Assignment 2a: Bubble sort

#include <iostream>

#include <vector>

#include <omp.h>

using namespace std;

void bubble\_sort\_odd\_even(vector<int>& arr) {

bool isSorted = false;

while (!isSorted) {

isSorted = true;

#pragma omp parallel for

for (int i = 0; i < arr.size() - 1; i += 2) {

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

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

isSorted = false;

}

}

#pragma omp parallel for

for (int i = 1; i < arr.size()- 1; i += 2) {

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

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

isSorted = false;

}

}

}

}

int main() {

vector<int> arr = {5, 2, 9, 1, 7, 6, 8, 3, 4};

double start, end;

// Measure performance of parallel bubble sort using odd-

even transposition

start = omp\_get\_wtime();

bubble\_sort\_odd\_even(arr);

end = omp\_get\_wtime();

cout << "Parallel bubble sort using odd-even transposition

time: " << end - start << endl;

}

**Assignment 2b: Merge sort**

#include <iostream>

#include <vector>

#include <omp.h>

using namespace std;

void merge(vector<int>& arr, int l, int m, int r) {

int i, j, k;

int n1 = m- l + 1;

int n2 = r - m;

vector<int> L(n1), R(n2);

for (i = 0;i < n1; i++) {

L[i] = arr[l +i]

}

for (j = 0; j < n2; j++) {

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

}

i = 0;

j = 0;

k = l;

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

if (L[i] <= R[j]) {

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

} else {

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

}

}

void merge\_sort(vector<int>& arr, int l, int r) {

if (l < r) {

int m = l + (r- l) / 2;

#pragma omp task

merge\_sort (arr, l, m);

#pragma omp task

merge\_sort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

void parallel\_merge\_sort(vector<int>& arr) {

#pragma omp parallel {

#pragma omp single

merge\_sort(arr, 0, arr.size()- 1);

}

}

int main() {

vector<int> arr = {5, 2, 9, 1, 7, 6, 8, 3, 4};

double start, end;

// Measure performance of sequential merge sort

start = omp\_get\_wtime();

merge\_sort(arr, 0, arr.size()- 1);

end = omp\_get\_wtime();

cout << "Sequential merge sort time: " << end- start <<

endl;

// Measure performance of parallel merge sort

arr = {5, 2, 9, 1, 7, 6, 8, 3, 4};

start = omp\_get\_wtime();

parallel\_merge\_sort(arr);

end = omp\_get