Practical 2 : Merge Sort

```
#include<iostream>
#include<stdlib.h>
#include<omp.h>
#include<cstdlib>
using namespace std;

const int threshold = 1000; // Set your desired threshold here

void mergesort_parallel(int a[],int i,int j);
void merge_parallel(int a[], int i1, int j1, int i2, int j2);
void merge_seq(int a[],int i1,int j1,int i2,int j2);
void mergesort_seq(int a[],int i,int j);

void mergesort_parallel(int a[], int i, int j)
{
   int mid;
   if (i < j)
   {
      mid = (i + j) / 2;

      if ((j - i + 1) <= threshold)
      {
         mergesort_seq(a, i, mid);
         mergesort_seq(a, mid + 1, j);
      }
      else
      {
         #pragma omp parallel sections
         {
            #pragma omp section
            {
               mergesort_parallel(a, i, mid);
            }

            #pragma omp section
            {
               mergesort_parallel(a, mid + 1, j);
            }
         }
      }

      merge_parallel(a, i, mid, mid + 1, j);
   }
}

void merge_parallel(int a[], int i1, int j1, int i2, int j2)
{
   int size = (j1 - i1 + 1) + (j2 - i2 + 1);
   int* temp = new int[size];
   int i=i1;
   int j=i2;
   int k=0;

   while(i<=j1 && j<=j2)
```

```
    {
        if(a[i]<a[j])
        {
            temp[k++]=a[i++];
        }
        else
        {
            temp[k++]=a[j++];
        }
    }

    while(i<=j1)
    {
        temp[k++]=a[i++];
    }

    while(j<=j2)
    {
        temp[k++]=a[j++];
    }

    for(i=i1,j=0;i<=j2;i++,j++)
    {
        a[i]=temp[j];
    }

    delete[] temp; // Free the dynamically allocated memory
}


void mergesort_seq(int a[], int i, int j)
{
    int mid;
    if (i < j)
    {
        mid = (i + j) / 2;

        mergesort_seq(a, i, mid);
        mergesort_seq(a, mid + 1, j);

        merge_seq(a, i, mid, mid + 1, j);
    }
}


void merge_seq(int a[],int i1,int j1,int i2,int j2)
{
    int size = (j1 - i1 + 1) + (j2 - i2 + 1);
    int* temp = new int[size];
    int i=i1;
    int j=i2;
    int k=0;

    while(i<=j1 && j<=j2)
    {
        if(a[i]<a[j])
        {
```

```
            temp[k++]=a[i++];
        }
        else
        {
            temp[k++]=a[j++];
        }
    }

    while(i<=j1)
    {
        temp[k++]=a[i++];
    }

    while(j<=j2)
    {
        temp[k++]=a[j++];
    }

    for(i=i1,j=0;i<=j2;i++,j++)
    {
        a[i]=temp[j];
    }

    delete[] temp; // Free the dynamically allocated memory
}

int main()
{
    cout<<"This is Atharva Pingale's Code";
    cout<<"\nPractical 2 : Merge Sort\n";
    int *a,*b;
    long int n,i;
    double start_time, end_time, seq_time, parallel_time;

    cout<<"\nEnter total no of elements=>";
    cin>>n;
    a = new int[n];
    b = new int[n];

    for(i=0;i<n;i++)
    {
        int random_value = (rand()%(9999999 - 9999 + 1)) + 9999;
        a[i] = random_value;
        b[i] = random_value;
    }

    // Sequential algorithm
    start_time = omp_get_wtime();
    mergesort_seq(a, 0, n-1);
    end_time = omp_get_wtime();
    seq_time = end_time - start_time;

    cout<<"\nFirst 20 elements of sorted array ( Sequential algorithm ) : \n";
    for(i=0;i<20;i++)
    {
        cout<<a[i]<<", ";
    }
```

