Computer Engineering &
Software Systems Program

Ain Shams University
Faculty of Engineering

Lab Assignment 1

Under Supervision of

Dr. Tamer Mostafa

&

Eng. Beshoy

Submitted By:

Ahmed Khaled Saad Ali Mekheimer
ID: 1809799

1. Problem Statement

Implementing the Selection Sort Algorithm using openMP, usually this algorithm is simple but problem is it edits in the same array that was unsorted to make it sorted, which is a big problem when using threads.
Threads writing/reading on/from same elements of array will cause a race condition, so we will have to write sorted numbers in another array.
Another problem is how many numbers from unsorted array will each thread take to sort.
Also, how to sort array with duplicate values, we will have 2 solutions one to sort array with duplicate values, and other for non-duplicate values in the array.

2. Solution & Output Screenshots

- **Explanation**
  We have TWO solutions(programs) ONE that partially handles duplicates (including duplicates in the implementation wasn't put as a requirement but I did it anyways) because srand() is what generates duplicate values.
  OTHER solution performs on non duplicate values in the array but array values are just from bigger to smaller.

  However, **Selection_Sort_OMP() Method** is the same in both programs, difference is **Fill_duplicates() method** will be called in 1st Solution which partially solves duplicates issue when using srand().
  OTHER solution doesn't use **Fill_duplicates() method.**

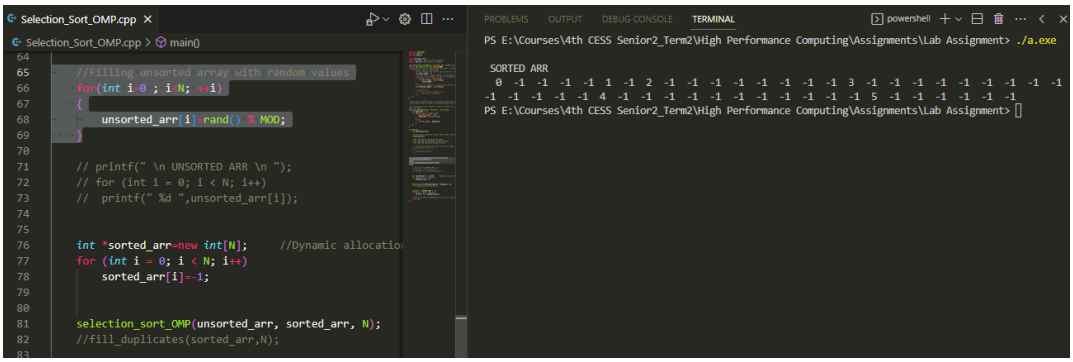
In the code, I have declared two for loops one that makes unsorted array values from bigger to smaller which is for ONLY 2nd Solution, 2nd loop makes its values from srand() with duplicate values which is ONLY for 1st Solution.

```
//ONLY USE ONE OF THE BELOW FOR LOOPS
    //1ST LOOP FOR NON DUPLICATE VALUES SOLUTION
    //2ND LOOP FOR DUPLICATE VALUES SOLUTION

    // //Filling unsorted array with values from bigger to smaller which will
make sort function sort all of the array
    // for(int i=0 ; i<N; i++)
    // {
    //  unsorted_arr[i]=N-i;
    // }


    // //Filling unsorted array with random values
    // for(int i=0 ; i<N; ++i)
    // {
    //  unsorted_arr[i]=rand() % MOD;
    // }
```

