Client.java

```
public class Client {

    public static void main(String[] args) {
        // this program was built on a machine running Ubuntu 16.04.1 (Linux)
        // change depending on your platform (i know nothing about Windows)
        FileManip.file_prefix = "/tmp/";

        String final_sorted_data = "sorted_data";
        String starting_file = "data-file";

        try {
            System.out.print("Generating random data...");
            StringList sl = FileManip.generateRandomNumberFileGroup(
                    starting_file, // seed for filename generation
                    5,             // 5 separate files
                    100000000);   // 20000000 ints/file (~80MB/file)
            System.out.println("DONE");

            // pre-sort each file, individually
            System.out.print("Sorting separate smaller files...");
            for(int i = 0; i < sl.getSize(); i++) {
                String filename = sl.at(i);

                // in this way, the working set is NEVER larger than the contents of a single file
                int[] arr = FileManip.getArray(filename);
                QuickSort.Sort(arr);                // sort a chunk of random data
                FileManip.writeBinaryToFile(filename, arr); // write sorted data back to file
            }
            System.out.println("DONE");

            // data is now organized as a series of smaller sorted data files
            // this data will be combined later
            // verify that all of the smaller files have been sorted correctly
            for(int i = 0; i < sl.getSize(); i++) {
                System.out.print("Verifying " + sl.at(i) + "...");
                if(verifyOrderingOfFile(sl.at(i)))
                    System.out.println("PASS");
                else
                    System.out.println("FAIL");
            }

            File final_data = new File(FileManip.file_prefix + final_sorted_data);
            final_data.createNewFile();
            final_data.delete();
            final_data.createNewFile(); // guarantees that there is now an EMPTY file here

            // combine all the files
            System.out.print("Combining data files...");
```

```
        for(int i = 0; i < sl.getSize(); i++)
            combineSortedFiles(sl.at(i), final_sorted_data);
        System.out.println("DONE");

        System.out.print("Verifying final data file...");
        if(verifyOrderingOfFile(final_sorted_data))
            System.out.println("SUCCESS!");
        else {
            System.out.println("FAILURE");
            return; // no point in creating ASCII files if the data is wrong
        }

        System.out.print("Generating ASCII file from final data file...");
        FileManip.convertBinaryToAscii(final_sorted_data);
        System.out.println("DONE");

    } catch(Exception e) {
        e.printStackTrace();
    }
}

/**
 * @param filename
 * @return whether the given file is sorted correctly
 * @throws FileNotFoundException
 * @throws IOException
 */
public static boolean verifyOrderingOfFile(String filename)
        throws FileNotFoundException, IOException {
    DataInputStream dis = FileManip.openFileForInput(filename);

    int token = dis.readInt();

    while(dis.available() > 0) {
        int tmp = dis.readInt();

        if(tmp < token) {
            System.out.println("Data is not sorted correctly");
            return false;
        }

        token = tmp; // new target to test against
    }

    // made it all the way through
    return true;
}

/**
```

```
 * @param file1
 * @param file2
 * @throws java.io.FileNotFoundException
 * A temporary file is created, data from file1 and file2 is added to
 * it. temporary file is renamed as file2. original files are deleted
 * Method does not care about file sizes
 */
public static void combineSortedFiles(String file1, String file2)
        throws FileNotFoundException, IOException {
    boolean file1_good = false;
    boolean file2_good = false;

    DataInputStream ds1 = FileManip.openFileForInput(file1);
    DataInputStream ds2 = FileManip.openFileForInput(file2);

    // this filename was generated by button-mashing the keyboard
    String new_file_name = "tmp_7874hje7fy38dhhfd8e";
    FileOutputStream output = FileManip.openFileForOutput(new_file_name);

    // read at most 8 bytes of data
    int f1_token = Integer.MAX_VALUE;
    int f2_token = Integer.MAX_VALUE;

    if(ds1.available() > 0) {
        f1_token = ds1.readInt();
        file1_good = true;
    }

    if(ds2.available() > 0) {
        f2_token = ds2.readInt();
        file2_good = true;
    }

    // while both of the files have data
    while(file1_good && file2_good) {
        if(f1_token < f2_token) {
            // write integer, then verify next token
            output.write(ByteBuffer.allocate(4).putInt(f1_token).array());

            if(ds1.available() > 0)
                f1_token = ds1.readInt(); // prep next token
            else
                file1_good = false; // file has no more data

