HPC Lab Exp No.2

import java.util.\*;

import java.util.concurrent.\*;

import java.util.Scanner;

public class Main {

public static void bubbleSort(int[] arr) {

int n = arr.length;

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (arr[j] > arr[j + 1]) {

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

public static void parallelBubbleSort(int[] arr) {

int n = arr.length;

ExecutorService executor = Executors.newCachedThreadPool();

for (int i = 0; i < n - 1; i++) {

final int currentI = i;

List<Future<?>> futures = new ArrayList<>();

for (int j = 0; j < n - currentI - 1; j++) {

final int currentJ = j;

futures.add(executor.submit(() -> {

if (arr[currentJ] > arr[currentJ + 1]) {

int temp = arr[currentJ];

arr[currentJ] = arr[currentJ + 1];

arr[currentJ + 1] = temp;

}

}));

}

for (Future<?> future : futures) {

try {

future.get();

} catch (InterruptedException | ExecutionException e) {

e.printStackTrace();

}

}

}

executor.shutdown();

}

public static void mergeSort(int[] arr, int left, int right) {

if (left < right) {

int mid = (left + right) / 2;

mergeSort(arr, left, mid);

mergeSort(arr, mid + 1, right);

merge(arr, left, mid, right);

}

}

public static void merge(int[] arr, int left, int mid, int right) {

int n1 = mid - left + 1;

int n2 = right - mid;

int[] leftArray = new int[n1];

int[] rightArray = new int[n2];

System.arraycopy(arr, left, leftArray, 0, n1);

System.arraycopy(arr, mid + 1, rightArray, 0, n2);

int i = 0, j = 0, k = left;

while (i < n1 && j < n2) {

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

arr[k] = leftArray[i];

i++;

} else {

arr[k] = rightArray[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = leftArray[i];

i++;

k++;

}

while (j < n2) {

arr[k] = rightArray[j];

j++;

k++;

}

}

public static void parallelMergeSort(int[] arr, int left, int right) {

ForkJoinPool forkJoinPool = new ForkJoinPool();

forkJoinPool.submit(() -> mergeSortRecursive(arr, left, right));

forkJoinPool.shutdown();

}

private static void mergeSortRecursive(int[] arr, int left, int right) {

if (left < right) {

int mid = (left + right) / 2;

mergeSortRecursive(arr, left, mid);

mergeSortRecursive(arr, mid + 1, right);

merge(arr, left, mid, right);

}

}

public static long measureTime(Runnable sortMethod) {

long startTime = System.currentTimeMillis();

sortMethod.run();

long endTime = System.currentTimeMillis();

return endTime - startTime;

}

public static void printArray(int[] arr) {

for (int i : arr) {

System.out.print(i + " ");

}

System.out.println();

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

boolean continueRunning = true;

while (continueRunning) {

System.out.println("\nChoose an option:");

System.out.println("1. Sequential Bubble Sort");

System.out.println("2. Parallel Bubble Sort");

System.out.println("3. Sequential Merge Sort");

System.out.println("4. Parallel Merge Sort");

System.out.println("5. Exit");

System.out.print("Enter your choice: ");

int choice = scanner.nextInt();

System.out.print("Enter the number of elements: ");

int size = scanner.nextInt();

int[] arr = new int[size];

System.out.print("Enter the " + size + " space-separated values of the array: ");

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

arr[i] = scanner.nextInt();

}

switch (choice) {

case 1:

// Sequential Bubble Sort

System.out.println("\nPerforming Sequential Bubble Sort...");

int[] arrBubbleSeq = arr.clone();

long bubbleSeqTime = measureTime(() -> bubbleSort(arrBubbleSeq));

System.out.println("Sorted Array: ");

printArray(arrBubbleSeq);

System.out.println("Time taken: " + bubbleSeqTime + " ms");

break;

case 2:

// Parallel Bubble Sort

System.out.println("\nPerforming Parallel Bubble Sort...");

int[] arrBubblePar = arr.clone();

long bubbleParTime = measureTime(() -> parallelBubbleSort(arrBubblePar));

System.out.println("Sorted Array: ");

printArray(arrBubblePar);

System.out.println("Time taken: " + bubbleParTime + " ms");

break;

case 3:

// Sequential Merge Sort

System.out.println("\nPerforming Sequential Merge Sort...");

int[] arrMergeSeq = arr.clone();

long mergeSeqTime = measureTime(() -> mergeSort(arrMergeSeq, 0, arrMergeSeq.length - 1));

System.out.println("Sorted Array: ");

printArray(arrMergeSeq);

System.out.println("Time taken: " + mergeSeqTime + " ms");

break;

case 4:

// Parallel Merge Sort

System.out.println("\nPerforming Parallel Merge Sort...");

int[] arrMergePar = arr.clone();

long mergeParTime = measureTime(() -> parallelMergeSort(arrMergePar, 0, arrMergePar.length - 1));

System.out.println("Sorted Array: ");

printArray(arrMergePar);

System.out.println("Time taken: " + mergeParTime + " ms");

break;

case 5:

continueRunning = false;

System.out.println("Exited the program successfully");

break;

default:

System.out.println("Invalid choice! Please choose a valid option.");

break;

}

if (continueRunning) {

System.out.print("\nDo you want to continue? (yes/no): ");

String userChoice = scanner.next();

if (userChoice.equalsIgnoreCase("no")) {

continueRunning = false;

System.out.println("Exited the program successfully");

}

}

}

scanner.close();

}

}

OUTPUT:





