Algorithm used for MapReduce Program

1. Mapper

(a) Declare 2 private static list namely days and hours
(b) List days contains "2017-12-24", "2017-12-25", "2017-12-26", "2017-12-27", "2017-12-28", "2017-12-29", "2017-12-30" as I plan to extract data only for these days.
(c) List hours contains "08", "09", "10", "11", "12", "13" as I plan to extract data only for these hours for dates mentioned in days list.
(d) Declare 1 private static IntWritable variable ONE for writing boxed integer value 1.
(e) Map function in Mapper class reads the file in (key, value) format
(f) Value is split on "," as delimiter.
(g) 1st field is taken as songId, 4th field is taken as streamHour and 5th field is taken as streamDate.
(h) If streamDate is one of present in days list and streamHour is one of present in hours list then form a string of format streamDate + ":" + songId and emit (key, value) pair where key is string formed of (streamDate + ":" + songId) and value is ONE.
(i) For each record from input data, if date is one of from days list and hour is one of from hours list, we emit (key, value) pair as mapper output where key is of the form (streamDate + ":" + songId) and value is ONE (1).

2. Partitioner

(a) I am using 7 reducers to avoid too much workload on single reducer and in order to ensure data for each day goes to a single reducer, I am using a partitioner so I can create 7 files (1 file for each date as output).
(b) Configurable interface is implemented by the custom partitioner class and both getConf() and setConf() methods are defined.
(c) In setConf() method which takes Configuration as input parameter, configuration is set and also a HashMap data structure is also setup which contains dates and a number for partitioner as key, value pairs.

```
days.put("2017-12-24", 0);
days.put("2017-12-25", 1);
days.put("2017-12-26", 2);
days.put("2017-12-27", 3);
days.put("2017-12-28", 4);
days.put("2017-12-29", 5);
days.put("2017-12-30", 6);
```

(d) In getConf() method, configuration setup is returned.
(e) In getPartition() method, which accepts key, value and numReduceTasks (no of reducers), key is split on ":" as delimiter and 1st field is chosen which contains date and this date is used as a key to get the value from days HashMap. Value returned is an integer ranging from 0 – 6 which is then returned to the caller.
(f) 0 – 6 (in total 7) are numbers for each reducer and based on this returned value, partitioner partitions the mapper output and ensures that data for a single day goes to a single reducer like data for 24-12-2017 will go to reducer 0, data for 25-12-2017 will go to reducer 1 and so on.

3. Combiner

(a) I am using reducer itself as a combiner so all values for each common key in a particular partition at all mappers can be grouped together before sending to reducer. There will be 7 partitions at each mapper containing data for each day in (key, value) form respectively. Values associated with same key for the same day in each partition will be grouped together by combiner before sending to reducer. This will reduce I/O overhead between mapper and reducer.

(b) Combiner converts output of mapper which is in (key, value) form into (key, list of values) form.

4. Reducer

(a) I am using 7 reducers and each reducer receives data for a single day in (key, list of values) pair where key is of the form streamDate +":" songId and value is a list like {1,1,1,1}.

(b) Each reducer receives data from each mapper for a particular day and performs aggregation on all (key, list of values) pairs.

(c) Reduce method is called for each key and list of values pair and all the values for that key are added (aggregated) and finally aggregated (key, value) pairs are emitted.

(d) For each loop is used to iterate over list of values for a given key and these values are added to get a final count and then (key, count) is emitted as (key, value) pair.

5. Driver

(a) Driver class extends configured class and implements Tool interface.

(b) In driver class, run() method is implemented. It accepts command line argument like input file path and output path. It defines job instance and set various parameters for the MapReduce job by using various APIs like:

```
job = Job.getInstance(conf, "Saavn Project");
job.setJarByClass(SaavnProjectDriver.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(Text.class);
job.setMapOutputKeyClass(Text.class);
job.setMapOutputValueClass(IntWritable.class);
job.setMapperClass(SaavnProjectMapper.class);
job.setReducerClass(SaavnProjectReducer.class);
job.setCombinerClass(SaavnProjectReducer.class);
job.setPartitionerClass(SaavnProjectPartitioner.class);
job.setNumReduceTasks(7);
FileInputFormat.addInputPath(job, inputFile);
FileOutputFormat.setOutputPath(job, outputDir);
```

(c) After all setup, it waits for completion.

(d) Upon successful completion, no of input records to map function and no of output records from reduce function are also printed along with job success message.

End of algorithm for Map Reduce program