        } else {
            // write integer, then verify next token
            output.write(ByteBuffer.allocate(4).putInt(f2_token).array());

            if(ds2.available() > 0)
```

```
            f2_token = ds2.readInt();
        else
            file2_good = false;

    }
}

// one or both of the files has run out of data
if(file1_good) {
    output.write(ByteBuffer.allocate(4).putInt(f1_token).array());
    while(ds1.available() > 0)
        output.write(ByteBuffer.allocate(4).putInt(ds1.readInt()).array());
}

if(file2_good) {
    output.write(ByteBuffer.allocate(4).putInt(f2_token).array());
    while(ds2.available() > 0)
        output.write(ByteBuffer.allocate(4).putInt(ds2.readInt()).array());
}

// both input files have been combined into one, time to delete and rename
ds1.close();
ds2.close();

// delete both old files (their data is now in /tmp/tmp_7874hje7fy38dhhfd8e)
FileManip.deleteFile(file1);
FileManip.deleteFile(file2);

FileManip.renameFile(new_file_name, file2);
}
}
```

FileManip.java

```
import java.io.*;
import java.nio.*;
import java.util.*;

/**
 *
 * @author joey
 */
public class FileManip {
    public static String file_prefix = "/tmp/";
    private static Random rand = new Random();

    /**
     * @param filename
     * @param arr
     * @throws IOException
     * Write ASCII representation of integers to file
     */
    public static void writeAsciiToFile(String filename, int[] arr)
            throws IOException {
        PrintWriter pw =
            new PrintWriter(
                new BufferedWriter(
                    new FileWriter(
                        new File(file_prefix + filename)))); // ... so many allocations

        for(int i : arr)
            pw.println(i);

        pw.close();
    }

    /**
     * @param sl linked list of strings representing filenames
     * @throws FileNotFoundException
     * @throws IOException
     * print the first 10 binary integers in the file
     * prefixing each group with the filename
     */
    public static void printStringsFromFiles(StringList sl)
            throws FileNotFoundException, IOException {
        for(int i = 0; i < sl.getSize(); i++) {
            DataInputStream dis = openFileForInput(sl.at(i));
            System.out.println(sl.at(i));

            for(int j = 0; j < 10; j++)
                System.out.println(dis.readInt());
        }
```

```
    }

    /**
     * @param filename base filename, others are derived from this one
     * @param num_files number of files to split random numbers into
     * @param num_integers number of random numbers to generate
     * @return linked list of Strings
     * @throws java.io.FileNotFoundException
     */
    public static StringList generateRandomNumberFileGroup(
            String filename,
            int num_files,
            int num_integers) throws FileNotFoundException, IOException {

        if(num_integers % num_files != 0)
            throw new IllegalArgumentException("num_integers needs to be evenly divisible by
num_files");

        int numbers_per_file = num_integers / num_files;
        StringList sl = new StringList();

        for(int i = 0; i < num_files; i++) {
            String file = filename + i;
            sl.addString(file);
            // the + i is how unique names are generated for each file
            generateRandomNumberFile(file, numbers_per_file);
        }

        return sl;
    }

    /**
     * @param filename
     * @param arr
     * @throws FileNotFoundException
     * @throws IOException
     * Writes the contents of arr to file given by filename
     * current contents are overwritten
     */
    public static void writeBinaryToFile(String filename, int[] arr)
            throws FileNotFoundException, IOException {
        FileOutputStream fos = new FileOutputStream(file_prefix + filename);
        for(int i = 0; i < arr.length; i++)
            fos.write(ByteBuffer.allocate(4).putInt(arr[i]).array());
    }

    /**
     * @param filename
     * pretty self-explanatory delete a file
```

```
 * Wrapper over the File operations in Java
 */
public static void deleteFile(String filename) {
    File file = new File(file_prefix + filename);
    file.delete();
}

/**
 * @param oldfilename
 * @param newfilename
 * rename file, deleting existing file if needed
 */
public static void renameFile(String oldfilename, String newfilename) {
    File file = new File(file_prefix + newfilename);
    if(file.exists())
        file.delete(); // delete file if it exists

    File newfile = new File(file_prefix + oldfilename);
    newfile.renameTo(file);
}
/**
 * @param filename
 * @throws FileNotFoundException
 * @throws IOException
 * Generate a file with easy-to-read ASCII values
 * instead of raw binary values
 */
public static void convertBinaryToAscii(String filename)
        throws FileNotFoundException, IOException {
    DataInputStream dis  = openFileForInput(filename);

    // filename generated by button mashing the keyboard
    String temp_file_name = "djd84j47ry47ehd78fh38dj";

