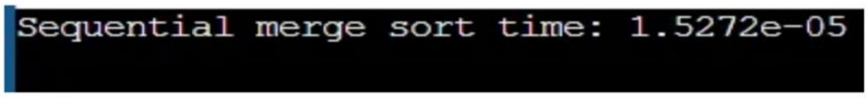
Merge sort Code:

```
#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_wtime();
      Return 0;
      }
```

Output:



```
Sequential merge sort time: 1.5272e-05
```

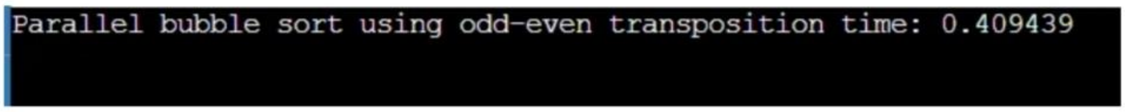
output 1

Bubble Sort Code:

```
#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;
}
```

Output:



```
Parallel bubble sort using odd-even transposition time: 0.409439
```

output 2