#include <iostream>

#include <omp.h>

#include <vector>

#include <cstdlib>

#include <ctime>

#include <algorithm>

using namespace std;

// Sequential Bubble Sort

void bubbleSortSequential(vector<int>& arr) {

int n = arr.size();

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

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

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

swap(arr[j], arr[j + 1]);

}

// Parallel Bubble Sort

void bubbleSortParallel(vector<int>& arr) {

int n = arr.size();

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

#pragma omp parallel for

for (int j = i % 2; j < n - 1; j += 2) {

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

swap(arr[j], arr[j + 1]);

}

}

}

}

// Merge utility

void merge(vector<int>& arr, int left, int mid, int right) {

vector<int> temp(right - left + 1);

int i = left, j = mid + 1, k = 0;

while(i <= mid && j <= right) {

if(arr[i] <= arr[j]) temp[k++] = arr[i++];

else temp[k++] = arr[j++];

}

while(i <= mid) temp[k++] = arr[i++];

while(j <= right) temp[k++] = arr[j++];

for(i = left; i <= right; ++i) arr[i] = temp[i - left];

}

// Sequential Merge Sort

void mergeSortSequential(vector<int>& arr, int left, int right) {

if(left < right) {

int mid = (left + right) / 2;

mergeSortSequential(arr, left, mid);

mergeSortSequential(arr, mid + 1, right);

merge(arr, left, mid, right);

}

}

// Parallel Merge Sort with OpenMP

void mergeSortParallel(vector<int>& arr, int left, int right, int depth = 0) {

if(left < right) {

int mid = (left + right) / 2;

if (depth < 4) {

#pragma omp parallel sections

{

#pragma omp section

mergeSortParallel(arr, left, mid, depth + 1);

#pragma omp section

mergeSortParallel(arr, mid + 1, right, depth + 1);

}

} else {

mergeSortSequential(arr, left, mid);

mergeSortSequential(arr, mid + 1, right);

}

merge(arr, left, mid, right);

}

}

// Utility to print array

void printArray(const vector<int>& arr) {

for (int val : arr) cout << val << " ";

cout << "\n";

}

int main() {

int size;

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

cin >> size;

vector<int> original(size);

cout << "Enter " << size << " integers:\n";

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

cin >> original[i];

// Sequential Bubble Sort

vector<int> arr1 = original;

double start = omp\_get\_wtime();

bubbleSortSequential(arr1);

double end = omp\_get\_wtime();

cout << "\nSorted (Sequential Bubble Sort): ";

printArray(arr1);

cout << "Time: " << end - start << " seconds\n";

// Parallel Bubble Sort

vector<int> arr2 = original;

start = omp\_get\_wtime();

bubbleSortParallel(arr2);

end = omp\_get\_wtime();

cout << "\nSorted (Parallel Bubble Sort): ";

printArray(arr2);

cout << "Time: " << end - start << " seconds\n";

// Sequential Merge Sort

vector<int> arr3 = original;

start = omp\_get\_wtime();

mergeSortSequential(arr3, 0, arr3.size() - 1);

end = omp\_get\_wtime();

cout << "\nSorted (Sequential Merge Sort): ";

printArray(arr3);

cout << "Time: " << end - start << " seconds\n";

// Parallel Merge Sort

vector<int> arr4 = original;

start = omp\_get\_wtime();

mergeSortParallel(arr4, 0, arr4.size() - 1);

end = omp\_get\_wtime();

cout << "\nSorted (Parallel Merge Sort): ";

printArray(arr4);

cout << "Time: " << end - start << " seconds\n";

return 0;

}

**Output**

abc@abc-Latitude-5480:~$ g++ /home/abc/sort.cpp -fopenmp

abc@abc-Latitude-5480:~$ ./a.out

Enter the number of elements: 6

Enter 6 integers:

1 2 3 5 6 9

Sorted (Sequential Bubble Sort): 1 2 3 5 6 9

Time: 2.634e-06 seconds

Sorted (Parallel Bubble Sort): 1 2 3 5 6 9

Time: 0.000685994 seconds

Sorted (Sequential Merge Sort): 1 2 3 5 6 9

Time: 1.8928e-05 seconds

Sorted (Parallel Merge Sort): 1 2 3 5 6 9

Time: 0.000331795 seconds