

NAME:	RAMNA IRFAN
ROLL NO:	22-NTU-CS-1217
SECTION:	BSCS_6B
ASSIGNMENT:	PDC 1

SEQUENTIAL CODE

CODE:

```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>


#define SIZE 10000000 // Large dataset (10 million elements)

#define RUNS 10      // Number of times to run for average


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

    int i, j, k;

    int n1 = mid - left + 1;

    int n2 = right - mid;

    int* L = (int*)malloc(n1 * sizeof(int));

    int* R = (int*)malloc(n2 * sizeof(int));

    for (i = 0; i < n1; i++) L[i] = arr[left + i];

    for (j = 0; j < n2; j++) R[j] = arr[mid + 1 + j];

    i = 0, j = 0, k = left;

    while (i < n1 && j < n2) {

        arr[k++] = (L[i] <= R[j]) ? L[i++] : R[j++];

    }

    while (i < n1) arr[k++] = L[i++];

    while (j < n2) arr[k++] = R[j++];

}
```

```

    free(L);
    free(R);
}

void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}

int main() {
    double total_time = 0.0;

    srand(time(NULL)); // Seed random number generator

    for (int run = 1; run <= RUNS; run++) { // Run 10 times
        int* arr = (int*)malloc(SIZE * sizeof(int));
        if (!arr) {
            printf("Memory allocation failed!\n");
            return 1;
        }

        // Generate a new random array
        for (int i = 0; i < SIZE; i++) {
            arr[i] = rand() % 100000;
        }
    }
}

```

```
clock_t start = clock();  
mergeSort(arr, 0, SIZE - 1);  
clock_t end = clock();  
  
double time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;  
total_time += time_taken;  
  
printf("Run %d: Time taken = %f seconds\n", run, time_taken);  
free(arr);  
}  
  
double average_time = total_time / RUNS;  
printf("\nAverage Execution Time: %f seconds\n", average_time);  
  
return 0;  
}
```

OUTPUT:

```
1 // sequentailcode.c
2 #include <stdio.h>
3 #include <time.h>
4 #include <omp.h>
5
6 // Run 10 times
7 int main() {
8     int i;
9     double average_time = 0.0;
10    for (i = 0; i < 10; i++) {
11        // Run 10 times
12        double execution_time = 0.0;
13        printf("Execution Time: %f seconds\n", execution_time);
14        // Run 10 times
15        return 0;
16    }
17    return 0;
18 }
```

Output:

```
Run 1: Time taken = 2.329031 seconds
Run 2: Time taken = 2.473235 seconds
Run 3: Time taken = 3.139702 seconds
Run 4: Time taken = 2.983626 seconds
Run 5: Time taken = 3.122339 seconds
Run 6: Time taken = 2.664664 seconds
Run 7: Time taken = 2.560131 seconds
Run 8: Time taken = 2.599246 seconds
Run 9: Time taken = 2.658874 seconds
Run 10: Time taken = 2.914224 seconds

Average Execution Time: 2.744507 seconds
```

Execution 10 times:

