/\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bailey Thompson

\* Sorting (1.1)

\* 16 September 2016

\* Info: Sorts a random array of numbers.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

//declaring package

package sorting;

//declaring imports

import static java.lang.Integer.parseInt;

import java.util.Arrays;

import javax.swing.JOptionPane;

//declaring public class

public class Sorting {

int sizeOfArray, minNumber, maxNumber, hours, minutes, seconds, sortingType;

int[] randomNumbers;

Integer[] randomNumbersMerge;

String word = "", tempWord;

long totalTime, startTime;

//declaring main method

public static void main(String[] args) {

//sending to method BubbleSort

Sorting Sorting = new Sorting();

Sorting.Sorting();

}

//declaring private void method

private void Sorting() {

String[] buttons = {"Bubble Sort", "Selection Sort", "Insertion Sort", "Merge Sort", "Quick Sort"};

sortingType = JOptionPane.showOptionDialog(null, "Please pick your sorting mechanism.", "Sorting Program", JOptionPane.PLAIN\_MESSAGE, JOptionPane.PLAIN\_MESSAGE, null, buttons, buttons[2]);

sortingType += 1;

if (sortingType == 0) {

System.exit(0);

}

CheckEmpty();

UserInput();

RandomArrayFill();

//setting start time from system time

startTime = System.nanoTime();

switch (sortingType) {

case 1:

BubbleSort();

break;

case 2:

SelectionSort();

break;

case 3:

InsertionSort();

break;

case 4:

mergeSort(randomNumbersMerge);

break;

case 5:

int lowQuickSort = 0;

int highQuickSort = randomNumbers.length - 1;

quickSort(randomNumbers, lowQuickSort, highQuickSort);

break;

}

//setting total runtime

totalTime = System.nanoTime() - startTime;

//setting seconds from nanoseconds

totalTime /= 1000000000;

//calculating hours

hours = (int) Math.floor(totalTime / 3600);

//calculating minutes

minutes = (int) Math.floor((totalTime - hours \* 3600) / 60);

//calculating seconds

seconds = (int) Math.floor(totalTime - minutes \* 60);

Output();

}

//declaring private void method

private void UserInput() {

do {

word = tempWord = JOptionPane.showInputDialog(null, "Please input the size of the array.\nSizes must be a real positive integer value.", "Sorting Program", JOptionPane.PLAIN\_MESSAGE);

CheckEmpty();

tempWord = tempWord.replaceAll("[ 0123456789]", "");

} while (!"".equals(tempWord) || "".equals(word));

sizeOfArray = parseInt(word);

if ("4".equals(sortingType)) {

randomNumbersMerge = new Integer[sizeOfArray];

} else {

randomNumbers = new int[sizeOfArray];

}

do {

word = tempWord = JOptionPane.showInputDialog(null, "Please input the minimum value.\nValue must be a real integer value.", "Sorting Program", JOptionPane.PLAIN\_MESSAGE);

CheckEmpty();

tempWord = tempWord.replaceAll("[ -0123456789]", "");

} while (!"".equals(tempWord) || "".equals(word));

minNumber = parseInt(word);

do {

word = tempWord = JOptionPane.showInputDialog(null, "Please input the maximum value.\nValue must be a real integer value.", "Sorting Program", JOptionPane.PLAIN\_MESSAGE);

CheckEmpty();

tempWord = tempWord.replaceAll("[ -0123456789]", "");

} while (!"".equals(tempWord) || "".equals(word));

maxNumber = parseInt(word);

while (minNumber > maxNumber || !"".equals(tempWord) && "".equals(word)) {

word = tempWord = JOptionPane.showInputDialog(null, "Please input the maximum value.\nValue must be a real integer value.\nMaximum value also cannot be less than minimum value.", "Sorting Program", JOptionPane.PLAIN\_MESSAGE);

CheckEmpty();

tempWord = tempWord.replaceAll("[ -0123456789]", "");

if (!"".equals(word)) {

maxNumber = parseInt(word);

}

}

}

//declaring private void method

private void RandomArrayFill() {

for (int counterFill = 0; counterFill < sizeOfArray; counterFill++) {

if ("4".equals(sortingType)) {

randomNumbersMerge[counterFill] = ((int) (Math.random() \* (maxNumber - minNumber + 1))) + minNumber;

} else {

randomNumbers[counterFill] = ((int) (Math.random() \* (maxNumber - minNumber + 1))) + minNumber;

}

}

}

//declaring private void method

private void BubbleSort() {

int temp;

for (int i = 0; i < sizeOfArray; i++) {

for (int j = 1; j < (sizeOfArray - i); j++) {

if (randomNumbers[j - 1] > randomNumbers[j]) {

temp = randomNumbers[j - 1];

randomNumbers[j - 1] = randomNumbers[j];

randomNumbers[j] = temp;

