals = line.split(' ')
    centers.extend([vals])
fd.close()

for line in sys.stdin:
    line = line.strip()
    vals = line.split(' ')

    clusterNum = None
    distance = None
    i = 0

    #compare to each center and store the smallest distance
    for center in centers:

        euclidDist = math.sqrt( (float(vals[0])-float(center[0]))**2 + (float(v$

        if clusterNum:
            if euclidDist < distance:
                clusterNum = i+1
                distance = euclidDist
        else: #always record the first cluster
            clusterNum = i+1
            distance = euclidDist

        i += 1

    print  clusterNum, '\t', vals[0], '\t', vals[1]
```

kmeansReducer.py

```
#!/usr/bin/python

import sys

currId = None  # this is the "current" key
currXs = []
currYs = []
id = None
```

```
# The input comes from standard input (line by line)
for line in sys.stdin:

    line = line.strip()
    ln = line.split('\t')
    id = ln[0]

    if currId == id:
        currXs.append(float(ln[1]))
        currYs.append(float(ln[2]))
    else:
        if currId:

            #calculate center
            centerX = sum(currXs)/len(currXs)
            centerY = sum(currYs)/len(currYs)
            print '%s %s %s %s' % (centerX, centerY, currId, zip(currXs, cu$

            currXs = []
            currYs = []

        currId = id
        currXs.append(float(ln[1]))
        currYs.append(float(ln[2]))

# output the last key
if currId == id:
    #calculate center
    centerX = sum(currXs)/len(currXs)
    centerY = sum(currYs)/len(currYs)
    print '%s %s %s %s' % (centerX, centerY, currId, zip(currXs, currYs))
```

Executions:

Execution 1:

hadoop jar hadoop-streaming-2.6.4.jar -input /user/ec2-user/data/testdata2.txt -file centers.txt -mapper kmeansMapper.py -file kmeansMapper.py -reducer kmeansReducer.py -file kmeansReducer.py -output /data/kmeans1

Execution 2:

Replace the centers file:

```
rm centers.txt
hadoop fs -get /data/kmeans1/part-00000 centers.txt
```

Run with new centers:

```
hadoop jar hadoop-streaming-2.6.4.jar -input /user/ec2-user/data/testdata2.txt -file centers.txt -mapper kmeansMapper.py -file kmeansMapper.py -reducer kmeansReducer.py -file kmeansReducer.py -output /data/kmeans2
```

testdata2.txt

11 35
2 36
14 5
38 8
10 49
17 9
33 45
13 19
27 3
18 16
28 40
4 32
46 25
21 15
29 7
43 22
37 48
44 34
20 12
31 6
39 26
41 23
50 47
24 30
1 42
45 26
46 28
1 48
27 44
35 24
29 32
7 23
16 5
14 18
50 20
39 36
40 21

2 33
10 31
4 22
42 11
6 13
47 9
30 8
37 17
3 19
12 38
34 15
25 43
49 41
17 7
35 11
39 8
43 2
48 15
24 25
23 50
41 38
3 16
12 21
22 46
13 1
20 4
37 33
36 19
5 29
34 47
14 28
42 44
49 30
6 18
10 40
31 45
27 26
32 9
28 30
33 26
43 42
16 4
50 21
47 24
27 29
7 35
15 31

49 23
41 18
17 11
2 14
10 9
25 3
22 34
20 8
44 38
40 46
36 19
48 39
12 13
1 37
6 32
45 5

Submit a single document containing your written answers. Be sure that this document contains your name and "CSC 555 Project Phase 2" at the top.