```java
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Scanner;

/**
 *  COSC311 - Project 4
 *
 *  This application takes input of a file containing 10,000 integers and reads
 *      the numbers into an array. This array is then sorted using a Merge Sort
 *      algorithm, which implements recursive partitioning of the array until
 *      single element sub-arrays are reached, at which point adjacent
 *      sub-arrays are then merged together.
 *
 *  The user enters the name of the input file and names the output file to be
 *      created.
 *
 *  @author Mordechai Sadowsky, Robert Lafore
 *  @version 08-apr-2014
 */
public class MergeSort {

    private static final int SIZE = 10000;
    private static int[] theArray = new int[SIZE];
    private static final String PATH =
            "/Users/Mordechai/git/COSC311/Program4/src/";

    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);

        System.out.println("Welcome to MergeSort.");
       System.out.print("Please enter an input data file name: ");
        String inputName = keyboard.next();
        System.out.print("\nPlease enter an output data file name: ");
        String outputName = keyboard.next();

        Scanner fileInput = null;
        PrintWriter fileOutput = null;
        try {
            fileInput = new Scanner(new FileInputStream(PATH+inputName));
            File outFile = new File(PATH+outputName);  //creates a new file
            outFile.createNewFile();                   //on disk for output
            fileOutput = new PrintWriter(new FileOutputStream(outFile));
        }
        catch (IOException e) {
            System.out.println(e.getMessage());
            System.out.println("Don't forget to update file path name!");
            System.exit(1);
        }

        //read numbers from file into array
        for (int i = 0; i < SIZE; i++)
            theArray[i] = fileInput.nextInt();
        //timed sorting algorithm
        long initialTime = System.currentTimeMillis();
        mergeSort();
        long finalTime = System.currentTimeMillis();

        //write numbers from array out to file
        for (int i = 0; i < SIZE; i++)
            fileOutput.println(theArray[i]);

        System.out.print("File successfully sorted and output stored in");
        System.out.println(PATH+inputName);
        System.out.print("Sort algorithm execution time (in milliseconds): ");
        System.out.print(finalTime-initialTime);

        fileOutput.close();
        fileInput.close();
        keyboard.close();
    }

    /**
     *  This makes the initial call to <code>recMergeSort</code> and creates a
     *      duplicate array for performing the merges.
     */
    public static void mergeSort() {
        int[] workSpace = new int[SIZE];
        recMergeSort(workSpace, 0, SIZE-1);
    }

    /**
     *  This recursive method takes a sub-array, recurses on its left half,
     *      then on its right half, and then merges the two.
     *
     *  @param workSpace copy of main array
     *  @param lowerBound left end of the sub-array
     *  @param upperBound right end of the sub-array
     */
    private static void recMergeSort(int[] workSpace, int lowerBound,
                                        int upperBound) {
        if (lowerBound == upperBound)
            return;
        else {
            int mid = (lowerBound+upperBound)/2;
            recMergeSort(workSpace, lowerBound, mid);
            recMergeSort(workSpace, mid+1, upperBound);
            merge(workSpace, lowerBound, mid+1, upperBound);
        }
    }

    /**
     *  Takes the working copy of the main array, and merges two adjacent sub-
     *      arrays.
     *
     *  @param workSpace copy of main array
     *  @param lowPtr left end of first sub-array
     *  @param highPtr left end of second sub-array (one past right end of first
         )
```

```java
     *  @param upperBound right end of second sub-array
     */
    private static void merge(int[] workSpace, int lowPtr, int highPtr,
                                int upperBound) {
        int j = 0;
        int lowerBound = lowPtr;
        int mid = highPtr-1;
        int n = upperBound-lowerBound+1;
        while (lowPtr <= mid && highPtr <= upperBound) {
            if (theArray[lowPtr] < theArray[highPtr])
                workSpace[j++] = theArray[lowPtr++];
            else
                workSpace[j++] = theArray[highPtr++];
        }
        while (lowPtr <= mid) {
            workSpace[j++] = theArray[lowPtr++];
        }
        while (highPtr <= upperBound) {
            workSpace[j++] = theArray[highPtr++];
        }
        for (j=0; j<n; j++)
            theArray[lowerBound+j] = workSpace[j];
    }
}
```