```
%%writefile bubble.cpp
#include<omp.h>
#include<iostream>
using namespace std;

void parallel_bubblesort(int n, int*nums){
    //Implementing parallel bubble sort
    int start=0;
    for(int i=0; i<n-1; i++) {
        start = i%2;
        #pragma omp parallel for
        for(int j=start; j<n-1; j++) {
            if(nums[j] > nums[j+1])
              swap(nums[j], nums[j+1]);
        }
    }
}

void sequential_bubblesort(int n, int*nums){
    {
        for (int i = 0; i < n-1; i++){
            for (int j = 0; j < n-i-1; j++){
                if (nums[j] > nums[j+1]) {
                    swap(nums[j], nums[j+1]);
                }
            }
        }
    }
}

int main() {
    int n = 100;
    srand(n);
    int *par_nums=new int[n];
    int *seq_nums = new int[n];
    cout<<"\nUnsorted List:";
    for(int i=0; i<n; i++) {
        seq_nums[i] = rand()% 1000;
        par_nums[i] = seq_nums[i];
        cout<<"  "<<seq_nums[i];
    }

    //SEQUENTIAL BUBBLE SORT
    //Getting start time
      double start_time = omp_get_wtime();
    //Calling sequential sort
      sequential_bubblesort(n,seq_nums);
    //Getting end time
      double end_time = omp_get_wtime();
      double sequential_time = end_time-start_time;

    //PARALLEL BUBBLE SORT
    //Setting up threads
      omp_set_num_threads(2);
```

```
    omp_set_num_threads(2);
  //Getting start time
    start_time = omp_get_wtime();
  //Calling parallel sort
    parallel_bubblesort(n,par_nums);
  //Getting end time
    end_time = omp_get_wtime();
    double parallel_time = end_time-start_time;

  cout<<"\nSorted array (Sequential): "<<endl;
  for(int i=0; i<n; i++){
    cout<<"  "<<seq_nums[i];
  }

  cout<<"\nSorted array (Parallel): "<<endl;
  for(int i=0; i<n; i++){
    cout<<"  "<<par_nums[i];
  }
  cout<<"\nParallel execution time = "<<parallel_time<<endl;
  cout<<"\nSequential execution time = "<<sequential_time<<endl;
  return 0;

}
```

```
Overwriting bubble.cpp
```

```
!g++ -fopenmp bubble.cpp
```

```
!./a.out
```

```
Unsorted List:  240  301  479  884  856  623  905  270  981  371  180  828  597  747
Sorted array (Sequential):
  13  13  18  33  37  46  48  104  116  129  141  148  148  150  151  155  174  177
Sorted array (Parallel):
  13  13  18  33  37  46  48  104  116  129  141  148  148  150  151  155  174  177
Parallel execution time = 0.000239888

Sequential execution time = 5.4437e-05
```

```
%%writefile merge.cpp
#include<omp.h>
#include<iostream>
using namespace std;

void merge(int *arr,int l,int mid,int r)
{
  int n1 = mid-l+1;
  int n2 = r-mid;

  int i,j,k;

  int *a = new int[n1];
  int *b = new int[n2];
```

```
  for(i=0;i<n1;i++){
    a[i] = arr[i+l];
  }

  for(i=0;i<n2;i++){
    b[i] = arr[i+mid+1];
  }

  k = l;
  i = 0;
  j = 0;

  while(i<n1 && j<n2){
    if(a[i]<b[j]){
      arr[k] = a[i];
      k++;
      i++;
    } else{
      arr[k] = b[j];
      k++;
      j++;
    }
  }

  while(i<n1){
    arr[k] = a[i];
    k++;
    i++;
  }

  while(j<n2){
    arr[k] = b[j];
    k++;
    j++;
  }
}

void sequential_mergesort(int *arr,int l, int r)
{
  if(l<r)
  {
    int mid = (l+r)/2;
    sequential_mergesort(arr,l,mid);
    sequential_mergesort(arr,mid+1,r);
    merge(arr,l,mid,r);
  }
}

void parallel_mergesort(int *arr,int l,int r)
{
  if(l<r)
  {
    int mid = (l+r)/2;
    #pragma omp parallel sections num_threads(2)
```

```
    {
      #pragma omp section
      {
        parallel_mergesort(arr,l,mid);
      }
      #pragma omp section
      {
        parallel_mergesort(arr,mid+1,r);
      }
    }
    merge(arr,l,mid,r);
  }
}
int main() {
    int n = 100;
    srand(n);
    int *par_nums=new int[n];
    int *seq_nums = new int[n];
    cout<<"\nUnsorted List:";
    for(int i=0; i<n; i++) {
        seq_nums[i] = rand()% 1000;
        par_nums[i] = seq_nums[i];
        cout<<"  "<<seq_nums[i];
    }

    //SEQUENTIAL MERGE SORT
    //Getting start time
      double start_time = omp_get_wtime();
    //Calling sequential sort
      sequential_mergesort(seq_nums,0,n-1);
    //Getting end time
      double end_time = omp_get_wtime();
      double sequential_time = end_time-start_time;

    //PARALLEL MERGE SORT
    //Setting up threads
      omp_set_num_threads(2);
    //Getting start time
      start_time = omp_get_wtime();
    //Calling parallel sort
      parallel_mergesort(par_nums,0,n-1);
    //Getting end time
      end_time = omp_get_wtime();
      double parallel_time = end_time-start_time;

    cout<<"\nSorted array (Sequential): "<<endl;
    for(int i=0; i<n; i++){
      cout<<"  "<<seq_nums[i];
    }

    cout<<"\nSorted array (Parallel): "<<endl;
    for(int i=0; i<n; i++){
      cout<<"  "<<par_nums[i];
    }
    cout<<"\nParallel execution time = "<<parallel_time<<endl;
```

```
    cout<<"\nSequential execution time = "<<sequential_time<<endl;
    return 0;

}
```

```
Overwriting merge.cpp
```

```
!g++ -fopenmp merge.cpp
```

```
!./a.out
```

```
ed List:  240  301  479  884  856  623  905  270  981  371  180  828  597  747  490
 array (Sequential):
13  18  33  37  46  48  104  116  129  141  148  148  150  151  155  174  177  180  1
 array (Parallel):
13  18  33  37  46  48  104  116  129  141  148  148  150  151  155  174  177  180  1
el execution time = 0.000418623

cial execution time = 4.0197e-05
```