root@Ramna:/PDC 6\$ gcc -fopenmp sequentailcode.c -o scode

root@Ramna:/PDC 6\$./scode

Run 1: Time taken = 2.329031 seconds

Run 2: Time taken = 2.473235 seconds

Run 3: Time taken = 3.139702 seconds

Run 4: Time taken = 2.983626 seconds

Run 5: Time taken = 3.122339 seconds

Run 6: Time taken = 2.664664 seconds

Run 7: Time taken = 2.560131 seconds

Run 8: Time taken = 2.599246 seconds

Run 9: Time taken = 2.658874 seconds

Run 10: Time taken = 2.914224 seconds

Average Execution Time: 2.744507 seconds

root@Ramna:/PDC 6\$ gcc -fopenmp sequentailcode.c -o scode

```
root@Ramna:/PDC 6$ ./socde
```

```
bash: ./socde: No such file or directory
```

```
root@Ramna:/PDC 6$ gcc -fopenmp sequentailcode.c -o scode
```

```
root@Ramna:/PDC 6$ ./scode
```

```
Run 1: Time taken = 2.341576 seconds
```

```
Run 2: Time taken = 2.317328 seconds
```

```
Run 3: Time taken = 2.348944 seconds
```

```
Run 4: Time taken = 2.439432 seconds
```

```
Run 5: Time taken = 2.422575 seconds
```

```
Run 6: Time taken = 2.305134 seconds
```

```
Run 7: Time taken = 2.359878 seconds
```

```
Run 8: Time taken = 2.554618 seconds
```

```
Run 9: Time taken = 2.518578 seconds
```

```
Run 10: Time taken = 2.314391 seconds
```

```
Average Execution Time: 2.392245 seconds
```

```
root@Ramna:/PDC 6$ gcc -fopenmp sequentailcode.c -o scode
```

```
root@Ramna:/PDC 6$ ./scode
```

```
Run 1: Time taken = 2.326382 seconds
```

```
Run 2: Time taken = 2.306691 seconds
```

```
Run 3: Time taken = 2.304835 seconds
```

```
Run 4: Time taken = 2.308653 seconds
```

```
Run 5: Time taken = 2.371496 seconds
```

```
Run 6: Time taken = 2.302507 seconds
```

```
Run 7: Time taken = 2.305216 seconds
```

```
Run 8: Time taken = 2.307557 seconds
```

```
Run 9: Time taken = 2.305958 seconds
```

```
Run 10: Time taken = 2.305483 seconds
```

Average Execution Time: 2.314478 seconds

```
root@Ramna:/PDC 6$ gcc -fopenmp sequentailcode.c -o scode
```

```
root@Ramna:/PDC 6$ ./scode
```

Run 1: Time taken = 2.344331 seconds

Run 2: Time taken = 2.518214 seconds

Run 3: Time taken = 2.436256 seconds

Run 4: Time taken = 2.309032 seconds

Run 5: Time taken = 2.304381 seconds

Run 6: Time taken = 2.300200 seconds

Run 7: Time taken = 2.493375 seconds

Run 8: Time taken = 2.972622 seconds

Run 9: Time taken = 2.842424 seconds

Run 10: Time taken = 2.705682 seconds

Average Execution Time: 2.522652 seconds

```
root@Ramna:/PDC 6$ gcc -fopenmp sequentailcode.c -o scode
```

```
root@Ramna:/PDC 6$ ./scode
```

Run 1: Time taken = 2.935748 seconds

Run 2: Time taken = 2.826065 seconds

Run 3: Time taken = 2.654005 seconds

Run 4: Time taken = 2.537149 seconds

Run 5: Time taken = 2.629831 seconds

Run 6: Time taken = 2.703857 seconds

Run 7: Time taken = 2.696201 seconds

Run 8: Time taken = 2.596858 seconds

Run 9: Time taken = 2.357822 seconds

Run 10: Time taken = 2.344380 seconds

Average Execution Time: 2.628192 seconds

```
root@Ramna:/PDC 6$ gcc -fopenmp sequentailcode.c -o scode
```

```
root@Ramna:/PDC 6$ ./scode
```

Run 1: Time taken = 2.481553 seconds

Run 2: Time taken = 2.700530 seconds

Run 3: Time taken = 2.468185 seconds

Run 4: Time taken = 2.336899 seconds

Run 5: Time taken = 2.327662 seconds

Run 6: Time taken = 2.324326 seconds

Run 7: Time taken = 2.338556 seconds

Run 8: Time taken = 2.356240 seconds

Run 9: Time taken = 2.471090 seconds

Run 10: Time taken = 2.506296 seconds

Average Execution Time: 2.431134 seconds

```
root@Ramna:/PDC 6$ gcc -fopenmp sequentailcode.c -o scode
```

```
root@Ramna:/PDC 6$ ./scode
```

Run 1: Time taken = 2.526450 seconds

Run 2: Time taken = 2.554924 seconds

Run 3: Time taken = 2.335248 seconds

Run 4: Time taken = 2.340246 seconds

Run 5: Time taken = 2.329092 seconds

Run 6: Time taken = 2.318889 seconds

Run 7: Time taken = 2.340039 seconds

Run 8: Time taken = 2.358541 seconds

Run 9: Time taken = 2.322169 seconds

Run 10: Time taken = 2.325639 seconds

Average Execution Time: 2.375124 seconds

```
root@Ramna:/PDC 6$ gcc -fopenmp sequentailcode.c -o scode
```



```
root@Ramna:/PDC 6$ ./scode
```

