import java.util.Scanner;

import java.util.Random;

import javax.swing.\*;

import java.awt.\*;

import java.applet.\*;

import java.awt.Graphics;

public class MultiThreadSorting

{

public static final String RED="\u001B[31m";

public static final String GREEN="\u001B[32m";

public static final String PURPLE="\u001B[35m";

public static final String BLUE="\u001B[34m";

public static final String CYAN="\u001B[36m";

public static final String RESET="\u001B[0m";

public static void finalMerge(int[] a, int[] b)

{

int[] result = new int[a.length + b.length];

int i=0;

int j=0;

int r=0;

System.out.println("\n\n\n\n");

String bold= "\033[1mTwo Sorted Array's:\033[0m";

System.out.print(bold);

System.out.print("\t"+CYAN+"|");

int k;

for (k=0; k<a.length-1; k++)

{

System.out.print(CYAN+a[k]+" |");

}

System.out.print(CYAN+a[k]+"|"+RESET);

System.out.print("\t "+GREEN+"|");

for (k=0; k<b.length-1; k++)

{

System.out.print(GREEN+b[k]+" |");

}

System.out.print(GREEN+b[k]+"|"+RESET);

System.out.println("\n\n\n\n\n\n");

/\*

for(int k=0;k<5;k++)

{

System.out.println("\t\t\t\t\t\t |");

}

System.out.println("\t\t\t\t\t\t V");

\*/

while (i < a.length && j < b.length)

{

if (a[i] <= b[j])

{

result[r]=a[i];

i++;

r++;

}

else

{

result[r]=b[j];

j++;

r++;

}

if (i==a.length)

{

while (j<b.length)

{

result[r]=b[j];

r++;

j++;

}

}

if (j==b.length)

{

while (i<a.length)

{

result[r]=a[i];

r++;

i++;

}

}

} //while

bold= "\033[1mFinal Array:\033[0m";

System.out.print(bold);

System.out.print("\t\t"+BLUE+"|");

for (i=0; i<result.length-1; i++)

{

System.out.print(BLUE+result[i]+" |");

}

System.out.print(result[i]+"|"+RESET);

System.out.println("\n\n\n\n\n");

/\* JTextArea area = new JTextArea(" ");

area.setForeground(Color.red);

\*/

} //final merge

public static void main(String[] args) throws InterruptedException

{

for(;;)

{

System.out.println("\n\n\n");

Random rand = new Random();

System.out.print("Enter the ArraySize(preferably <=20) :");

Scanner s = new Scanner(System.in);

int size = s.nextInt();

int[] original = new int[size];

String bold= "\033[1mOrigianl Array:\033[0m";

System.out.print(bold);

System.out.print("\t\t");

int i;

System.out.print("|");

for (i=0; i<original.length-1; i++)

{

original[i] = rand.nextInt(100)+7;

System.out.print(original[i]+" |");

}

System.out.println(original[i]+"|"+"\n"); //last element without ","

/\*for(i=0;i<5;i++)

System.out.print("\t\t\t\t\t|\t\t\t|\n");

System.out.println("\t\t\t\t\tV\t\t\tV\n");

\*/

long startTime = System.currentTimeMillis();

int[] subArr1 = new int[original.length/2];

int[] subArr2 = new int[original.length - original.length/2];

System.arraycopy(original, 0, subArr1, 0, original.length/2);

System.arraycopy(original, original.length/2, subArr2, 0, original.length - original.length/2);

Sorter runner1 = new Sorter(subArr1);

Sorter runner2 = new Sorter(subArr2);

runner1.start();

runner2.start();

runner1.join();

runner2.join();

finalMerge (runner1.getInternal(), runner2.getInternal()); //final merge call

long stopTime = System.currentTimeMillis();

long elapsedTime = stopTime - startTime;

System.out.println("\n"+RED+"MultiThreadSorting using two threads takes: "+RESET+(float)elapsedTime/1000 + " seconds\n\n\n");

System.out.print("Exit? Y/N?: ");

s = new Scanner(System.in);

char yn = s.next().charAt(0);

if(yn=='n'||yn=='N')

{//loop

}

else

{

break;

}

}

} //main

} //Class MultiThreadSorting

class Sorter extends Thread

{

private int[] internal;

Sorter(int[] arr) //constructor acting as set method

{

internal = arr;

}

public int[] getInternal() // nothing but get method

{

return internal;

}

public int[] leftHalf(int[] array)

{

int size1 = array.length / 2;

int[] left = new int[size1];

for (int i = 0; i < size1; i++)

{

left[i] = array[i];

}

return left;

}

public int[] rightHalf(int[] array)

{

int size1 = array.length / 2;

int size2 = array.length - size1;

int[] right = new int[size2];

for (int i = 0; i < size2; i++)

{

right[i] = array[i + size1];

}

return right;

}

public void merge(int[] result, int[] left, int[] right)

{

int i1 = 0;

int i2 = 0;

for (int i = 0; i < result.length; i++)

{

if (i2 >= right.length || (i1 < left.length && left[i1] <= right[i2]))

{

result[i] = left[i1];

i1++;

}

else

{

result[i] = right[i2];

i2++;

}

}

}

public void MultiThreadSort(int[] array) //A recursive function

{

if (array.length > 1)

{

int[] left = leftHalf(array);

int[] right = rightHalf(array);

MultiThreadSort(left); //recursion (0 to n/2)

MultiThreadSort(right); //recursion (n/2+1 to n)

merge(array, left, right);

}

}

public void run()

{

MultiThreadSort(internal); //function(mergersort)

}

}