1. Map-Reduce Algorithm

1.1 – NoCombiner

```
class Mapper
{
        map(..., Line line)
        {
                parse (station, date, tempType, temp) from line

                if(tempType == "TMIN")
                        emit(station, (0,0,temp,1)); // (maxTempSum, maxTempCount, minTempSum,
                                                                                minTempCount)
                else if(tempType == "TMAX")
                        emit(station, (temp,1,0,0));

        }
}


class Reducer
{
  reduce(Station station, [(maxTempSum1, maxTempCount1, minTempSum1,  minTempCount1), ...])
  {
      totalMaxTempSum=0, totalMaxTempCount=0
      totalMinTempSum=0,  totalMinTempCount=0

      for all (maxTempSum, maxTempCount, minTempSum, minTempCount) in input list do
                totalMaxTempSum += maxTempSum
                totalMaxTempCount += maxTempCount
                totalMinTempSum += minTempSum
                totalMinTempCount += minTempCount

     meanMinTempStr = totalMinTempCount>0 ? totalMinTempSum/ totalMinTempCount : "NULL"
     meanMaxTempStr = totalMaxTempCount>0 ? totalMaxTempSum/ totalMaxTempCount : "NULL"

     emit(station + ", " + meanMinTempStr + ", " + meanMaxTempStr, NULL);
  }
}
```

1.2 – Combiner

```
class Mapper
{
        map(..., Line line)
        {
                parse (station, date, tempType, temp) from line

                if(tempType == "TMIN")
                        emit(station, (0,0,temp,1)); // (maxTempSum, maxTempCount, minTempSum,
                                                                                minTempCount)
                else if(tempType == "TMAX")
                        emit(station, (temp,1,0,0));

        }
}

class Combiner
{
        reduce(Station station, [(maxTempSum1, maxTempCount1, minTempSum1,
                                                                        minTempCount1), ...])
        {
                totalMaxTempSum=0, totalMaxTempCount=0
                totalMinTempSum=0, totalMinTempCount=0

                for all (maxTempSum, maxTempCount, minTempSum, minTempCount) in input list do
                        totalMaxTempSum += maxTempSum
                        totalMaxTempCount += maxTempCount
                        totalMinTempSum += minTempSum
                        totalMinTempCount += minTempCount

                emit(station, (totalMaxTempSum, totalMaxTempCount, totalMinTempSum,
                                                                        totalMinTempCount));
        }
}

class Reducer
{
   reduce(Station station, [(maxTempSum1, maxTempCount1, minTempSum1,  minTempCount1), ...])
   {
       totalMaxTempSum=0, totalMaxTempCount=0
       totalMinTempSum=0,  totalMinTempCount=0

       for all (maxTempSum, maxTempCount, minTempSum, minTempCount) in input list do
                totalMaxTempSum += maxTempSum
                totalMaxTempCount += maxTempCount
                totalMinTempSum += minTempSum
                totalMinTempCount += minTempCount
```

```
        meanMinTempStr = totalMinTempCount>0 ? totalMinTempSum/ totalMinTempCount : "NULL"
        meanMaxTempStr = totalMaxTempCount>0 ? totalMaxTempSum/ totalMaxTempCount : "NULL"

        emit(station + ", " + meanMinTempStr + ", " + meanMaxTempStr, NULL);
    }
}
```

1.3 – InMapperComb

```
class Mapper
{
        HashMap H

        setup()
        {
                H = new HashMap //key = station, value = (maxTempSum, maxTempCount,
                                                          minTempSum, minTempCount)
        }

        map(..., Line line)
        {
                parse (station, date, tempType, temp) from line

                if(tempType == "TMIN")
                        H[station].minTempSum += temp
                        H[station].minTempCount++

                else if(tempType == "TMAX")
                        H[station].maxTempSum += temp
                        H[station].maxTempCount++
        }

        cleanup()
        {
                for each Station s in H do
                        emit(s, H[s])
        }
}
```

```
class Reducer
{
   reduce(Station station, [(maxTempSum1, maxTempCount1, minTempSum1,  minTempCount1), ...])
   {
       totalMaxTempSum=0, totalMaxTempCount=0
       totalMinTempSum=0,  totalMinTempCount=0

       for all (maxTempSum, maxTempCount, minTempSum, minTempCount) in input list do
                   totalMaxTempSum += maxTempSum
                   totalMaxTempCount += maxTempCount
                   totalMinTempSum += minTempSum
                   totalMinTempCount += minTempCount

      meanMinTempStr = totalMinTempCount>0 ? totalMinTempSum/ totalMinTempCount : "NULL"
      meanMaxTempStr = totalMaxTempCount>0 ? totalMaxTempSum/ totalMaxTempCount : "NULL"

      emit(station + ", " + meanMinTempStr + ", " + meanMaxTempStr, NULL);
   }
}
```

