Name : Akshata Jadhav

BE-A-25

Practical No 2: 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.

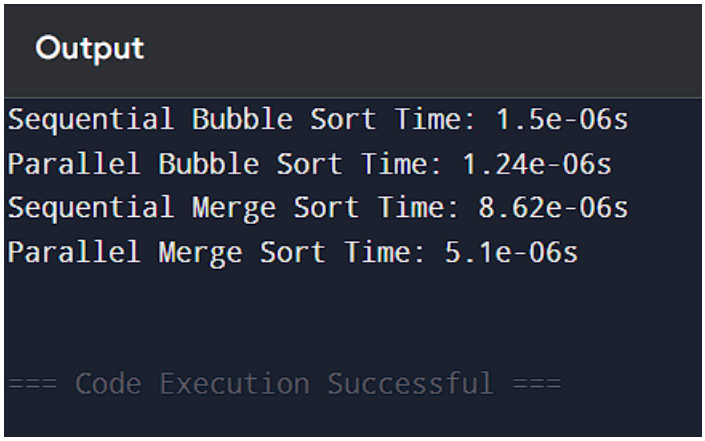
Code:

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

using namespace std;
using namespace std::chrono;

void bubbleSort(vector<int>& arr, bool parallel) {
    int n = arr.size();
    bool swapped;
    do {
        swapped = false;
        if (parallel) {
            #pragma omp parallel for
            for (int i = 0; i < n - 1; i++) {
                if (arr[i] > arr[i + 1]) {
                    swap(arr[i], arr[i + 1]);
                    #pragma omp critical
                    swapped = true;
                }
            }
        } else {
            for (int i = 0; i < n - 1; i++) {
                if (arr[i] > arr[i + 1]) {
```

```
                swap(arr[i], arr[i + 1]);
                swapped = true;
            }
        }
    }
  } while (swapped);
}

void merge(vector<int>& arr, int l, int m, int r) {
  int n1 = m - l + 1, n2 = r - m;
  vector<int> left(arr.begin() + l, arr.begin() + m + 1);
  vector<int> right(arr.begin() + m + 1, arr.begin() + r + 1);
  int i = 0, j = 0, k = l;
  while (i < n1 && j < n2) arr[k++] = (left[i] <= right[j]) ? left[i++] : right[j++];
  while (i < n1) arr[k++] = left[i++];
  while (j < n2) arr[k++] = right[j++];
}

void mergeSort(vector<int>& arr, int l, int r, bool parallel) {
  if (l >= r) return;
  int m = l + (r - l) / 2;
  if (parallel) {
    #pragma omp parallel sections
    {
      #pragma omp section
      mergeSort(arr, l, m, parallel);
      #pragma omp section
      mergeSort(arr, m + 1, r, parallel);
    }
  } else {
    mergeSort(arr, l, m, parallel);
```

```
        mergeSort(arr, m + 1, r, parallel);
    }
    merge(arr, l, m, r);
}

int main() {
    vector<int> arr = {5, 3, 8, 4, 2, 7, 1, 6};
    vector<int> arr1 = arr, arr2 = arr, arr3 = arr, arr4 = arr;

    auto start = high_resolution_clock::now();
    bubbleSort(arr1, false);
    auto end = high_resolution_clock::now();
    cout << "Sequential Bubble Sort Time: " << duration<double>(end - start).count() << "s\n";

    start = high_resolution_clock::now();
    bubbleSort(arr2, true);
    end = high_resolution_clock::now();
    cout << "Parallel Bubble Sort Time: " << duration<double>(end - start).count() << "s\n";

    start = high_resolution_clock::now();
    mergeSort(arr3, 0, arr3.size() - 1, false);
    end = high_resolution_clock::now();
    cout << "Sequential Merge Sort Time: " << duration<double>(end - start).count() << "s\n";

    start = high_resolution_clock::now();
    mergeSort(arr4, 0, arr4.size() - 1, true);
    end = high_resolution_clock::now();
    cout << "Parallel Merge Sort Time: " << duration<double>(end - start).count() << "s\n";

    return 0;
}
```

Output:


Output
Sequential Bubble Sort Time: 1.5e-06s
Parallel Bubble Sort Time: 1.24e-06s
Sequential Merge Sort Time: 8.62e-06s
Parallel Merge Sort Time: 5.1e-06s
=== Code Execution Successful ===