Write a program to implement **Parallel Merge sort** using OpenMP. Use existing algorithms and measure the performance of sequential and parallel algorithms

#include<iostream>

#include<cstdlib>

#include<omp.h>

#include<time.h>

using namespace std;

void merge(int array[],int low1, int high1,int low2,int high2, int n)

{

int temp[n];

int i=low1,j=low2,k=0;

while(i<=high1 && j<=high2)

{

if(array[i]<array[j])

temp[k++]=array[i++];

else

temp[k++]=array[j++];

}

while(i<=high1)

temp[k++]=array[i++];

while(j<=high2)

temp[k++]=array[j++];

for(i=low1,j=0;i<=high2;i++,j++)

array[i]=temp[j];

}

void mergesort(int array[], int low, int high, int n)

{

if(low<high)

{

int mid=(low+high)/2;

#pragma omp parallel sections

{

#pragma omp section

{

mergesort(array,low,mid,n);

}

#pragma omp section

{

mergesort(array,mid+1,high,n);

}

}

merge(array,low,mid,mid+1,high,n);

// mergesort(array,low,mid,n);

// mergesort(array,mid+1,high,n);

// merge(array,low,mid,mid+1,high,n);

}

}

void display(int array[], int n)

{

for(int i=0;i<n;i++) cout<<array[i]<<" ";

}

int main()

{

int n;

cout<<"Enter the number of elements : ";

cin>>n;

int array[n] = {0};

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

{

array[i]=rand()%32;

}

cout<<"Original Array: ";

display(array,n);

cout<<endl;

clock\_t start = clock();

mergesort(array,0,n-1,n);

clock\_t stop = clock();

cout<<"Final Array: ";

display(array,n);

cout<<endl;

cout<<"Time required : "<<(double)(stop-start)\*1000/CLOCKS\_PER\_SEC<<" ms";

return 0;

}

/\*

PS D:\C++> g++ -fopenmp parallel\_merge.cpp

PS D:\C++> ./a out

Enter the number of elements : 10

Original Array: 9 3 30 4 1 12 22 14 18 16

Final Array: 1 3 4 9 12 14 16 18 22 30

Time required : 0 ms

\*/