MCA Assignment

Title: **Implementation and Performance Analysis of Parallel and Serial Counting Sort Algorithm using OpenMP**

Submitted by:

Danish Kalam	1MS18CS038
Faisal Noor Rather	1MS18CS04
Ashwin Rao	1MS18CS031

Abstract

Count Sort works by counting the number of elements less than a[i] for each element a[i] in the list a and inserting a[i] into the list subscript position determined by count. When the algorithm completes, the original list is replaced with a temporary list.

Introduction

To implement the parallel Counting Sort using OpenMP.

We have to define the upper bound and arrays needed to the process.

```
#define max_num 100000
int n;
int a[max_num], sorted[max_num];
```

Here, ***a*** is the original array and ***sorted*** is the sorted array.

We have to define the main method as follows.

```
int main() {
   int i;
   printf("Enter the size of the data to be sorted (the maximum
   value is 100000, enter 0 to end) : ");
   while (scanf_s("%d", &n), n) {
      if (n >= max_num) {
         puts("Data size is too large, re-enter");
         continue;
      }
      generate();
      parallel();
      serial();
      printf("\nEnter the data size to be sorted, enter 0 to end :
      ");
   }
```

```
    return 0;
}
```

Here we have to create three methods.

```
generate();
parallel();
serial();
```

generate() is to create array with random numbers. parallel() to execute parallel Counting Sorting. And serial() is to execute serial Counting Sorting.

We can define the generate() as follows.

```
void generate() {
    srand(time(NULL));
    int i;
    for (i = 0; i < n; i++) {
        a[i] = rand() % 1000 + 1;
        sorted[i] = 0;
    }
}
```

We can define parallel() as follows.

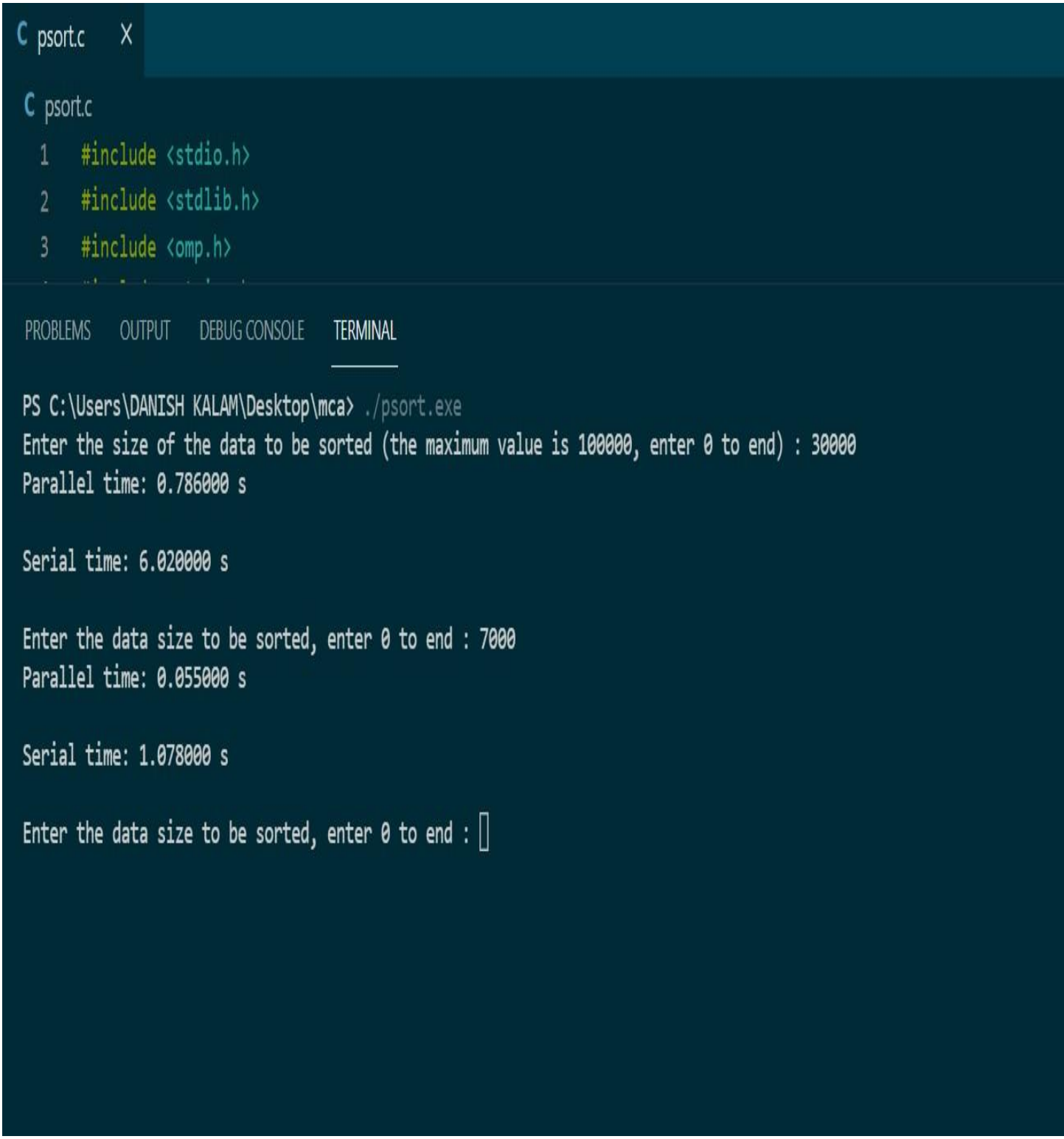
```
void parallel() {
    int i, j, count;
    omp_set_num_threads(50);
    double start_time = omp_get_wtime();
    #pragma omp parallel private(i, j, count)
    {
        #pragma omp for
        for (i = 0; i < n; i++) {
            count = 0;
            for (j = 0; j < n; j++) {
                if (a[i] > a[j])
                    count++;
            }
```

```
            while (sorted[count] != 0)
                count++;
            sorted[count] = a[i];
        }
    }    double end_time = omp_get_wtime();
    double time_used = end_time - start_time;
    printf("Parallel time: %f s\n", time_used);
}
```

We can define serial() as follows.

```
void serial() {
    int i, j, count;
    double start_time = omp_get_wtime();
    for (i = 0; i < n; i++) {
        count = 0;
        for (j = 0; j < n; j++) {
            if (a[i] > a[j]) {
                count++;
            }
        }
        while (sorted[count] != 0)
            count++;
        sorted[count] = a[i];
    }
    double end_time = omp_get_wtime();
    double time_used = end_time - start_time;
    printf("\nSerial time: %f s\n", time_used);
}
```

Output:


psort.c
psort.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <omp.h>
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PS C:\Users\DANISH KALAM\Desktop\mca> ./psort.exe
Enter the size of the data to be sorted (the maximum value is 100000, enter 0 to end) : 30000
Parallel time: 0.786000 s
Serial time: 6.020000 s
Enter the data size to be sorted, enter 0 to end : 7000
Parallel time: 0.055000 s
Serial time: 1.078000 s
Enter the data size to be sorted, enter 0 to end :