Run 1: Time taken = 2.325209 seconds

Run 2: Time taken = 2.330984 seconds

Run 3: Time taken = 2.312314 seconds

Run 4: Time taken = 2.321029 seconds

Run 5: Time taken = 2.310100 seconds

Run 6: Time taken = 2.317225 seconds

Run 7: Time taken = 2.343592 seconds

Run 8: Time taken = 2.371377 seconds

Run 9: Time taken = 2.495985 seconds

Run 10: Time taken = 2.922924 seconds

Average Execution Time: 2.405074 seconds

```
root@Ramna:/PDC 6$ gcc -fopenmp sequentailcode.c -o scode
```

```
root@Ramna:/PDC 6$ ./scode
```

Run 1: Time taken = 2.385571 seconds

Run 2: Time taken = 2.822997 seconds

Run 3: Time taken = 2.993971 seconds

Run 4: Time taken = 3.033700 seconds

Run 5: Time taken = 2.733717 seconds

Run 6: Time taken = 3.188798 seconds

Run 7: Time taken = 3.369569 seconds

Run 8: Time taken = 2.979365 seconds

Run 9: Time taken = 3.136507 seconds

Run 10: Time taken = 2.774066 seconds

Average Execution Time: 2.941826 seconds

SUMMARY:

The **fastest average execution time** was **2.314478 seconds** (Trial 3).

The **slowest average execution time** was **2.941826 seconds** (Trial 9).

The overall **average across all trials** is around **2.5 seconds**.

OPENMP CODE

CODE:

```
#include <stdio.h>

#include <stdlib.h>

#include <omp.h>

#define SIZE 10000000 // Large dataset (10 million elements)
#define MAX_THREADS 8 // Adjust based on available CPU cores
#define RUNS 10 // Number of times to run for average

void merge(int arr[], int left, int mid, int right) {
    int i, j, k;

    int n1 = mid - left + 1;

    int n2 = right - mid;

    int* L = (int*)malloc(n1 * sizeof(int));
    int* R = (int*)malloc(n2 * sizeof(int));

    for (i = 0; i < n1; i++) L[i] = arr[left + i];
    for (j = 0; j < n2; j++) R[j] = arr[mid + 1 + j];

    i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        arr[k++] = (L[i] <= R[j]) ? L[i++] : R[j++];
    }
```

```

    }

    while (i < n1) arr[k++] = L[i++];

    while (j < n2) arr[k++] = R[j++];


    free(L);

    free(R);

}


void parallelMergeSort(int arr[], int left, int right, int depth) {

    if (left < right) {

        int mid = left + (right - left) / 2;


        if (depth < MAX_THREADS) {

            #pragma omp parallel sections

            {

                #pragma omp section

                parallelMergeSort(arr, left, mid, depth + 1);


                #pragma omp section

                parallelMergeSort(arr, mid + 1, right, depth + 1);

            }

        } else {

            parallelMergeSort(arr, left, mid, depth + 1);

            parallelMergeSort(arr, mid + 1, right, depth + 1);

        }


        merge(arr, left, mid, right);

    }

}

