**Merge sort**

#include <iostream>

#include <vector>

#include <omp.h>

#include <chrono>

using namespace std;

bool isSorted(const vector<int>& arr) {

int n = arr.size();

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

if (arr[i] < arr[i - 1]) {

return false;

}

}

return true;

}

void merge(vector<int>& arr, int low, int mid, int high) {

int i = low;

int j = mid + 1;

vector<int> merged(high - low + 1);

for (int k = 0; k < merged.size(); k++) {

if (i > mid) {

merged[k] = arr[j++];

} else if (j > high) {

merged[k] = arr[i++];

} else if (arr[i] <= arr[j]) {

merged[k] = arr[i++];

} else {

merged[k] = arr[j++];

}

}

for (int k = 0; k < merged.size(); k++) {

arr[low + k] = merged[k];

}

}

void mergeSort(vector<int>& arr, int low, int high) {

if (low < high) {

int mid = (low + high) / 2;

// Recursively sort the left and right halves.

mergeSort(arr, low, mid);

mergeSort(arr, mid + 1, high);

// Merge the sorted halves.

#pragma omp parallel

{

#pragma omp for

for (int i = low; i <= high; i++) {

// Do nothing, just for parallelization.

}

merge(arr, low, mid, high);

}

}

}

int main() {

int n;

cout << "Enter the number of elements: ";

cin >> n;

vector<int> arr(n);

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

cout << "Enter element " << i + 1 << ": ";

cin >> arr[i];

}

cout << "Unsorted array: ";

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

cout << arr[i] << " ";

}

cout << endl;

// Merge sort.

cout << "Sequential merge sort: ";

auto start = std::chrono::high\_resolution\_clock::now();

mergeSort(arr, 0, n - 1);

auto end = std::chrono::high\_resolution\_clock::now();

std::chrono::duration<double> elapsed = end - start;

cout << elapsed.count() << " seconds" << endl;

// Parallel merge sort.

cout << "Parallel merge sort: ";

start = std::chrono::high\_resolution\_clock::now();

#pragma omp parallel

{

#pragma omp single

mergeSort(arr, 0, n - 1);

}

end = std::chrono::high\_resolution\_clock::now();

elapsed = end - start;

cout << elapsed.count() << " seconds" << endl;

cout << "Sorted array: ";

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

cout << arr[i] << " ";

}

cout << endl;

// Check if the array is sorted.

if (isSorted(arr)) {

cout << "The array is sorted." << endl;

} else {

cout << "The array is not sorted." << endl;

}

return 0;

}

**Bubble sort**

#include <iostream>

#include <vector>

#include <omp.h>

#include <chrono>

using namespace std;

void bubbleSort(vector<int>& arr) {

int n = arr.size();

#pragma omp parallel for

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;

}

}

}

}

int main() {

int n;

cout << "Enter the number of elements: ";

cin >> n;

vector<int> arr(n);

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

cout << "Enter element " << i + 1 << ": ";

cin >> arr[i];

}

cout << "Unsorted array: ";

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

cout << arr[i] << " ";

}

cout << endl;

// Bubble sort.

cout << "Sequential bubble sort: ";

auto start = std::chrono::high\_resolution\_clock::now();

bubbleSort(arr);

auto end = std::chrono::high\_resolution\_clock::now();

std::chrono::duration<double> elapsed = end - start;

cout << elapsed.count() << " seconds" << endl;

cout << "Sorted array: ";

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

cout << arr[i] << " ";

}

cout << endl;

// Parallel bubble sort.

cout << "Parallel bubble sort: ";

start = std::chrono::high\_resolution\_clock::now();

#pragma omp parallel

bubbleSort(arr);

end = std::chrono::high\_resolution\_clock::now();

elapsed = end - start;

cout << elapsed.count() << " seconds" << endl;

return 0;

}