# Lab - 19.3.2018

1. Create a text file with country names and state names. Sort the country names in the ascending order and for every country sort the state names in the descending order using map reduce programming.

Note:

```
Public class secsort {


   public static class CompositeKeyWritable implements
Writable,
             WritableComparable<CompositeKeyWritable> {


   private String deptNo;

   private String emp;


   public CompositeKeyWritable() {

   }


   public CompositeKeyWritable(String deptNo, String emp) {

      this.deptNo = deptNo;

      this.emp = emp;

   }


      public String toString() {
        return (new
StringBuilder().append(deptNo).append("\t")
                .append(emp)).toString();
    }
    public void readFields(DataInput dataInput) throws
IOException {
        deptNo = WritableUtils.readString(dataInput);
        --
    }

    public void write(DataOutput dataOutput) throws
IOException {
        ---
    }

    public int compareTo(CompositeKeyWritable objKeyPair) {
            int result =
deptNo.compareTo(objKeyPair.deptNo);
        if (0 == result) {
            result = emp.compareTo(objKeyPair.emp);
        }
        return result;
```

```
   }


   public  static class mapper1 extends

   Mapper<LongWritable, Text, CompositeKeyWritable,

NullWritable> {

   public void map(LongWritable key, Text value, Context

context)

         throws IOException, InterruptedException {


      if (value.toString().length() > 0) {

         String arrEmpAttributes[] =

value.toString().split(",");


         context.write(

               new CompositeKeyWritable(

                     arrEmpAttributes[1].toString(),

                     (arrEmpAttributes[0].toString())),

NullWritable.get());

      }


   }

}
```

```
   public static class SecondarySortBasicPartitioner
extends
   Partitioner<CompositeKeyWritable, NullWritable> {
   public int getPartition(CompositeKeyWritable key,
NullWritable value,
         int numReduceTasks) {

      return (key.deptNo.hashCode() % numReduceTasks);
   }
}

   public static class
SecondarySortBasicCompKeySortComparator extends
WritableComparator {

        protected SecondarySortBasicCompKeySortComparator()
{
            super(CompositeKeyWritable.class, true);
         }
   public int compare(WritableComparable w1,
WritableComparable w2) {
      CompositeKeyWritable key1 = (CompositeKeyWritable) w1;

      CompositeKeyWritable key2 = (CompositeKeyWritable) w2;

            int cmpResult =

key1.deptNo.compareTo(key2.deptNo);

            if (cmpResult == 0)

            {

            return -key1.emp.compareTo(key2.emp);


            }

            return cmpResult;

         }

      }




   public static class SecondarySortBasicGroupingComparator

extends WritableComparator {

        protected SecondarySortBasicGroupingComparator() {

            super(CompositeKeyWritable.class, true);

         }

public int compare(WritableComparable w1,

WritableComparable w2) {

            CompositeKeyWritable key1 =
(CompositeKeyWritable) w1;
            CompositeKeyWritable key2 =
(CompositeKeyWritable) w2;
            return key1.deptNo.compareTo(key2.deptNo);
         }
      }

   public static class SecondarySortBasicReducer extends
   Reducer<CompositeKeyWritable, NullWritable,
CompositeKeyWritable, NullWritable> {

public void reduce(CompositeKeyWritable key,
Iterable<NullWritable> values,  Context context) throws
IOException, InterruptedException {

      ----
   }
}
```