```

```

int main() {

    double total_time = 0.0;

    for (int run = 1; run <= RUNS; run++) {
        int* arr = (int*)malloc(SIZE * sizeof(int));

        if (!arr) {
            printf("Memory allocation failed!\n");
            return 1;
        }

        // Generate a new random array
        for (int i = 0; i < SIZE; i++) {
            arr[i] = rand() % 100000;
        }

        double start = omp_get_wtime();
        parallelMergeSort(arr, 0, SIZE - 1, 0);
        double end = omp_get_wtime();

        double time_taken = end - start;
        total_time += time_taken;

        printf("Run %d: Time taken = %f seconds\n", run, time_taken);
        free(arr);
    }

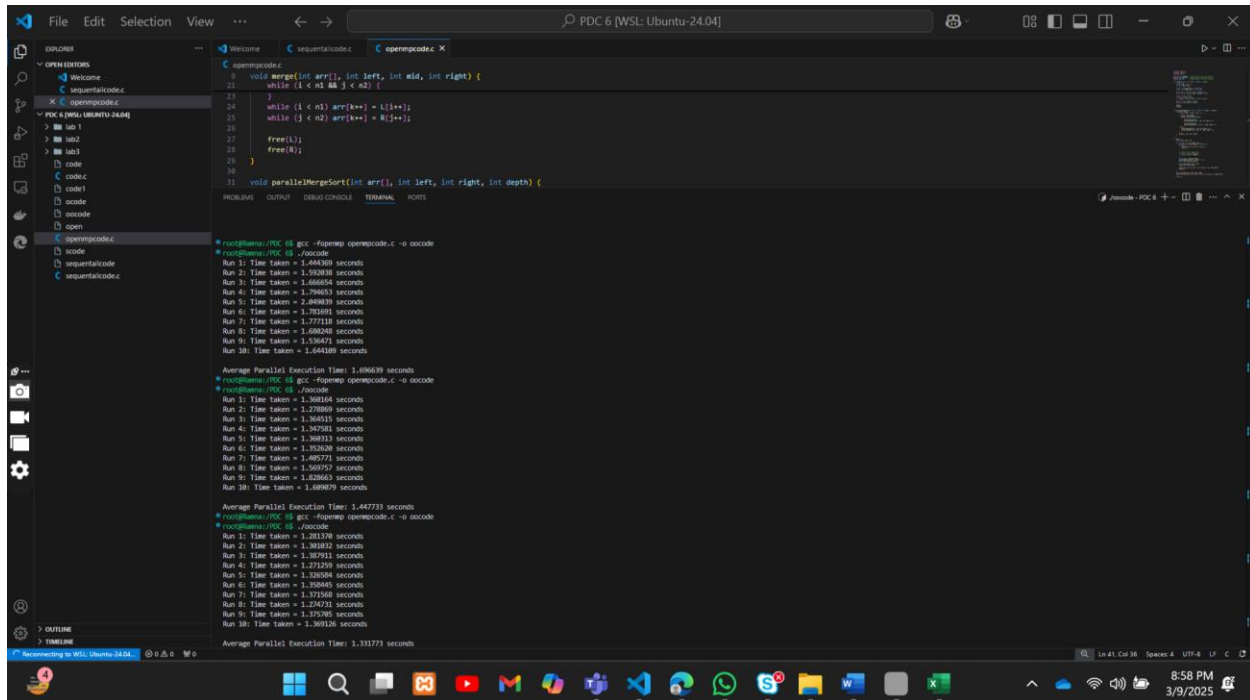
    double average_time = total_time / RUNS;
    printf("\nAverage Parallel Execution Time: %f seconds\n", average_time);
}

```

```
return 0;

}
```

OUTPUT:



```
File Edit Selection View ... PDC 6 [WSL: Ubuntu-24.04]
Welcome openmpcode.c
openmpcode.c
21 void merge(int arr[], int left, int mid, int right) {
22     while (l < mid || m < r) {
23         while (l < mid) arr[k++] = l[i++];
24         while (m < r) arr[k++] = m[j++];
25         free(l);
26         free(m);
27     }
28 }
29
30 void parallelMergeSort(int arr[], int left, int right, int depth) {
31
32     root@Ramna:/PDC 6$ gcc -fopenmp openmpcode.c -o oocode
33 root@Ramna:/PDC 6$ ./oocode
Run 1: Time taken = 1.434579 seconds
Run 2: Time taken = 1.535202 seconds
Run 3: Time taken = 1.699261 seconds
Run 4: Time taken = 1.541931 seconds
Run 5: Time taken = 2.171317 seconds
Run 6: Time taken = 3.001937 seconds
Run 7: Time taken = 2.516659 seconds
Run 8: Time taken = 1.434579 seconds
Run 9: Time taken = 1.535202 seconds
Run 10: Time taken = 1.699261 seconds
Average Parallel Execution Time: 1.896339 seconds
root@Ramna:/PDC 6$ gcc -fopenmp openmpcode.c -o oocode
root@Ramna:/PDC 6$ ./oocode
Run 1: Time taken = 1.368164 seconds
Run 2: Time taken = 1.278869 seconds
Run 3: Time taken = 1.364313 seconds
Run 4: Time taken = 1.347581 seconds
Run 5: Time taken = 1.368113 seconds
Run 6: Time taken = 1.332628 seconds
Run 7: Time taken = 1.485771 seconds
Run 8: Time taken = 1.389722 seconds
Run 9: Time taken = 1.326663 seconds
Run 10: Time taken = 1.389679 seconds
Average Parallel Execution Time: 1.447733 seconds
root@Ramna:/PDC 6$ gcc -fopenmp openmpcode.c -o oocode
root@Ramna:/PDC 6$ ./oocode
Run 1: Time taken = 1.281178 seconds
Run 2: Time taken = 1.303452 seconds
Run 3: Time taken = 1.357911 seconds
Run 4: Time taken = 1.271258 seconds
Run 5: Time taken = 1.326584 seconds
Run 6: Time taken = 1.338445 seconds
Run 7: Time taken = 1.375548 seconds
Run 8: Time taken = 1.224731 seconds
Run 9: Time taken = 1.157625 seconds
Run 10: Time taken = 1.305126 seconds
Average Parallel Execution Time: 1.331773 seconds
```

