**SharedMemorySort.java**

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.RandomAccessFile;
import java.util.ArrayList;
import java.util.List;

public class SharedMemorySort
{
	//Declare No. of Threads, Files
	public static int nthreads=4;
	public static int nfiles = 10;
	public static List<File> fileNames = new ArrayList<File>();
	public static File input = new File("/home/ubuntu/64/New1");
	public static void main(String args[]) throws IOException
	{
		try
		{
			FileReader fr = new FileReader(input);
			BufferedReader br = new BufferedReader(fr);
			BufferedWriter fw = null;
			long csize = input.length()/(nfiles);
			String s=null;

			/*Read File and Split into Chunks with csize*/
			long start_t = System.currentTimeMillis();
			s = br.readLine();
			for(int j=0;j<nfiles;j++)
			{
				File file = new File("/home/ubuntu/64/Input"+j);
				fileNames.add(file);
				fw = new BufferedWriter(new FileWriter(file));
				int count=0;
				while(s!=null && count!=csize)
				{
					count+=100;
					fw.write(s+" \n");
					fw.flush();
					s = br.readLine();
				}
			}
			long end_t = System.currentTimeMillis();
			System.out.println("Split Files: "+(end_t-
start_t)/1000+"secs.");
			fw.close();
			br.close();

			/*Divide Files among Threads and Call Quick Sort*/
			start_t = System.currentTimeMillis();
			Thread th[] = new Thread[nthreads];
			for(int a=0;a<nthreads;a++)
			{
```

```
                        th[a] = new
ReadSort(a*(nfiles/nthreads),nfiles/nthreads);
                        th[a].start();
                  }

                  /*Wait for all Threads before Proceed to Merge Sort*/
                  for(int a=0;a<nthreads;a++)
                        th[a].join();
                  end_t = System.currentTimeMillis();
                  System.out.println("Quick Sort: "+(end_t-start_t)/1000+"
secs.");

                  /*Call Merge Sort*/
                  start_t = System.currentTimeMillis();
                  mergeSort(nfiles);
                  end_t = System.currentTimeMillis();
                  System.out.println("Merge Sort: "+(end_t-start_t)/1000+"
secs.");
            }
            catch(Exception e)
            {e.printStackTrace();}
      }

      /*K-Way Merge Sort*/
      public static void mergeSort(int nfiles)
      {
            List<String> merge = new ArrayList<String>();
            merge.clear();
            try
            {
                  BufferedReader br[] = new BufferedReader[nfiles];
                  BufferedWriter bw = null;
                  String s=null;
                  for(int i=0;i<nfiles;i++)
                  {
                        br[i] = new BufferedReader(new
FileReader(fileNames.get(i)));
                        s = br[i].readLine();
                        if(s!=null)
                              merge.add(i,s);
                  }
                  bw = new BufferedWriter(new
FileWriter("/home/ubuntu/64/Output"));
                  for(long f=0;f<input.length();f++)
                  {
                        String small = merge.get(0);
                        for(int j=0;j<nfiles;j++)
                        {

      if(small.substring(0,10).compareTo(merge.get(j).substring(0, 10))>0)
                              {
                                    small = merge.get(j);
                              }
                        }
                        bw.write(small+"\n");
                        bw.flush();
                        String next = br[merge.indexOf(small)].readLine();
```

```
                        if(next!=null)
                        {
                              merge.set(merge.indexOf(small),next);
                        }
                        else
                        {
                              merge.remove(small);
                              int index = merge.indexOf(small);
                              for(int j=0;j<nfiles;j++)
                              {
                                    if(j!=index)

while((small=br[j].readLine())!=null)
                                          merge.add(small);
                              }
                              while(merge.size()>0)
                              {
                                    next=merge.get(0);
                                    for(int j=0;j<merge.size();j++)
                                    {

if(next.substring(0,10).compareTo(merge.get(j).substring(0,10))>0)
                                          {
                                                next = merge.get(j);
                                          }
                                    }
                                    bw.write(next+"\n");
                                    bw.flush();
                                    merge.remove(next);
                              }
                              f=input.length();
                        }
                  }
                  for(int i=0;i<nfiles;i++)
                  {
                        br[i].close();
                  }
                  bw.close();
            }
            catch (Exception e)
            {
                  e.printStackTrace();
            }
      }

