1. Write a MapReuduce program to perform secondary sorting on a sales dataset by region (alphabetically), then by total sales (descending).

```
East,10000
West,9000
East,12000
North,7000
East,7000
West,9500
North,7500
South,6000
```

```
import java.io.IOException;
import java.util.*;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class SecondarySortSimple {

  // Mapper
  public static class SortMapper extends Mapper<LongWritable, Text, Text,
IntWritable> {
    private Text region = new Text();
    private IntWritable sales = new IntWritable();

    @Override
    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {
      String[] parts = value.toString().split(",");
      if (parts.length == 2) {
        region.set(parts[0].trim());
        sales.set(Integer.parseInt(parts[1].trim()));
        context.write(region, sales);
      }
    }
  }

  // Reducer
  public static class SortReducer extends Reducer<Text, IntWritable, Text,
IntWritable> {
    private IntWritable result = new IntWritable();

    @Override
    public void reduce(Text key, Iterable<IntWritable> values, Context context)
        throws IOException, InterruptedException {
            // Collect sales values into a list
            List<Integer> salesList = new ArrayList<>();
            for (IntWritable val : values) {
                salesList.add(val.get());
            }

            // Sort sales descending
            Collections.sort(salesList, Collections.reverseOrder());

            // Output sorted sales for each region
            for (int s : salesList) {
                result.set(s);
                context.write(key, result);
            }
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Secondary Sort Simple");

        job.setJarByClass(SecondarySortSimple.class);
        job.setMapperClass(SortMapper.class);
        job.setReducerClass(SortReducer.class);
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(IntWritable.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

2. Write a MapReduce program that searches for a given keyword in a large text corpus and returns all matching lines.
Use Case: Searching logs for "ERROR", or filtering tweets containing "BigData".
Input Sample:
BigData is revolutionizing industries.
Cloud computing and BigData go hand in hand.
Artificial Intelligence is the future.
This line does not have the keyword.
Expected Output:
BigData is revolutionizing industries.
Cloud computing and BigData go hand in hand.

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class KeywordSearch {

    public static class KeywordMapper extends Mapper<LongWritable, Text,
NullWritable, Text> {
        private String keyword;

        @Override
        protected void setup(Context context) {
            // Read keyword from configuration
            keyword = context.getConfiguration().get("search.keyword").toLowerCase();
        }

        @Override
        public void map(LongWritable key, Text value, Context context)
                throws IOException, InterruptedException {
            String line = value.toString().toLowerCase();
            if (line.contains(keyword)) {
                context.write(NullWritable.get(), value);
            }
        }
    }

    public static void main(String[] args) throws Exception {
        if (args.length < 3) {
            System.err.println("Usage: KeywordSearch <input> <output> <keyword>");
            System.exit(-1);
        }

        Configuration conf = new Configuration();
        conf.set("search.keyword", args[2]);

        Job job = Job.getInstance(conf, "Keyword Search");
        job.setJarByClass(KeywordSearch.class);
        job.setMapperClass(KeywordMapper.class);
        job.setNumReduceTasks(0); // Mapper-only job

        job.setOutputKeyClass(NullWritable.class);
        job.setOutputValueClass(Text.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

TO RUN

```
hadoop fs -rm -r /user/hadoop/input

hadoop fs -mkdir -p /user/hadoop/input

hadoop fs -put input.txt /user/hadoop/input

--------------------------

nano <(class_name)>.java

javac -classpath $(hadoop classpath) -d units <(class_name)>.java

jar cf units.jar -C units/ .

hadoop jar units.jar <(class_name)> /user/hadoop/input
/user/hadoop/output

hadoop fs -cat /user/hadoop/output/part-r-00000
```