#include <iostream>

#include <omp.h>

using namespace std;

void merge(int arr[], int low, int mid, int high) {

// Create arrays of left and right partititons

int n1 = mid - low + 1;

int n2 = high - mid;

int left[n1];

int right[n2];

// Copy all left elements

for (int i = 0; i < n1; i++) left[i] = arr[low + i];

// Copy all right elements

for (int j = 0; j < n2; j++) right[j] = arr[mid + 1 + j];

// Compare and place elements

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

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

if (left[i] <= right[j]){

arr[k] = left[i];

i++;

}

else{

arr[k] = right[j];

j++;

}

k++;

}

// If any elements are left out

while (i < n1) {

arr[k] = left[i];

i++;

k++;

}

while (j < n2) {

arr[k] = right[j];

j++;

k++;

}

}

void parallelMergeSort(int arr[], int low, int high) {

if (low < high) {

int mid = (low + high) / 2;

#pragma omp parallel sections

{

#pragma omp section

{

parallelMergeSort(arr, low, mid);

}

#pragma omp section

{

parallelMergeSort(arr, mid + 1, high);

}

}

merge(arr, low, mid, high);

}

}

void mergeSort(int arr[], int low, int high) {

if (low < high) {

int mid = (low + high) / 2;

mergeSort(arr, low, mid);

mergeSort(arr, mid + 1, high);

merge(arr, low, mid, high);

}

}

int main() {

int n = 10;

int arr[n];

double start\_time, end\_time;

// Create an array with numbers starting from n to 1.

for(int i = 0, j = n; i < n; i++, j--) arr[i] = j;

// Measure Sequential Time

start\_time = omp\_get\_wtime();

mergeSort(arr, 0, n - 1);

end\_time = omp\_get\_wtime();

cout << "Time taken by sequential algorithm: " << end\_time - start\_time << " seconds\n";

// Reset the array

for(int i = 0, j = n; i < n; i++, j--) arr[i] = j;

//Measure Parallel time

start\_time = omp\_get\_wtime();

parallelMergeSort(arr, 0, n - 1);

end\_time = omp\_get\_wtime();

cout << "Time taken by parallel algorithm: " << end\_time - start\_time << " seconds";

return 0;

}