}

}

}

}

//declaring private void method

private void SelectionSort() {

int temp;

for (int counter1 = 0; counter1 < sizeOfArray - 1; counter1++) {

int minIndex = counter1;

for (int counter2 = counter1 + 1; counter2 < sizeOfArray; counter2++) {

if (randomNumbers[minIndex] > randomNumbers[counter2]) {

minIndex = counter2;

}

}

if (minIndex != counter1) {

temp = randomNumbers[counter1];

randomNumbers[counter1] = randomNumbers[minIndex];

randomNumbers[minIndex] = temp;

}

}

}

//declaring private void method

private void InsertionSort() {

int counter1, counter2, newValue;

for (counter1 = 1; counter1 < sizeOfArray; counter1++) {

newValue = randomNumbers[counter1];

counter2 = counter1;

while (counter2 > 0 && randomNumbers[counter2 - 1] > newValue) {

randomNumbers[counter2] = randomNumbers[counter2 - 1];

counter2--;

}

randomNumbers[counter2] = newValue;

}

}

public static void mergeSort(Comparable[] randomNumbersMerge) {

Comparable[] tmp = new Comparable[randomNumbersMerge.length];

mergeSort(randomNumbersMerge, tmp, 0, randomNumbersMerge.length - 1);

}

private static void mergeSort(Comparable[] randomNumbersMerge, Comparable[] tmp, int left, int right) {

if (left < right) {

int center = (left + right) / 2;

mergeSort(randomNumbersMerge, tmp, left, center);

mergeSort(randomNumbersMerge, tmp, center + 1, right);

merge(randomNumbersMerge, tmp, left, center + 1, right);

}

}

private static void merge(Comparable[] randomNumbersMerge, Comparable[] tmp, int left, int right, int rightEnd) {

int leftEnd = right - 1;

int k = left;

int num = rightEnd - left + 1;

while (left <= leftEnd && right <= rightEnd) {

if (randomNumbersMerge[left].compareTo(randomNumbersMerge[right]) <= 0) {

tmp[k++] = randomNumbersMerge[left++];

} else {

tmp[k++] = randomNumbersMerge[right++];

}

}

while (left <= leftEnd) // Copy rest of first half

{

tmp[k++] = randomNumbersMerge[left++];

}

while (right <= rightEnd) // Copy rest of right half

{

tmp[k++] = randomNumbersMerge[right++];

}

// Copy tmp back

for (int i = 0; i < num; i++, rightEnd--) {

randomNumbersMerge[rightEnd] = tmp[rightEnd];

}

}

private static void quickSort(int[] arr, int lowQuickSort, int highQuickSort) {

if (arr == null || arr.length == 0) {

return;

}

if (lowQuickSort >= highQuickSort) {

return;

}

// pick the pivot

int middle = lowQuickSort + (highQuickSort - lowQuickSort) / 2;

int pivot = arr[middle];

// make left < pivot and right > pivot

int i = lowQuickSort, j = highQuickSort;

while (i <= j) {

while (arr[i] < pivot) {

i++;

}

while (arr[j] > pivot) {

j--;

}

if (i <= j) {

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

i++;

j--;

}

}

// recursively sort two sub parts

if (lowQuickSort < j) {

quickSort(arr, lowQuickSort, j);

}

if (highQuickSort > i) {

quickSort(arr, i, highQuickSort);

}

}

//declaring private void method

private void Output() {

if (sizeOfArray < 100 && !"4".equals(sortingType)) {

JOptionPane.showConfirmDialog(null, "The Sorted Numbers Are: " + Arrays.toString(randomNumbers), "Sorting Program", JOptionPane.OK\_CANCEL\_OPTION, JOptionPane.PLAIN\_MESSAGE);

} else if (sizeOfArray < 100 && "4".equals(sortingType)) {

JOptionPane.showConfirmDialog(null, "The Sorted Numbers Are: " + Arrays.toString(randomNumbersMerge), "Sorting Program", JOptionPane.OK\_CANCEL\_OPTION, JOptionPane.PLAIN\_MESSAGE);

} else if (hours >= 1) {

JOptionPane.showConfirmDialog(null, "It took you:\n" + hours + " hours\n" + minutes + " minutes\n" + seconds + " seconds", "Sorting Program", JOptionPane.OK\_CANCEL\_OPTION, JOptionPane.PLAIN\_MESSAGE);

} else if (minutes >= 1) {

JOptionPane.showConfirmDialog(null, "It took you:\n" + minutes + " minutes\n" + seconds + " seconds", "Sorting Program", JOptionPane.OK\_CANCEL\_OPTION, JOptionPane.PLAIN\_MESSAGE);

} else {

JOptionPane.showConfirmDialog(null, "It took you:\n" + seconds + " seconds", "Sorting Program", JOptionPane.OK\_CANCEL\_OPTION, JOptionPane.PLAIN\_MESSAGE);

}

}

//declaring private void method

private void CheckEmpty() {

if (word == null) {

System.exit(0);

}

}

}