Question1- PSEUDOCODE

MAPPER-*USED FOR NOMAPPER AND COMBINER PARTS.*

```
Class AverageMapper{
        Map(){
            ->Split on ","
            ->check if the type of temperature is (TMIN/TMAX)
                yes->emit(stationId, "tempType,temperature")
            }
        }
```

REDUCER-Used in all the 3 cases

```
Class AvgReducer{
        Reduce{
            double maxTempSum = 0;
            double minTempSum = 0;
            Double maxCount= 0d;
            Double minCount=0d;
            Double maxAvg=null;
            Double minAvg=null;
            For(all the values){
                Check the length of the value
                ->if its 2 this means no combiner is used
                    then
                        ->check the type and then add the temp to the
                        minTempSum/maxTempSum, and increrment
                        minCount/maxCount based on the type.
                ->if not equal to 2 this means the combiner or
                  inMapper Combining is used
                    then
                        ->split on ","
                        ->maxTempSum+=split[0],
                        ->maxCount+=split[1],
                        ->minTempSum+=split[2],
                        ->minCount+=split[3]
                }
            emit(stationId, "Mean Minimum temp, Mean Maximum temp")
        }

        }
```

COMBINER

Takes in the Text as the Key and Text as the value.
Key- stationId, value-(maxTempSum,maxCount,minTempSum,minCount).
Reducer handles both cases that use the combiner and those that dont too.
Combines the data in the map call before emitting,thereby reducing the data sent.

```
class AvgCombiner{
        reduce(){
```

```
                double maxTempSum = 0;
                double minTempSum = 0;
                Double maxCount= 0d;
                Double minCount=0d;
        For(all values)
                ->Split on “,”
                ->check if the type of temperature is (TMIN/TMAX)
                        ->then depending on the type add sum temp to
        maxTempSum/minTempSum+=split[0]/split[2].
                        ->also increment the counter    maxCount/minCount+=split[1]/split[3].
        }
        emit(stationId,”maxTempSum, maxCount, minTempSum, minCount”)
}
```

INMAPPERCOMBINER

```
class InMapperCombiner{
        setup(){
        new hashmap inMapCombiner
        Key-stationId, Value-
[maxTempSum,maxCount,minTempSum,minCount]
        }
        Map(){
         for(all values){
                ->split on ","
                ->check if the stationId(split[0]) is present in inMapCombiner
                        ->if yes
                                if(split[1]="TMAX")
                                minMaxTemp[0]= minMaxTemp[0]+
Double.parseDouble(temprature(split[3]));
                                minMaxTemp[1]++;
                                inMapCombiner.put(stationId, minMaxTemp);
                                ->Do same for TMIN if the type(split[1]) is TMIN
                        ->if not upadate the count and sum variable for TMIN and
                        TMAX and add to the map.
        }
        }
        cleanup(){
                for(all keys in inMapCombiner){
                emit(key,"maxTempSum, maxCount,minTempSum, minCount")
                //here key is the stationId.
                }
        }
```

```
}
```

WEEK2:

SECONDARY SORT
ALGO IN NUTSHELL
The basic idea here is that we use the tupple of (staionId,year) as the key and to that we apply a partitioner which only partitions on staitionID, while the KeyComparator first checks for the stationId if they are equal it checks and sorts on the basis of year.

Then to this our GroupingComparator groups just on the basis of the staionId, which reduces the number of reduce calls to only one per staionID. These all ensure that in one reduce call we will have all the data for all years for a particular stationId in sorted order by year.

Then we just iterate and find the Mean Minimum and Mean Maximum temperature by staionId by year.

MyKey(CompositeKey)

```
CompositeKey "MyKey" is used as output from the Map and input to the Reduce.
class MyKey{
CompositeKey "MyKey" is used as output from the Map and input to the Reduce. This then contains getters and setters, and also a compareTo method
        public int compareTo(MyKey o) {
                objectComapre=comapreStaionId of o and this.
                if(objectComapre!=0){
                        return objectComapre;
                }
                return this.year.compareTo(o.year);
        }
}
```

MAPPER
A mapper is used to read the files line by line and present it to the map function as key -> Long and values -> Line then we split the lines and extract information from the line like the stationId, Year, Temperature, TMAX/TMIN. Ouput Key of the mapper will be -> MyKey -> StationId, Year
Value Output of mapper will be -> Text ->(tempType, temp) The reason for this choice is that we need to display the years(which can be extracted from the

key), the max average temp, min average temperature for a given StationId. And we need the years in station Id to be sorted so we will need to add it to the key and make a composite key, as the keys are sortable.

```
class SortMapper{
        //Map outputs a "MyKey" as the key and Text(Type,temperature)
        //where type is either TMAX or TMIN and temprature is corresponding
        //temperature.
        Map(){
                for(value in values){
                        split on ","
                        check value[1] i.e. type is TMAX or TMIN
                        if yes
                                emit(MyKey,"type, temperature")
                }
        }
}
```