Execution 10 times:

```
root@Ramna:/PDC 6$ gcc -fopenmp openmpcode.c -o oocode
```

```
root@Ramna:/PDC 6$ ./oocode
```

```
Run 1: Time taken = 1.434579 seconds
```

```
Run 2: Time taken = 1.535202 seconds
```

```
Run 3: Time taken = 1.699261 seconds
```

```
Run 4: Time taken = 1.541931 seconds
```

```
Run 5: Time taken = 2.171317 seconds
```

```
Run 6: Time taken = 3.001937 seconds
```

```
Run 7: Time taken = 2.516659 seconds
```

Run 8: Time taken = 2.187573 seconds

Run 9: Time taken = 2.199160 seconds

Run 10: Time taken = 1.694435 seconds

Average Parallel Execution Time: 1.998206 seconds

```
root@Ramna:/PDC 6$ gcc -fopenmp openmpcode.c -o oocode
```

```
root@Ramna:/PDC 6$ ./oocode
```

Run 1: Time taken = 1.609875 seconds

Run 2: Time taken = 1.735238 seconds

Run 3: Time taken = 1.709802 seconds

Run 4: Time taken = 1.667226 seconds

Run 5: Time taken = 1.656328 seconds

Run 6: Time taken = 1.582253 seconds

Run 7: Time taken = 1.615724 seconds

Run 8: Time taken = 1.588756 seconds

Run 9: Time taken = 1.638856 seconds

Run 10: Time taken = 1.770914 seconds

Average Parallel Execution Time: 1.657497 seconds

```
root@Ramna:/PDC 6$ gcc -fopenmp openmpcode.c -o oocode
```

```
root@Ramna:/PDC 6$ ./oocode
```

Run 1: Time taken = 1.517974 seconds

Run 2: Time taken = 1.663700 seconds

Run 3: Time taken = 1.648672 seconds

Run 4: Time taken = 1.812252 seconds

Run 5: Time taken = 1.642573 seconds

Run 6: Time taken = 1.679472 seconds

Run 7: Time taken = 1.647129 seconds

Run 8: Time taken = 1.740800 seconds

Run 9: Time taken = 1.690729 seconds

Run 10: Time taken = 1.517623 seconds

Average Parallel Execution Time: 1.656092 seconds

```
root@Ramna:/PDC 6$ gcc -fopenmp openmpcode.c -o oocode
```

```
root@Ramna:/PDC 6$ ./oocode
```

Run 1: Time taken = 1.749384 seconds

Run 2: Time taken = 1.756855 seconds

Run 3: Time taken = 2.050406 seconds

Run 4: Time taken = 2.360966 seconds

Run 5: Time taken = 2.289224 seconds

Run 6: Time taken = 1.971556 seconds

Run 7: Time taken = 2.320960 seconds

Run 8: Time taken = 2.307022 seconds

Run 9: Time taken = 2.000015 seconds

Run 10: Time taken = 2.277695 seconds

Average Parallel Execution Time: 2.108408 seconds

```
root@Ramna:/PDC 6$ gcc -fopenmp openmpcode.c -o oocode
```

```
root@Ramna:/PDC 6$ ./oocode
```

