**Bubble sort:**

#include <iostream>

#include <omp.h>

using namespace std;

void parallelBubbleSort(int arr[], int n) {

bool swapped = true;

#pragma omp parallel

{

while (swapped) {

swapped = false;

#pragma omp for

for (int i = 0; i< n - 1; i += 2) {

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

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

swapped = true;

}

}

#pragma omp for

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

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

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

swapped = true;

}

}

}

}

}

int main() {

int arr[] = {5, 3, 8, 6, 7, 2};

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

parallelBubbleSort(arr, n);

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

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

}

cout<<endl;

return 0;

}

Merge sort:

#include <iostream>

#include <omp.h>

using namespace std;

void merge(int arr[], int left, int mid, int right) {

int n1 = mid - left + 1;

int n2 = right - mid;

int L[n1], R[n2];

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

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

}

for (int i = 0; i< n2; i++) {

R[i] = arr[mid + 1 + i];

}

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++];

}

}

void parallelMergeSort(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 arr[] = {5, 3, 8, 6, 7, 2};

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

parallelMergeSort(arr, 0, n - 1);

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

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

}

cout<<endl;

return 0;

}