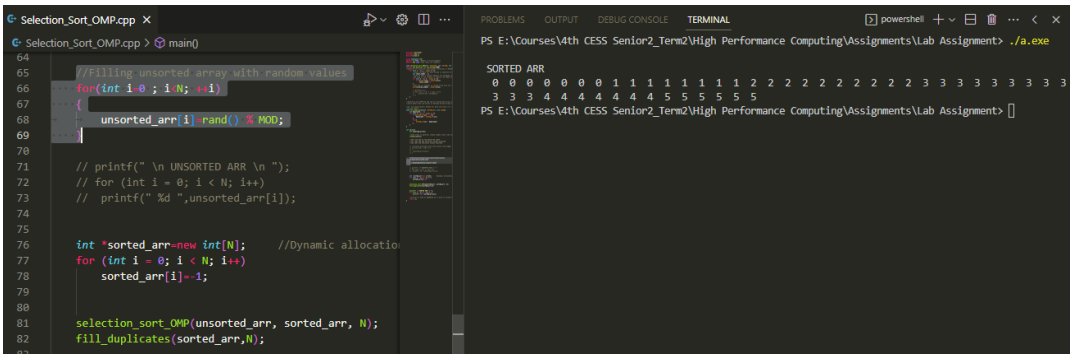
- **Selection_Sort_OMP() Method :**
  -We have 2 loops, inner & outer. Outer loop is to hold an element from unsorted array, Inner loop is to compare this element with the whole array elements.
  -We are using “parallel for” directive to distribute iterations of OUTER LOOP on threads which means that each thread is assigned number of iterations or number of elements from unsorted array to do the Inner loop.
  -In the Outer loop, a Local variable is assigned for each thread to identify Correct place of its assigned elements in sorted array.
  -In the Inner loop, a thread compares its assigned elements with all elements of unsorted array and as long as the element is smaller its index to be put in sorted array is incremented (local variable++)
  -This algorithm sorts array correctly, but it neglects duplicate values which means that old values in defined sorted array(-1) will remain.
  So, we have TWO Solutions:
  1) Use fill_duplicates() method because srand() is what generates duplicate values.
  2) Non duplicate values are in the array and array values are just from bigger to smaller, so no need to deal with duplicates.

- **1st Solution (Duplicates)**
- **Fill_duplicates() :**
  -For each old value (-1) replaces it with the value before it in the array so we must after calling Selection_Sort_OMP() Method in main we call fill_duplicates method, and to show its functionality we will run the program with/without calling fill_duplicates() with just 100 elements in the below screenshots. Unfortunately, it sometimes gets number of duplicates right and sometimes wrong.

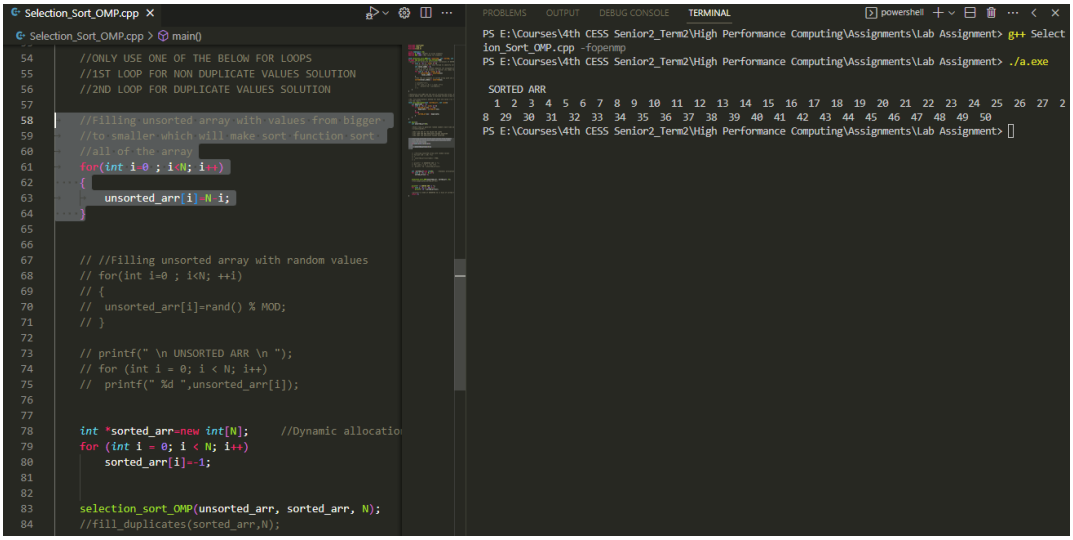
  Didn't call fill_duplicates():



  Called fill_duplicates():