REDUCER

The reducer will get the sorted list of station id's and corresponding values. The comparator would have sorted the list based on the years and the grouper would have grouped the list based on the station id and every reduce call will have all the temperatures of all the years for that particular stationId.

```
Class SortReducer{
                Reduce(){
                        double maxTempSum = 0;
                        double minTempSum = 0;
                        Double maxCount= 0d;
                        Double minCount=0d;
                        Double maxAvg=null;
                        Double minAvg=null;
                        String currentYear="null"
                        String temperature
                        For(all the values){

        If (change is the currentYear){
                String tempYear="("+ currentYear + ","+ minAvg+","+maxAvg+")";
        }
        ->if its 2 this means no combiner is used
```

then

->check the type and then add the temp to the minTempSum/maxTempSum, and increrment minCount/maxCount based on the type.

}

//emit once all the years for that stationID and done i.e. at the end of the reduce call.
emit(stationId, “Mean Minimum temp, Mean Maximum temp”)

}

}

PARTIONER

Partitioning only on the basis of StaionID and ignoring the year.
This ensures that all the records for a particular StationId are sent to a single Reducer.

```
Class MyPartioner{
        getPartition(){
        return Math.abs(key.getStationId().hashCode()%numPartitions);
}
}
```

GROUPCOMPARATOR

This ensures that a single reduce call gets all the records for a particular staionId(ignoring the year) and also the list we get will be sorted by year. Assuming the staionID has an entry for all the years we will get
something like:-

NOTE-The year will be extracted from the key and is not the part of the value.

(A,[(1880,TMAX,20),(1800,TMIN,-9)....(1889,TMAX,10)..]

```
compare(){
        return key1.getStationId().compareTo(key2.getStationId());
}
```

Q) Performance Comparison

	Start Time	End Time	Run 1	Run 2

NO COMBINER	(Run1) 18:44:16 (Run2) 19:06:06	18:45:33 19:07:28	77	82
COMBINER	(Run1) 19:43:26 (Run2) 19:31:39	19:44:42 19:32:57	76	78
INMAPPER COMBINER	(Run1) 20:03:26 (Run2) 20:47:32	20:04:43 20:48:47	77	75
Sec Sort	17:09:54	17:10:55	61	

Q) Was the Combiner called at all in program Combiner? Was it called more than once per Map task?

Answer-Yes the Combiner was called in the combiner. The following values verifies that the combiner ran during the process:
Combine input records=8798241
Combine output records=223783

The combiner was surely called at least once which is evident from the above values in the log file. But from this info to tell how many times it was called is not possible as combiner emits when the buffer fills up and we are not aware how many times and for which Map task this actually happened.

Q) What difference did the use of a combiner make in Combiner compared to no Combiner?

Answer- The combiner reduced the input records that were sent across the network by more than half. This reduced the time taken to move records across network.
Reduce input records=223782 – From combiner
Reduce input records=8798241 – From No Combiner.

“Spilled Records" means the total number of records that were written to disk during a job and includes both map and reduce side spills.(https://www.dezyre.com/questions/3781/mapreduce-what-is-spilled-records-count). So less the spill records less time needed for IO.

Spilled Records=17596482 – From No Combiner
Spilled Records=447564 – From Combiner

Q) Was the local aggregation effective in InMapperComb compared to NoCombiner?

Answer-Local aggregation was effective I guess in InMapperCombiner over NoCombiner. This is evident from the number of Reduce Input records and Spilled Records which are quite less in InMapperCombining than in NoCombiner.

Reduce input records=8798241 – From No Combiner.
Reduce input records=223783-From InMapper Combiner.

Spilled Records=17596482 – From No Combiner
Spilled Records= 447566 – From Combiner

Q) Which one is better, Combiner or InMapperCombiner? Briefly justify your answer.
Answer-I feel the InMapperCombining is the better of all the 3 approaches as

- It gives more control to the user and if used with caution it leads to great saving in the amount of data sent over the network(or I/O).
- Secondly this is surely going to execute the logic written as compared to a combiner which might or might not be called.

Q) How do the running times and accuracy of these MapReduce programs compare to the sequential implementation of per-station mean temperature? Modify, run, and time the sequential version of your HW1 program on the 1991.csv data. Make sure to change it to measure the end-to-end running time by including the time spent reading the file.

Answer-The MR program takes more time as compared to the sequential program. This may be attributed to the fact that in MR execution there is data transfer over the network. In case of the Sequential program the execution takes place on the single machine. The accuracy is pretty much the same as both of methods provide the same averages.

Sequential Execution - 39 sec
Map Reduce - from 70 – 80 sec