Name: Mayur Goraksha Gaikwad

Roll No.: 19121028

Subject: High Performance Computing



**Experiment No.:2**

**Title**: Implement Parallel Merge Sort using OpenMP.

**Code:**  
#include <iostream>

#include <omp.h>

using namespace std;

void merge(int \*arr, int l, int m, int r) {

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

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];

i++;

} else {

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

void sequential\_merge\_sort(int \*arr, int l, int r) {

if (l >= r) return;

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

sequential\_merge\_sort(arr, l, m);

sequential\_merge\_sort(arr, m + 1, r);

merge(arr, l, m, r);

}

void parallel\_merge\_sort(int \*arr, int l, int r) {

if (l >= r) return;

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

#pragma omp parallel num\_threads(2)

{

#pragma omp sections

{

#pragma omp section

parallel\_merge\_sort(arr, l, m);

#pragma omp section

parallel\_merge\_sort(arr, m + 1, r);

}

}

merge(arr, l, m, r);

}

int main() {

int arr[] = {5, 2, 9, 1, 5, 6};

int n = sizeof(arr) / sizeof(arr[0]);

cout << "Before sorting: ";

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

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

}

cout << endl;

parallel\_merge\_sort(arr, 0, n - 1);

cout << "After sorting: ";

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

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

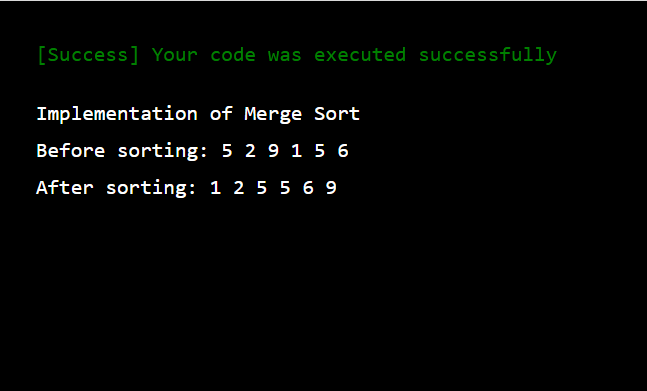
}

cout << endl;

return 0;

}

**Output:**

****