Note:

1. Change the virtual machine name to **YOUR NAME** (e.g. NguyenVanA).
2. For each problem below, take the **screenshots** of command for submitting the job (hadoop jar) and showing the output (-cat) **with the new virtual machine name**. Put these screenshots to a word file with descriptions.
3. Copy and paste the java code to the end of the word file.
4. Convert the word file to pdf and submit the pdf file

Problem 1:

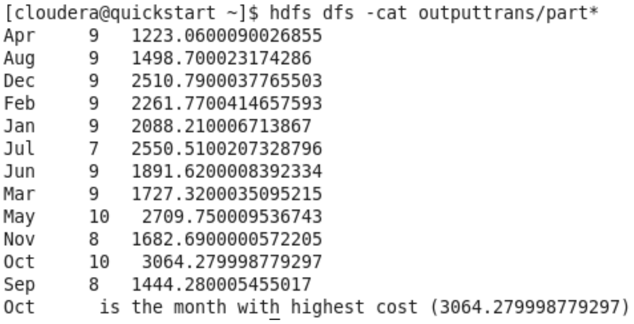
Given two separate datasets of a sports complex with the following schemas:

Cust ID	First Name	Last Name	Age	Profession

Trans ID	Date	Cust ID	Cost	Game	Equipment	City	State	Mode

Put **trans240_1.txt, trans240_2.txt,** and **cust.txt** to HDFS and perform the following queries using MapReduce:

A. For each month, show the number of distinct players, and the total cost. Then, show the month with highest total cost and the total cost. The output should have the format like below:

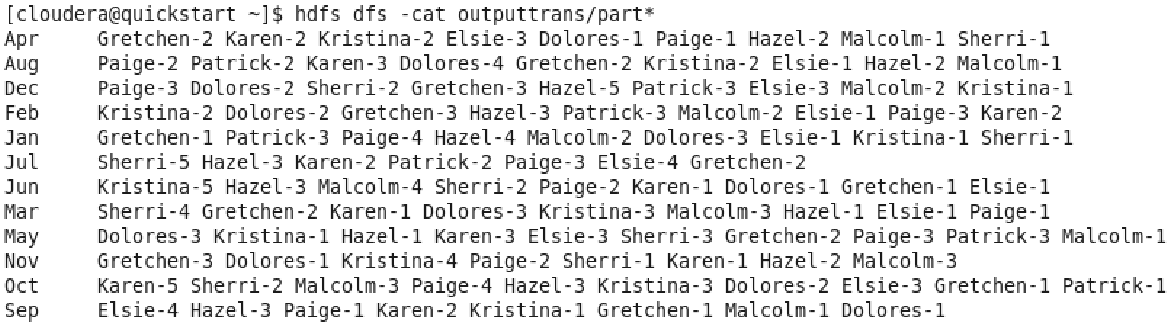


```
[cloudera@quickstart ~]$ hdfs dfs -cat outputtrans/part*
Apr     9   1223.0600090026855
Aug     9   1498.700023174286
Dec     9   2510.7900037765503
Feb     9   2261.7700414657593
Jan     9   2088.210006713867
Jul     7   2550.5100207328796
Jun     9   1891.6200008392334
Mar     9   1727.3200035095215
May     10   2709.750009536743
Nov     8   1682.6900000572205
Oct     10   3064.279998779297
Sep     8   1444.280005455017
Oct      is the month with highest cost (3064.279998779297)
```

*Hint: put the two transaction files to an input directory in HDFS. The month can be obtained by splitting the Date. The java code of this problem should be similar to that of TransAnalysis3. To get the month with highest total cost, you can create class-level variables **maxMonth, maxCost**, which will be updated every time the reduce() function run. Then you retrieve the values of these variables and write them out in cleanup method().*

```
@Override
 public void cleanup(Context context) throws IOException, InterruptedException {
          //write the month with highest cost
          context.write(new Text(maxMonth), new Text(" is the month with highest cost (" + String.valueOf(maxCost) +")") );
 }
```

B. For each month, show the list of first name of distinct players and the number of transactions of that player. The output should have the format like below:



```
[cloudera@quickstart ~]$ hdfs dfs -cat outputtrans/part*
Apr     Gretchen-2 Karen-2 Kristina-2 Elsie-3 Dolores-1 Paige-1 Hazel-2 Malcolm-1 Sherri-1
Aug     Paige-2 Patrick-2 Karen-3 Dolores-4 Gretchen-2 Kristina-2 Elsie-1 Hazel-2 Malcolm-1
Dec     Paige-3 Dolores-2 Sherri-2 Gretchen-3 Hazel-5 Patrick-3 Elsie-3 Malcolm-2 Kristina-1
Feb     Kristina-2 Dolores-2 Gretchen-3 Hazel-3 Patrick-3 Malcolm-2 Elsie-1 Paige-3 Karen-2
Jan     Gretchen-1 Patrick-3 Paige-4 Hazel-4 Malcolm-2 Dolores-3 Elsie-1 Kristina-1 Sherri-1
Jul     Sherri-5 Hazel-3 Karen-2 Patrick-2 Paige-3 Elsie-4 Gretchen-2
Jun     Kristina-5 Hazel-3 Malcolm-4 Sherri-2 Paige-2 Karen-1 Dolores-1 Gretchen-1 Elsie-1
Mar     Sherri-4 Gretchen-2 Karen-1 Dolores-3 Kristina-3 Malcolm-3 Hazel-1 Elsie-1 Paige-1
May     Dolores-3 Kristina-1 Hazel-1 Karen-3 Elsie-3 Sherri-3 Gretchen-2 Paige-3 Patrick-3 Malcolm-1
Nov     Gretchen-3 Dolores-1 Kristina-4 Paige-2 Sherri-1 Karen-1 Hazel-2 Malcolm-3
Oct     Karen-5 Sherri-2 Malcolm-3 Paige-4 Hazel-3 Kristina-3 Dolores-2 Elsie-3 Gretchen-1 Patrick-1
Sep     Elsie-4 Hazel-3 Paige-1 Karen-2 Kristina-1 Gretchen-1 Malcolm-1 Dolores-1
```

*Hint: this problem is similar to TransAnalysis5 (using MapSide Join). In the reduce() function, for each month, you can count the number of transaction of each player by using a **userCount** of type **Map<String, Integer>**. When encountering a new name which is not present in the **Names** list, you would increase the value corresponding to this name in the **userCount**.*

```
if (!Names.contains(name)) {
        Names.add(name);
        userCount.put(name,1);
}
 else {
        userCount.put(name, userCount.get(name)+1);
}
```

Problem 2:

Given three separate datasets of a sports complex with the following schemas:

File **cust20_1.txt** and **cust20_2.txt:**

Cust ID	First Name	Last Name	Age	Profession

File **profession20_1.txt** and **profession20_2.txt:**

Profession	Average Salary

File **trans240_20_1.txt** and **trans240_20_2.txt:**

Trans ID	Date	Cust ID	Cost	Game	Equipment	City	State	Mode

The sport complex is running a promotion. Therefore, they want to make a list of potential customers who have jobs with average salary **greater than** 70000 but join **less than** 12 transactions. The list should contain the following information:

Name Transaction_count Profession Salary

For example, a line in the list should look like this:

Patrick 10 Veterinarian 100300

Let's write a **chained** mapreduce job using **reduce side join** to output this list in a single **hadoop jar** command.