1.4 – TemperatureTimeSeries

```
class Mapper
{
	map(..., Line line)
	{
		parse (station, date, tempType, temp) from line

		if(tempType == "TMIN")
			emit((station,date.year), (0,0,temp,1)); // (maxTempSum, maxTempCount,
                                                                 minTempSum, minTempCount)
		else if(tempType == "TMAX")
			emit((station,date.year), (temp,1,0,0));
	}
}

class Combiner
{
	reduce((station, year), [(maxTempSum1, maxTempCount1, minTempSum1,
                                                                 minTempCount1), ...])
	{
		totalMaxTempSum=0, totalMaxTempCount=0
		totalMinTempSum=0, totalMinTempCount=0

		for all (maxTempSum, maxTempCount, minTempSum, minTempCount) in input list do
			totalMaxTempSum += maxTempSum
			totalMaxTempCount += maxTempCount
			totalMinTempSum += minTempSum
			totalMinTempCount += minTempCount

		emit((station, year), (totalMaxTempSum, totalMaxTempCount, totalMinTempSum,
                                                                 totalMinTempCount));
	}
}

class Partitioner
{
	getPartition((station, year))
	{
		// partition only on station so that temp records of all the years comes to the same
		// reduce task for processing
		return myPartition(station)
	}
}
```

```
class KeyComparator
{
	compare((staion1, year2), (station2, year2))
	{
		// sort in ascending order of station first
		// if the station is equal, sort in ascending order of year
		cmp = station1.compareTo(station2);
		if (cmp != 0)
		{
			return cmp;
		}
		return year1.compareTo(year2);
	}
}

class GroupingComparator
{
	compare((staion1, year2), (station2, year2))
	{
		// sort in ascending order of station
		// Does not consider year for sorting, Hence two keys with the same station are
		// identical, no matter the year value
		return station1.compareTo(station2)
	}
}

class Reducer
{
  reduce((station, year), [(maxTempSum1, maxTempCount1, minTempSum1, minTempCount1), ...])
  {
	resultStr = station + ", ["
	totalMaxTempSum=0, totalMaxTempCount=0
	totalMinTempSum=0, totalMinTempCount=0
	currentYear = year

	for all (maxTempSum, maxTempCount, minTempSum, minTempCount) in input list do
	{
		if(currentYear == year)
		 {
			// Year is same as previous record so only merge the values to current total
			totalMaxTempSum += maxTempSum
			totalMaxTempCount += maxTempCount
			totalMinTempSum += minTempSum
			totalMinTempCount += minTempCount
		}
```

```
else
{
   // Year is changed from previous record so end of previous year records
   meanMinTempStr = totalMinTempCount>0 ? totalMinTempSum/totalMinTempCount : "NULL"
   meanMaxTempStr = totalMaxTempCount>0 ?totalMaxTempSum/totalMaxTempCount :  "NULL"

   resultStr += "(" + currentYear + ", " + meanMinTempStr +  ", " + meanMaxTempStr + "), ";

   // Change the Year and start a new iteration for current year
   currentYear = year;

   totalMaxTempSum = maxTempSum
   totalMaxTempCount = maxTempCount
   totalMinTempSum = minTempSum
   totalMinTempCount = minTempCount
}
}

meanMinTempStr = totalMinTempCount>0 ? totalMinTempSum/totalMinTempCount : "NULL"
meanMaxTempStr = totalMaxTempCount>0 ? totalMaxTempSum/totalMaxTempCount : "NULL"

// Append the mean temperatures of the last year
resultStr += "(" + currentYear + ", " + meanMinTempStr +  ", " + meanMaxTempStr + ")]";

emit(resultStr, NULL);

}
}
```

How reduce call works?

All the temperature records (of all years) of one station are processed in one reduce call. The records are sorted in station, year order, example:

(s1, 2001), (30,1,3,1)
(s1, 2001), (40,1,4,1)
(s1, 2002), (50,1,5,1)
(s1, 2003), (60,1,4,1)

All of the above four records will be processed in a single reduce call as the station is same for all the records. Our for loop will iterate four times and when we select year value in our third iteration then it will give us 2002 and not 2001.

So above program will give the output as follows:

s1, [(2001, 3.5, 35), (2002, 5, 50), (2003, 4, 60)]

2. Performance Comparison

Run time of five programs, except sequential(HW1) program all other programs are executed on EMR on aws using six m4.large machines (1 master, 5 workers) for given input.

All running times are in milliseconds. For the programs run on aws, time shown in controller column is taken from the controller log, and time shown in syslog column is calculated from the syslog file by taking the difference of first and last line's timestamp.

Program	Run-1 Timings		Run-2 Timings	
	Controller	Syslog	Controller	Syslog
NoCombiner	84000	77026	82000	76590
Combiner	76000	70653	76000	70,567
InMapperComb	76000	67184	74000	68426
Sequential		15533		15218
TemperatureTimeSeries	54000	46918	52000	45016

Note: I have considered run time of Controller log only for answering below questions, Syslog timings are just for reference.

Q-1 Was the Combiner called at all in program Combiner? Was it called more than once per Map task?

