```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 Heap Sort
 *      algorithm. This first rearranges the elements in the array to be an
 *      array representation of a heap, and then sorts the heap within the array.
 *
 *  The user enters the name of the input file and names the output file to be
 *      created.
 *
 *  @author Mordechai Sadowsky
 *  @version 08-apr-2014
 */
public class HeapSort {

    private static final int SIZE = 10000;
    private static int[] theHeap = new int[SIZE];
    private static int next = 0;
    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 HeapSort.");
        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++) {
            theHeap[i] = fileInput.nextInt();
            next++;
        }

        //timed sorting algorithm
        long initialTime = System.currentTimeMillis();
        heapify();
        heapSort();
        long finalTime = System.currentTimeMillis();

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

    /**
     *  Turns any array into a heap
     */
    public static void heapify() {
        //begins at the first node with children from the end of the array
        int i = (next-2)/2;

        //moves this node and all lower-indexed nodes down the heap
        for (int j = i; j >= 0; j--)
            trickleDown(j);
    }

    /**
     *  Transforms an array representation of a heap into a sorted array
     */
    public static void heapSort() {
        int limit = next;
        for (int i =1; i <= limit; i++) {
            int x = pop();
            theHeap[next] = x;
        }
    }

    /**
     *  Moves a node in a heap down the tree until all of its descendants are
     *      smaller than it.
     *
     * @param index of the node to be moved down the tree
     */
    public static void trickleDown(int index) {
        int largerChild;
        int value = theHeap[index];
        while (index < next/2) {
            int left = 2*index+1;
            int right = 2*index+2;
            if (right < next && theHeap[left] < theHeap[right])
                largerChild = right;
            else
                largerChild = left;

            if (value >= theHeap[largerChild])
                break;

            theHeap[index] = theHeap[largerChild];
            index = largerChild;
        }
        theHeap[index] = value;
    }

    /**
     *  Removes the largest value node in the heap and then fixes the resulting
     *      array to again be a heap
     * @return the top-most/largest valued element in the heap array
     */
    public static int pop() {
        int x = theHeap[0];
        theHeap[0] = theHeap[--next];
        trickleDown(0);
        return x;
    }
}
```