```
####################################################################################
##############################     Shared – Memory ####################################
####################################################################################


import random
import string
import os
import time
from multiprocessing import Process, Lock


# http://codexpi.com/merge-sort-python-iterative-recursive-implementations/  #

def merge_sort(alist):
    for j in range(1, len(alist)):
        j *= 2
        for i in range(0, len(alist), j):
            left = alist[i:(i + (j / 2))]
            right = alist[i + (j / 2):j - i]
            l = m = 0

            while l < len(left) and m < len(right):
                if left[l] < right[m]:
                    m += 1
                elif left[l] > right[m]:
                    left[l], right[m] = right[m], left[l]
                    l += 1
            alist[i:i + (j / 2)], alist[i + (j / 2):j - i] = left, right

        return alist



def Do_Sorting_Thread(f):
    m = 0
    for cnt, i in enumerate(f.readlines()):

            dictn[i[:10]] = i[10:]

        if cnt % lines == 0:
            a = []
            c = 0
            a.append(i[:10])
            c += 1
            mutex.acquire()
            fp = open("./Split_Files/File_" + str(m) + ".txt", "wb")
            m += 1
            mutex.release()

        else:

            if c < (lines - 1):
                a.append(i[:10])

            else:
                a.append(i[:10])
                for i in merge_sort(a):
                    fp.write(i + "\n")
                fp.close()

            c += 1

    files = []

    for filess in os.listdir("./Split_Files/"):
        if filess.endswith(".txt"):
            files.append(filess)

    alist = []

    for i in files:
        fl = open("./Split_Files/" + i, "r")
        alist.extend(fl.read().split("\n"))
        fl.close()

    alist = list(set(filter(None, alist)))

    alist = merge_sort(alist)

    out = open('output_100gb.txt', 'w')

    for cnt, i in enumerate(alist):
        out.write(i + dictn[i])

    out.close()
    print alist


if __name__ == '__main__':

    dictn = dict()

    mutex = Lock()

    starttime = time.time()
```

```
file = open('./Data/100gb_data.txt', 'rb')

buffer = 1000
filesize = os.stat('./Data/100gb_data.txt').st_size
threads = [1,2,4,8]
lines = filesize / (buffer * 10)

for i in threads:
    t = Process(target=Do_Sorting_Thread, args=(filesize/i,))
    t.start()
        t.join()

file.close()

endtime = time.time()

#print 'Start Time:', starttime
#print 'End Time:', endtime

for i in threads:
        print "#"*30
        print " Time Elapsed by "+ str(i) + "Thread(s) = "+ str(endtime - starttime) + " seconds"
    print "#"*30

folder = './Split_Files'

files = [ f for f in os.listdir("./Split_Files") if f.endswith(".txt") ]

for f in files:
    file_path = os.path.join(folder, f)
    os.unlink(file_path)
```

```
###########################################################################################
##############################     Hadoop Sort  ###########################################
###########################################################################################




import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import java.io.*;


public class SortHadoop{
	public static class SortingMapper extends Mapper<Object, Text, Text, Text>{

		private Text keys = new Text();
		private Text values = new Text();

		public void map (Object key, Text value, Context context)throws IOException,
InterruptedException{

		String text1 = (value.toString()).substring(1,10);
		String text2  = (value.toString()).substring(10);
		keys.set(text1);
		values.set(text2);
		context.write(keys,values);
		}
	}
	public static class SortingReducer extends Reducer<Text, Text, Text, Text>{

		private Text outputkey = new Text();
		private Text outputvalue = new Text();
		public void reduce (Text key, Iterable<Text> values, Context context)throws
IOException, InterruptedException{

		outputkey = key;
		for (Text val : values){
			outputvalue = val;
		}
```

```
                context.write(outputkey,outputvalue);
                }

        }

        public static void main(String[] args) throws Exception{


                long startTime = System.currentTimeMillis();

                Configuration conf = new Configuration();
                Job job = Job.getInstance(conf, "Hadoop sort");
                job.setJarByClass(SortHadoop.class);
                job.setMapperClass(SortingMapper.class);
                job.setCombinerClass(SortingReducer.class);
                job.setReducerClass(SortingReducer.class);
                job.setOutputKeyClass(Text.class);
                job.setOutputValueClass(Text.class);
                FileInputFormat.addInputPath(job, new Path(args[0]));
                FileOutputFormat.setOutputPath(job, new Path(args[1]));
                long endTime = System.currentTimeMillis();

                long totalTime = endTime - startTime;


                if (job.waitForCompletion(true))
                {
                        System.out.println("Total Elapsed Time on Hadoop: " + totalTime);

                        System.exit(0);
                }

                else
                {
                        System.out.println("Total Elapsed Time on Hadoop: " + totalTime);
                        System.exit(1);
                }



        }

}
```

```
#######################################################################################
###############################     Spark Sort  ########################################
#######################################################################################




from pyspark import SparkContext
import sys



if(len(sys.argv) < 3 ):
    print "Use spark_sort.py inputPath outputPath"
    sys.exit(1);

sc = SparkContext("local","Spark Sort")
        # Read input and output path

inputPath = sys.argv[1]

print ('Path of input file ->' + inputPath)

outputPath = sys.argv[2]

print ('Path of output file ->' + outputPath)

distFile = sc.textFile(inputPath)

def flatMap(line):
    return line.split("\n")

def map(word):
    return (str(word[:10]),str(word[10:]))

def reduce(a,b):
    print "*"*50
    #print type(a)," & " ,type(b)
    print "Value of B is ", (b[0]+b[1])

    return (a,b)


counts = distFile.flatMap(flatMap).map(map).sortByKey().reduce(reduce)
#print counts
counts.saveAsTextFile(outputPath)
```