Yes, Combiner called in program Combiner. You can see below in syslog of Combiner program

Map output records=8798241
Map output bytes=316736676
Map output materialized bytes=4018316
Input split bytes=1598
Combine input records=8798241
Combine output records=223783
Reduce input groups=14135
Reduce shuffle bytes=4018316
Reduce input records=223783

Yes, Combiner was called more than once per Map task. For checking this, I have added loggers and input and output counters in my Combiner's code. These counters are initialized in each Combiner's setup call and incremented as per the processing done by Combiner's reduce call and finally it logs the counters in Combiner's cleanup method. I have checked the container's syslog of each map task and the combiner is called more than once for that specific Map task.

Q-2 What difference did the use of a Combiner make in Combiner compared to NoCombiner

NoCombiner	**Combiner**
Map input records=30868726	Map input records=30868726
Map output records=8798241	Map output records=8798241
Map output bytes=316736676	Map output bytes=316736676
Map output materialized bytes=59765899	Map output materialized bytes=4018316
Input split bytes=1598	Input split bytes=1598
Combine input records=0	Combine input records=8798241
Combine output records=0	Combine output records=223783
Reduce input groups=14135	Reduce input groups=14135
Reduce shuffle bytes=59765899	Reduce shuffle bytes=4018316
Reduce input records=8798241	Reduce input records=223783
Reduce output records=14135	Reduce output records=14135

As we can see here, in NoCombiner the map output records (8798241) are transferred to reducer from mapper so total 59765899 bytes are shuffled.

Whereas in Combiner, Map output records are combined and only 223783 records are transferred to reducer from mapper so only 4018316 bytes are shuffled.

So by using we drastically reduced the number of bytes shuffled from Mapper to reducer so it reduces data transfer cost. As we have smaller input on reduce side so sorting cost as well as reduce processing cost is also reduced.

We can see the difference in running time as well, Combiner program executed faster than NoCombiner program.

Q-3 Was the local aggregation effective in InMapperComb compared to NoCombiner?

NoCombiner	**InMapperCombiner**
Map input records=30868726	Map input records=30868726
Map output records=8798241	Map output records=223783
Map output bytes=316736676	Map output bytes=8056188
Map output materialized bytes=59765899	Map output materialized bytes=4018316
Input split bytes=1598	Input split bytes=1598
Combine input records=0	Combine input records=0
Combine output records=0	Combine output records=0
Reduce input groups=14135	Reduce input groups=14135
Reduce shuffle bytes=59765899	Reduce shuffle bytes=4018316
Reduce input records=8798241	Reduce input records=223783
Reduce output records=14135	Reduce output records=14135

Yes, the local aggregation was also effective in InMapperComb compared to NoCombiner. No of bytes shuffled were reduced from 59765899 to 4018316 due to InMapperCombining. So InMapperComb has also reduced shuffle and sort cost same as described in above answer.

We can see the difference in running time as well, InMapperComb program executed faster than NoCombiner program.

Q-4 Which one is better, Combiner or InMapperComb? Briefly justify your answer.

Combiner	**InMapperCombiner**
Map input records=30868726	Map input records=30868726
Map output records=8798241	Map output records=223783
Map output bytes=316736676	Map output bytes=8056188
Map output materialized bytes=4018316	Map output materialized bytes=4018316
Input split bytes=1598	Input split bytes=1598
Combine input records=8798241	Combine input records=0
Combine output records=223783	Combine output records=0
Reduce input groups=14135	Reduce input groups=14135
Reduce shuffle bytes=4018316	Reduce shuffle bytes=4018316
Reduce input records=223783	Reduce input records=223783
Reduce output records=14135	Reduce output records=14135

In our case both have reduced the mapper output to the same number and number of bytes shuffled are also same. Running times are also almost same for both programs. But in general both have their own advantages and disadvantages.

Combiners cannot be controlled by user, they are in control of Map-Reduce system. The Map-Reduce system decides when and on which map output records the combiner is executed. It can be executed 0 or many times in single Map task. Another disadvantage is that it combines the data after it was generated.

Whereas InMapperCombiner is in control of programmer. It avoids generating large amounts of intermediate data by immediately aggregating it as it is produced by a Map call which reduces both local CPU and disk I/O cost on the Mappers **(Due to this reason, running time of InMapperComb is slightly lower than Combiner's running time)**. On the other side it increases code complexity and it needs to hold HashMap H in mempry which can cause outofmememory exception if exceeds the size so need to handle that case as well. But if we have smaller input chunks in mapper call then we can use this effectively.

Q-5 How do the running times and accuracy of these MapReduce programs compare to the sequential implementation of per-station mean temperature?

As mentioned above, my sequential program has a smaller running time than all other Map-Reduce programs. This is because we are running on a comparatively smaller amount of data. Map-Reduce programs are very effective when it comes to bigger data. So if we increase the input size then the running time of a sequential approach will be drastically increased compared to Map-Reduce programs. I have provided my sequential program source code, log file and output file in my submission.

When we have a big data and higher number of machines to process then sequential program cannot take advantage of multiple available machines while Map-Reduce programs can. Due to parallel processing they will process the given big data faster than sequential program.

I have verified mean minimum and maximum temperatures generated by Map-Reduce and sequential programs and all are similar and correct.