- **2nd Solution (No Duplicates, No random values)**



- **Code**
- **1st Solution (Duplicates)**

```
#include <iostream>
#include <omp.h>
#include<time.h>

using namespace std;
#define N 10000 //Number of array elements
#define MOD 1000 //Max value for elements

void selection_sort_OMP(int *unsorted, int *sorted, int size){
#pragma omp parallel for num_threads(500)
//parallel for directive distributes iterations of OUTER LOOP on threads
    for (int i = 0; i < size; i++){
        //Local variable for each thread to identify Correct place of its
assigned elements in sorted array
        int local_index = 0;
        //Loop so that a thread compares its assigned elements with all elements
of unsorted array
        //As long as the element is smaller its index to be put in sorted array
is incremented
        for (int j = 0; j < size; j++){
            if (unsorted[j] < unsorted[i])
                local_index++;
        }
        //Put the [i] element in sorted array with its local index
        sorted[local_index] = unsorted[i];

        // printf("\n");
        // for (int x = 0; x < size; x++){
        //     printf("%d ",sorted[x]);
        // }
    }
}

//Selection_sort_OMP did the job of sorting the array, but it ignores duplicate
value and leave their indexes
//which means that old values in defined sorted array(-1) will remain

//So, fill_duplicates() method for each old value (-1) replaces it with the value
before it in the array, explained more
//in the report.
void fill_duplicates(int *sorted_arr, int size){
```

```
        int duplicate = -1;
        for (int i = 0; i < size; i++){
        if (sorted_arr[i] != -1){
            duplicate = sorted_arr[i];
        }
        else{
            sorted_arr[i] = duplicate;
        }
    }
}

int main(){
    int unsorted_arr[N];

    //Used srand to generate random numbers each time we run program
    srand(time(0));


    //Filling unsorted array with random values
    for(int i=0 ; i<N; ++i)
    {
        unsorted_arr[i]=rand() % MOD;
    }

    // printf(" \n UNSORTED ARR \n ");
    // for (int i = 0; i < N; i++)
    //  printf(" %d ",unsorted_arr[i]);


    int *sorted_arr=new int[N];     //Dynamic allocation of sorted array with its
values being (-1)
    for (int i = 0; i < N; i++)
        sorted_arr[i]=-1;


    selection_sort_OMP(unsorted_arr, sorted_arr, N);    //Sorting array and
leaving duplicates
    fill_duplicates(sorted_arr,N);                      //Filling duplicates


    printf(" \n SORTED ARR \n ");
    for (int i = 0; i < 50; i++)
        printf(" %d ",sorted_arr[i]);
```

```
    //printf("\n SIZE of UNSORTED %d \n Size of sorted
%d",unsorted_arr,sizeof(sorted_arr));
    return 0;
}
```

- **2nd Solution (No Duplicates, No random values)**

```
#include <iostream>
#include <omp.h>
#include<time.h>

using namespace std;
#define N 10000 //Number of array elements
#define MOD 1000 //Max value for elements

void selection_sort_OMP(int *unsorted, int *sorted, int size){
#pragma omp parallel for num_threads(500)
//parallel for directive distributes iterations of OUTER LOOP on threads
    for (int i = 0; i < size; i++){
        //Local variable for each thread to identify Correct place of its
assigned elements in sorted array
        int local_index = 0;
        //Loop so that a thread compares its assigned elements with all elements
of unsorted array
        //As long as the element is smaller its index to be put in sorted array
is incremented
        for (int j = 0; j < size; j++){
            if (unsorted[j] < unsorted[i])
                local_index++;
        }
        //Put the [i] element in sorted array with its local index
        sorted[local_index] = unsorted[i];

        // printf("\n");
        // for (int x = 0; x < size; x++){
        //     printf("%d ",sorted[x]);
        // }
    }
}

//Selection_sort_OMP did the job of sorting the array, but it ignores duplicate
value and leave their indexes
//which means that old values in defined sorted array(-1) will remain

//So, fill_duplicates() method for each old value (-1) replaces it with the value
before it in the array, explained more
//in the report.
void fill_duplicates(int *sorted_arr, int size){
    int duplicate = -1;
```

```
        for (int i = 0; i < size; i++){
            if (sorted_arr[i] != -1){
            duplicate = sorted_arr[i];
        }
        else{
            sorted_arr[i] = duplicate;
        }
    }
}

int main(){
    int unsorted_arr[N];

    //Filling unsorted array with values from bigger
    //to smaller which will make sort function sort
    //all of the array
    for(int i=0 ; i<N; i++)
    {
        unsorted_arr[i]=N-i;
    }

    int *sorted_arr=new int[N];     //Dynamic allocation of sorted array with its
values being (-1)
    for (int i = 0; i < N; i++)
        sorted_arr[i]=-1;


    selection_sort_OMP(unsorted_arr, sorted_arr, N);    //Sorting array and
leaving duplicates

    printf(" \n SORTED ARR \n ");
    for (int i = 0; i < 50; i++)
        printf(" %d ",sorted_arr[i]);

    //printf("\n SIZE of UNSORTED %d \n Size of sorted
%d",unsorted_arr,sizeof(sorted_arr));
    return 0;
}
```