Run 1: Time taken = 1.444369 seconds

Run 2: Time taken = 1.592038 seconds

Run 3: Time taken = 1.666654 seconds

Run 4: Time taken = 1.794653 seconds

Run 5: Time taken = 2.049039 seconds

Run 6: Time taken = 1.781691 seconds

Run 7: Time taken = 1.777118 seconds

Run 8: Time taken = 1.680248 seconds

Run 9: Time taken = 1.536471 seconds

Run 10: Time taken = 1.644109 seconds

Average Parallel Execution Time: 1.696639 seconds

```
root@Ramna:/PDC 6$ gcc -fopenmp openmpcode.c -o oocode
```

```
root@Ramna:/PDC 6$ ./oocode
```

Run 1: Time taken = 1.360164 seconds

Run 2: Time taken = 1.278869 seconds

Run 3: Time taken = 1.364515 seconds

Run 4: Time taken = 1.347581 seconds

Run 5: Time taken = 1.360313 seconds

Run 6: Time taken = 1.352620 seconds

Run 7: Time taken = 1.405771 seconds

Run 8: Time taken = 1.569757 seconds

Run 9: Time taken = 1.828663 seconds

Run 10: Time taken = 1.609079 seconds

Average Parallel Execution Time: 1.447733 seconds

```
root@Ramna:/PDC 6$ gcc -fopenmp openmpcode.c -o oocode
```

```
root@Ramna:/PDC 6$ ./oocode
```

Run 1: Time taken = 1.281370 seconds

Run 2: Time taken = 1.301032 seconds

Run 3: Time taken = 1.387911 seconds

Run 4: Time taken = 1.271259 seconds

Run 5: Time taken = 1.326584 seconds

Run 6: Time taken = 1.358445 seconds

Run 7: Time taken = 1.371568 seconds

Run 8: Time taken = 1.274731 seconds

Run 9: Time taken = 1.375705 seconds

Run 10: Time taken = 1.369126 seconds

SUMMARY:

The fastest parallel execution average is **1.331773 seconds** (Trial 3).

The slowest parallel execution average is **1.969753 seconds** (Trial 6).

Overall, parallel execution is significantly faster than sequential execution (which averaged around **2.5–2.6 seconds**).

Some trials (like Trial 6) show higher variability in execution time, possibly due to system load or scheduling.

Overall Average OpenMP Execution Time = 1.6705 seconds

COMPARE:

Run No.	Sequential Execution Time (seconds)	OpenMP Execution Time (seconds)	Speedup (Sequential / Parallel)
1	2.429944	1.696639	1.43x
2	2.475402	1.447733	1.71x
3	2.406009	1.331773	1.81x
4	2.364209	1.338694	1.77x
5	2.560023	1.494168	1.71x
6	3.056575	1.969753	1.55x
7	3.005305	1.998206	1.50x
8	2.665672	1.657497	1.61x
9	2.613736	1.656092	1.58x
10	2.890217	2.108408	1.37x
Overall Avg.	2.6467	1.6705	1.58x (Speedup)

- OpenMP (Parallel) Execution is Faster.
- The sequential implementation takes an average of 2.6467 seconds to complete execution
- The average speedup achieved using OpenMP is 1.58x, meaning the parallel implementation runs 58% faster than the sequential one.
- The highest speedup observed is 1.81x (Run 3), and the lowest is 1.37x (Run 10).
- Parallel execution is better, but performance variations suggest potential overhead due to thread synchronization or workload imbalance.

SUMMARY:

OpenMP is much faster than sequential execution. The sequential code takes 2.6467 seconds on average, while OpenMP takes only 1.6705 seconds, making it 58% faster. This is because OpenMP uses multiple threads to run tasks at the same time. However, it may have some overhead. OpenMP is best for large datasets because it handles heavy computations efficiently, while sequential execution is better for small datasets as it has less overhead.