Name :- Umakant Dodtalle
Roll N.o :- 4127
Title: Write a program to implement Parallel Bubble Sort and Merge sort using OpenMP. Use existing algorithms and measure the performance of sequential and parallel algorithms.

```
#include <iostream>
#include <vector>
#include <omp.h>
#include <chrono>
#include <algorithm>

using namespace std;
using namespace std::chrono;

// Sequential Bubble Sort
void sequentialBubbleSort(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 parallelBubbleSort(vector<int>& arr) {
    int n = arr.size();
    bool swapped;
#pragma omp parallel
    {
        do {
            swapped = false;
#pragma omp for
            for (int i = 0; i < n - 1; ++i) {
                if (arr[i] > arr[i + 1]) {
                    swap(arr[i], arr[i + 1]);
                    swapped = true;
                }
            }
        } while (swapped);
    }
}

// Sequential Merge Sort Helper Function
void merge(vector<int>& arr, int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;
    vector<int> L(n1), R(n2);
    for (int i = 0; i < n1; ++i) {
        L[i] = arr[left + i];
    }
    for (int j = 0; j < n2; ++j) {
        R[j] = arr[mid + 1 + j];
    }
    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k++] = L[i++];
        } else {
            arr[k++] = R[j++];
        }
    }
    while (i < n1) {
        arr[k++] = L[i++];
    }
    while (j < n2) {
        arr[k++] = R[j++];
    }
}

// Sequential Merge Sort
void sequentialMergeSort(vector<int>& arr, int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        sequentialMergeSort(arr, left, mid);
        sequentialMergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}

// Parallel Merge Sort
void parallelMergeSort(vector<int>& arr, int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;
#pragma omp parallel sections
        {
#pragma omp section
            parallelMergeSort(arr, left, mid);
#pragma omp section
            parallelMergeSort(arr, mid + 1, right);
        }
        merge(arr, left, mid, right);
    }
}

int main() {
    int n = 10000; // Change the array size as needed
    vector<int> arr(n);
    // Initialize array with random values
    srand(time(0));
    for (int i = 0; i < n; ++i) {
        arr[i] = rand() % 10000;
    }
    // Measure time for Sequential Bubble Sort
    auto start = high_resolution_clock::now();
    sequentialBubbleSort(arr);
    auto stop = high_resolution_clock::now();
    auto durationSequentialBubbleSort = duration_cast<milliseconds>(stop - start);
    // Measure time for Parallel Bubble Sort
    start = high_resolution_clock::now();
    parallelBubbleSort(arr);
    stop = high_resolution_clock::now();
    auto durationParallelBubbleSort = duration_cast<milliseconds>(stop - start);
    // Reset array for merge sort
    for (int i = 0; i < n; ++i) {
        arr[i] = rand() % 10000;
    }
    // Measure time for Sequential Merge Sort
    start = high_resolution_clock::now();
    sequentialMergeSort(arr, 0, n - 1);
    stop = high_resolution_clock::now();
    auto durationSequentialMergeSort = duration_cast<milliseconds>(stop - start);
    // Measure time for Parallel Merge Sort
    start = high_resolution_clock::now();
    parallelMergeSort(arr, 0, n - 1);
    stop = high_resolution_clock::now();
    auto durationParallelMergeSort = duration_cast<milliseconds>(stop - start);
    // Display execution times
    cout << "Sequential Bubble Sort Time: " << durationSequentialBubbleSort.count() << " milliseconds"
<< endl;
    cout << "Parallel Bubble Sort Time: " << durationParallelBubbleSort.count() << " milliseconds" << endl;
    cout << "Sequential Merge Sort Time: " << durationSequentialMergeSort.count() << " milliseconds" <<
endl;
    cout << "Parallel Merge Sort Time: " << durationParallelMergeSort.count() << " milliseconds" << endl;
    return 0;
}
```

Output :-

```
PS E:\HPC> cd "e:\HPC\" ; if ($?) { g++ HPC2.cpp -o HPC2 } ; if ($?) { .\HPC2 }
Sequential Bubble Sort Time: 1621 milliseconds
Parallel Bubble Sort Time: 0 milliseconds
Sequential Merge Sort Time: 13 milliseconds
Parallel Merge Sort Time: 13 milliseconds
```