    PrintWriter pw =
        new PrintWriter(
            new BufferedWriter(
                new FileWriter(
                    new File(file_prefix + temp_file_name))));

    while(dis.available() > 0)
        pw.println(dis.readInt());

    dis.close();
    pw.close();

    // original file gets replaced
    renameFile(temp_file_name, filename);
}
```

```
/**
 * @param filename create file with this name
 * @param n number of random numbers to generate
 * @throws java.io.FileNotFoundException
 * Create file and fill with random numbers
 */
public static void generateRandomNumberFile(String filename, int n)
        throws FileNotFoundException, IOException {
    FileOutputStream fos = new FileOutputStream(file_prefix + filename);
    for(int i = 0; i < n; i++)
        fos.write(ByteBuffer.allocate(4).putInt(rand.nextInt()).array()); // fill with random numbers
}

/**
 * @param filename file to read data from
 * @return integer array containing data from file
 * @throws FileNotFoundException
 * read in an entire file of binary integer data
 */
public static int[] getArray(String filename)
        throws FileNotFoundException, IOException {

    File file = new File(file_prefix + filename);
    int nInts = ((int)file.length()) / 4; // number of 4-byte integers

    int[] iArr = new int[nInts];

    DataInputStream input =
            new DataInputStream(
                    new BufferedInputStream(
                            new FileInputStream(file_prefix + filename)));

    for(int i = 0; i < nInts; i++)
        iArr[i] = input.readInt();

    return iArr;
}

/**
 * @param filename
 * @return
 * @throws FileNotFoundException
 * @throws IOException
 * A wrapper over the FileOutputStream operations in Java
 */
public static FileOutputStream openFileForOutput(String filename)
        throws FileNotFoundException, IOException {
    FileOutputStream fos = new FileOutputStream(file_prefix + filename);
```

```
        return fos;
    }

    /**
     * @param filename
     * @return
     * @throws FileNotFoundException
     * @throws IOException
     * A wrapper over the DataInputStream operations in Java
     */
    public static DataInputStream openFileForInput(String filename)
            throws FileNotFoundException, IOException {
        DataInputStream input =
                new DataInputStream(
                        new BufferedInputStream(
                                new FileInputStream(file_prefix + filename)));

        return input;
    }

}
```

QuickSort.java

```
/**
 *
 * @author joey
 */
public class QuickSort {

    /**
     * @param d array to sort
     * Quick sort algorithm, shown to be faster than other algorithms
     * tried in the past
     */
    public static void Sort(int[] d) {
        Sort(d, 0, d.length-1);
    }

    /**
     * @param arr
     * @param low
     * @param high
     * QuickSort algorithm optimized for integers
     * http://www.geeksforgeeks.org/quick-sort/
     */
    private static void Sort(int[] arr, int low, int high) {
        if (low < high)
        {
            /* pi is partitioning index, arr[pi] is
              now at right place */
            int pi = partition(arr, low, high);

            // Recursively sort elements before
            // partition and after partition
            Sort(arr, low, pi-1);
            Sort(arr, pi+1, high);
        }
    }

    // rotate around high point
    private static int partition(int[] arr, int low, int high) {
        int pivot = arr[high];
        int i = (low-1); // index of smaller element
        for (int j=low; j<high; j++)
        {
            // If current element is smaller than or
            // equal to pivot
            if (arr[j] <= pivot)
            {
                i++;
```

```
            // swap arr[i] and arr[j]
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }

    // swap arr[i+1] and arr[high] (or pivot)
    int temp = arr[i+1];
    arr[i+1] = arr[high];
    arr[high] = temp;

    return i+1;
  }

}
```

StringList.java

```
/**
 *
 * @author joey
 */
public class StringList {
    public class StringNode {
        public StringNode next = null;
        public String str;

        public StringNode(String str, StringNode next) {
            this.next = next;
            this.str = str;
        }
    }

    StringNode head = null;
    private int size = 0;

    public void addString(String str) {
        head = new StringNode(str, head);
        size++;
    }

    public String getString() {
        String ret = head.str;
        head = head.next;
        size--;
        return ret;
    }

    public int getSize() {
        return size;
    }

    public String at(int index) {
        StringNode tmp = head;
        for(int i = 0; i < index; i++)
            tmp = tmp.next;
        return tmp.str;
    }

}
```

Output of sorting 100000000 (100 million) elements in 5 separate files. Previous runs with 1 billion elements ran in about 182 minutes but I forgot to get a screenshot of the output before exiting the program