```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 Quick Sort
 *      algorithm, which implements recursive partitioning of the array until
 *      sub-arrays smaller than ten elements are reached, which are then sorted
 *      using an Insertion Sort algorithm.
 *
 *  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 QuickSort {

    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 QuickSort.");
       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();
         quickSort();
         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 method starts the initial call to <code>recQuickSort</code> to
     *  begin partitioning and sorting the array.
     */
    public static void quickSort() {
         recQuickSort(0, SIZE-1);
    }

    /**
     *  This recursive method generates a pivot point; creates two sub-arrays,
     *      one with values all less than the pivot and the other all greater;
     *      and then repeats on the sub-arrays.
     *
     *  @param left is the left end of the partitioned sub-array
     *  @param right is the right end of the partitioned sub-array
     */
    public static void recQuickSort(int left, int right) {
         int size = right-left+1;
         if (size < 10)
             insertionSort(left, right);
         else {
             int pivot = generatePivot(left, right);
             int partition = partitionIt(left, right, pivot);
             recQuickSort(left, partition-1);
             recQuickSort(partition+1, right);
         }
    }

    /**
     *  This method compares the first, middle and last elements of a sub-array
     *      and sorts them.
     *
     *  @param left is the left end of the partitioned sub-array
     *  @param right is the right end of the partitioned sub-array
     *  @return the median value of the three compared elements as the pivot
     */
```

```
    public static int generatePivot(int left, int right) {
        int center = (left+right)/2;

        if (theArray[left] > theArray[center])
            swap(left, center);

        if (theArray[left] > theArray[right])
            swap(left, right);

        if (theArray[center] > theArray[right])
            swap(center, right);

        swap(center, right-1);

        return theArray[right-1];
    }

    /**
     *  Exchanges two values in the array.
     *
     *  @param index1 The index of the first value to be swapped
     *  @param index2 The index of the second value to be swapped
     */
    public static void swap(int index1, int index2) {
        int temp = theArray[index1];
        theArray[index1] = theArray[index2];
        theArray[index2] = temp;
    }

    /**
     *  This method takes a sub-array and rearranges its elements so that all
     *      of the elements on the left are less than the pivot value and all of
     *      the elements on the right are greater than the pivot.
     *  @param left is the left end of the partitioned sub-array
     *  @param right is the right end of the partitioned sub-array
     *  @param pivot is the pivot value about which the elements are rearranged
     *  @return The index of the dividing line between left and right.
     */
    public static int partitionIt(int left, int right, int pivot) {
        int leftPtr = left;
        int rightPtr = right - 1;
        while (true) {
            while (theArray[++leftPtr] < pivot) {}
            while (theArray[--rightPtr] > pivot) {}

            if (leftPtr >= rightPtr)
                break;
            else
                swap(leftPtr, rightPtr);
        }
        swap(leftPtr, right-1);
        return leftPtr;
    }

    /**
     *  Standard insertion sort algorithm used for sorting small sub-arrays.
```

```
     *
     *  @param is the left end of the partitioned sub-array
     *  @param right is the right end of the partitioned sub-array
     */
    public static void insertionSort(int left, int right) {
        int in, out;
        for (out = left+1; out <= right; out++) {
            int temp = theArray[out];
            in = out;
            while (in > left && theArray[in-1] >= temp) {
                theArray[in] = theArray[in-1];
                in--;
            }
            theArray[in] = temp;
        }
    }
}
```