Import java.util.Arrays;

Public class ParallelBubbleSort {

Public static void main(String[] args) {

Int[] array = {64, 34, 25, 12, 22, 11, 90};

System.out.println(“Original array: “ + Arrays.toString(array));

parallelBubbleSort(array);

System.out.println(“Sorted array: “ + Arrays.toString(array));

}

Public static void parallelBubbleSort(int[] array) {

Int n = array.length;

For (int I = 0; I < n – 1; i++) {

Boolean swapped = false;

For (int j = 0; j < n – I – 1; j++) {

If (array[j] > array[j + 1]) {

// Swap array[j] and array[j+1]

Int temp = array[j];

Array[j] = array[j + 1];

Array[j + 1] = temp;

Swapped = true;

}

}

// If no two elements were swapped in the inner loop, the array is already sorted

If (!swapped) {

Break;

}

}

}

}

Import java.util.Arrays;

Import java.util.concurrent.ForkJoinPool;

Import java.util.concurrent.RecursiveAction;

Public class ParallelMergeSort {

Public static void main(String[] args) {

Int[] array = {64, 34, 25, 12, 22, 11, 90};

System.out.println(“Original array: “ + Arrays.toString(array));

parallelMergeSort(array);

System.out.println(“Sorted array: “ + Arrays.toString(array));

}

Public static void parallelMergeSort(int[] array) {

ForkJoinPool pool = ForkJoinPool.commonPool();

Pool.invoke(new MergeSortTask(array, 0, array.length – 1));

}

Static class MergeSortTask extends RecursiveAction {

Private final int[] array;

Private final int start;

Private final int end;

MergeSortTask(int[] array, int start, int end) {

This.array = array;

This.start = start;

This.end = end;

}

@Override

Protected void compute() {

If (start < end) {

Int mid = (start + end) / 2;

MergeSortTask leftTask = new MergeSortTask(array, start, mid);

MergeSortTask rightTask = new MergeSortTask(array, mid + 1, end);

invokeAll(leftTask, rightTask);

merge(array, start, mid, end);

}

}

Private void merge(int[] array, int start, int mid, int end) {

Int n1 = mid – start + 1;

Int n2 = end – mid;

Int[] leftArray = new int[n1];

Int[] rightArray = new int[n2];

For (int I = 0; I < n1; ++i) {

leftArray[i] = array[start + i];

}

For (int j = 0; j < n2; ++j) {

rightArray[j] = array[mid + 1 + j];

}

Int I = 0, j = 0;

Int k = start;

While (I < n1 && j < n2) {

If (leftArray[i] <= rightArray[j]) {

Array[k] = leftArray[i];

I++;

} else {

Array[k] = rightArray[j];

J++;

}

K++;

}

While (I < n1) {

Array[k] = leftArray[i];

I++;

K++;

}

While (j < n2) {

Array[k] = rightArray[j];

J++;

K++;

}

}

}

}