      /*Read Chunk files and Perform Quick Sort*/
      static class ReadSort extends Thread
      {
            List<String> data = new ArrayList<String>();
            int filestart,ntimes;
            BufferedReader br = null;
            BufferedWriter fw = null;
            public ReadSort(int filestart,int ntimes)
            {
                  this.filestart = filestart;
                  this.ntimes = ntimes;
            }
```

```
	public void run()
	{
		try
		{
			String s = null;
			for(int f=filestart;f<filestart+ntimes;f++)
			{
				data.clear();
				br = new BufferedReader(new
FileReader(fileNames.get(f)));
				s = br.readLine();
				while(s!=null)
				{
					data.add(s);
					s = br.readLine();
				}
				quickSort(0,data.size()-1);
				fw = new BufferedWriter(new
FileWriter(fileNames.get(f)));
				for(int i=0;i<data.size();i++)
				{
					fw.write(data.get(i)+"\n");
					fw.flush();
				}
			}
			br.close();
			fw.close();
		}
		catch (Exception e)
		{
			e.printStackTrace();
		}
	}

	/*Quick Sort*/
	public void quickSort(int start, int end)
	{
		int a=start , b = end;
		String temp = null;
		String pivot = data.get(start+(end-
start)/2).substring(0,10);
		while (a <= b)
		{
			while (data.get(a).substring(0,10).compareTo(pivot) <
0)
			{
				a++;
			}
			while (data.get(b).substring(0,10).compareTo(pivot) >
0)
			{
				b--;
			}
			if (a <= b)
			{
				temp = data.get(a);
```

```
                                    data.set(a, data.get(b));
                                    data.set(b, temp);
                                    a++;
                                    b--;
                              }
                        }
                        if (start < b)
                              quickSort(start, b);
                        if (a < end)
                              quickSort(a, end);
                  }
            }
}
```

**HadoopSort.java**

```
import java.io.IOException;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.lib.IdentityReducer;

public class HadoopSort
{
      //Mapper Class
      public static class MapSort extends MapReduceBase implements
Mapper<LongWritable, Text, Text, NullWritable>
      {
            public MapSort()
            {}

            @Override
            public void map(LongWritable value, Text key,
OutputCollector<Text, NullWritable> output, Reporter r)
                        throws IOException
            {
                  output.collect(new Text(key.toString()+" "),
NullWritable.get());

            }

      }

      public static void main(String[] args) throws IOException
      {
            JobConf conf = new JobConf(HadoopSort.class);
            conf.setJobName("HadoopSort");
```

```
            long start = System.currentTimeMillis();

            //Create File I/O Objects
            FileInputFormat.setInputPaths(conf, new Path(args[0]));
            FileOutputFormat.setOutputPath(conf, new Path(args[1]));

            //Set Mapper & Reducer
            conf.setMapperClass(MapSort.class);
            conf.setReducerClass(IdentityReducer.class);

            //Set Output Key/Value Class
            conf.setMapOutputKeyClass(Text.class);
            conf.setMapOutputValueClass(NullWritable.class);

            //Run
            JobClient.runJob(conf);

            long end = System.currentTimeMillis();
            System.out.println("Time: "+(end-start)/1000+" secs");
      }
}
```

**SparkSort.java**

```
import java.util.Arrays;
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaPairRDD;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.api.java.function.FlatMapFunction;
import org.apache.spark.api.java.function.PairFunction;
import scala.Tuple2;

public class SparkSort
{
      public static void main(String args[]) throws Exception
      {
            //Create Spark and Java Context
            SparkConf sparkConf = new SparkConf().setAppName("SparkSort");
            JavaSparkContext ctx = new JavaSparkContext(sparkConf);

            //Read File, Split Lines
            JavaRDD<String> textFile = ctx.textFile(args[0]);
            JavaRDD<String> lines = textFile.flatMap(new
FlatMapFunction<String, String>() {
                  public Iterable<String> call(String s) { return
Arrays.asList(s.split(" \n")); }
            });

            //Map Lines to Key/Value
            JavaPairRDD<String, String> pairs = lines.mapToPair(new
PairFunction<String, String, String>()
            {
                  public Tuple2<String, String> call(String s)
                  {
                        return new Tuple2<String, String>(s,"");
```

```
                }
            });

            //Sort By Key
            JavaPairRDD<String, String> sort = pairs.sortByKey().coalesce(1);

            //Save as Text File
            sort.saveAsTextFile(args[1]);
            ctx.close();
        }
}
```