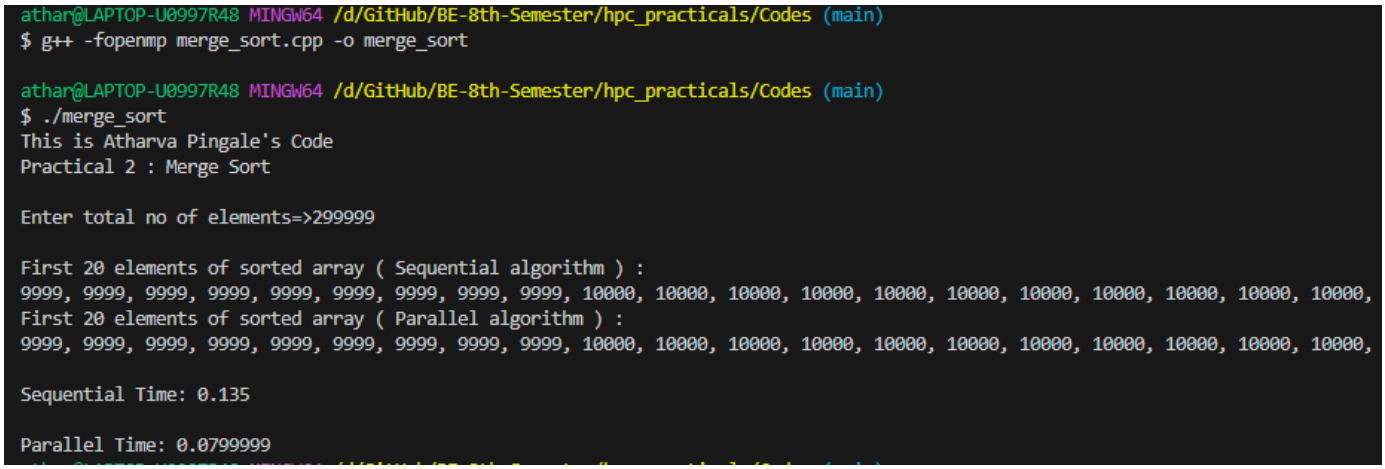
```
    // Parallel algorithm
    start_time = omp_get_wtime();
    mergesort_parallel(b, 0, n-1);
    end_time = omp_get_wtime();
    parallel_time = end_time - start_time;

    cout<<"\nFirst 20 elements of sorted array ( Parallel algorithm ) : \n";
    for(i=0;i<20;i++)
    {
       cout<<b[i]<<", ";
    }
    cout << "\n\nSequential Time: " << seq_time;
    cout << "\n\nParallel Time: " << parallel_time;

    delete[] a;
    delete[] b;

    return 0;
}
```

Output :



Total number of elements : 299999
Time taken by Sequential Algo : 0.135 seconds
Time taken by Parallel Algo : 0.0799999

```
athar@LAPTOP-U0997R48 MINGW64 /d/GitHub/BE-8th-Semester/hpc_practicals/Codes (main)
$ g++ -fopenmp merge_sort.cpp -o merge_sort

athar@LAPTOP-U0997R48 MINGW64 /d/GitHub/BE-8th-Semester/hpc_practicals/Codes (main)
$ ./merge_sort
This is Atharva Pingale's Code
Practical 2 : Merge Sort

Enter total no of elements=>599999

First 20 elements of sorted array ( Sequential algorithm ) :
9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999,
First 20 elements of sorted array ( Parallel algorithm ) :
9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999,

Sequential Time: 0.228

Parallel Time: 0.104
```

Total number of elements : 599999
Time taken by Sequential Algo : 0.228 seconds
Time taken by Parallel Algo : 0.104 seconds

```
athar@LAPTOP-U0997R48 MINGW64 /d/GitHub/BE-8th-Semester/hpc_practicals/Codes (main)
$ g++ -fopenmp merge_sort.cpp -o merge_sort

athar@LAPTOP-U0997R48 MINGW64 /d/GitHub/BE-8th-Semester/hpc_practicals/Codes (main)
$ ./merge_sort
This is Atharva Pingale's Code
Practical 2 : Merge Sort

Enter total no of elements=>1099999

First 20 elements of sorted array ( Sequential algorithm ) :
9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999,
First 20 elements of sorted array ( Parallel algorithm ) :
9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999, 9999,

Sequential Time: 0.371

Parallel Time: 0.182
```

Total number of elements : 1099999
Time taken by Sequential Algo : 0.371 seconds
Time taken by Parallel Algo : 0.182 seconds