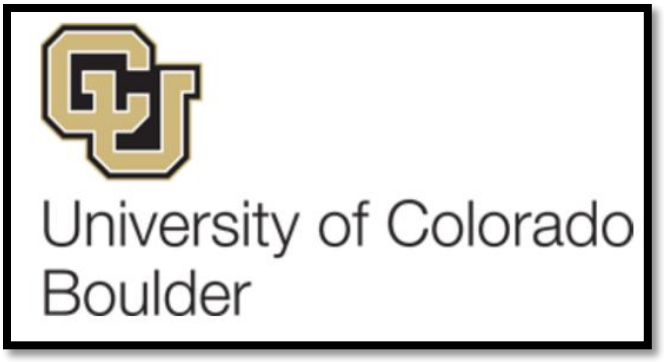


LAB - 0

CONCURRENT PROGRAMMING (ECEN 5033)

FALL 2019

REPORT BY:

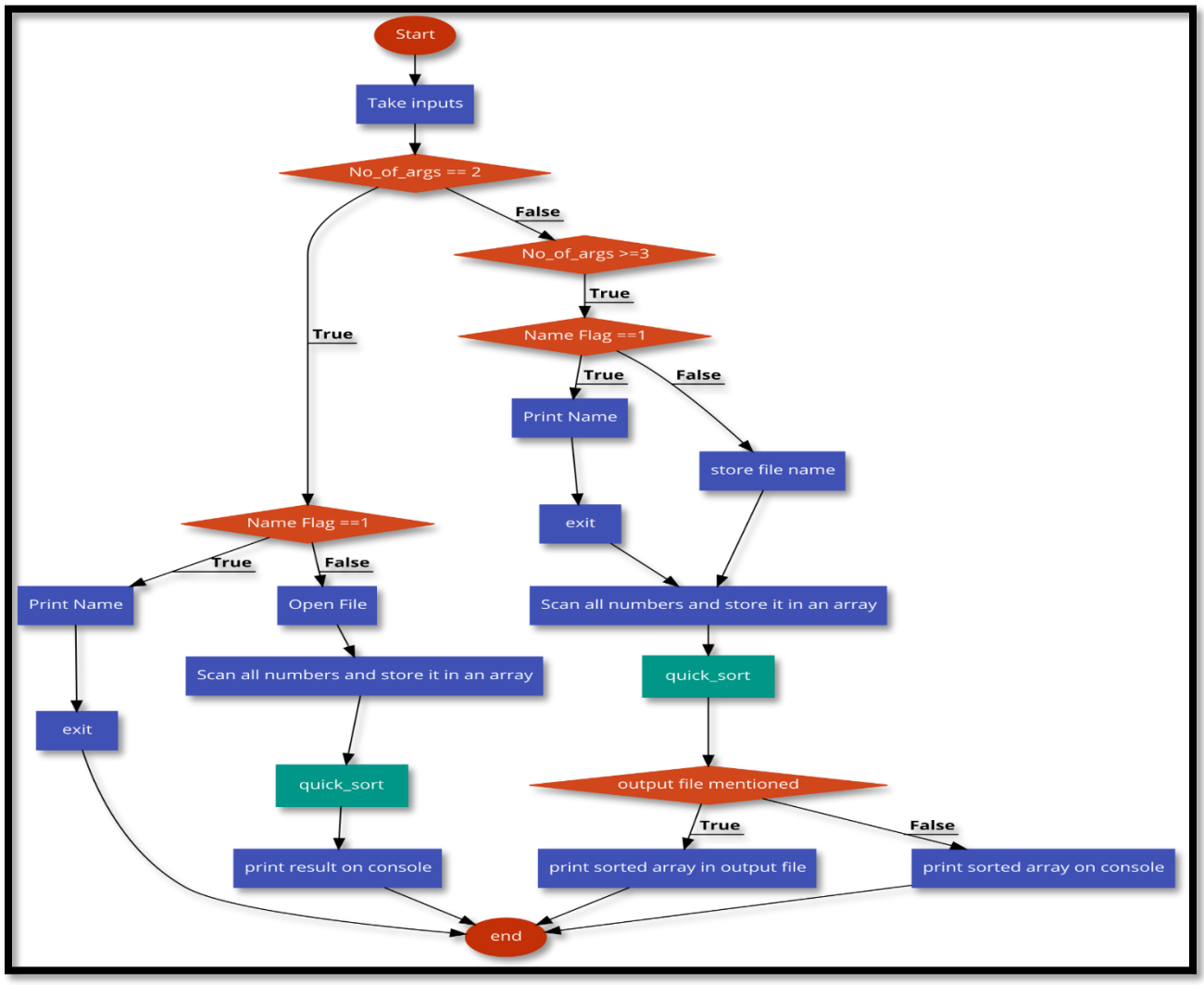
OM RAHEJA

WHY QUICK SORT?

Quick sort is an internal algorithm whereas merge sort is an external algorithm. Merge sort, like quick sort makes use of the divide and conquer strategy. Merge sort splits the array into two sub-arrays and requires additional storage for storing the sub-arrays. It also requires a third array to store the sorted array. This extra space used increases the running time of the algorithm. The quick sort algorithm on the other hand does not require any additional storage, thereby reducing the running time of the algorithm. Quick sort's efficiency is better than merge sort in case of small arrays. In a virtual memory environment, quick sort shows good cache locality and works faster than merge sort. The worst-case complexity for quick sort is $O(n^2)$ and the average case is O(nlogn). While merge sort splits the array into two equal sub-arrays, the partitioning of the array in case of quick sort is randomized. This reduces the chance for quick sort to encounter worst case scenario. So, in most cases the quick sort algorithm will execute in average case execution time and will be faster than merge sort.

CODE ORGANIZATION

This section of the report explains the basic code flow along with the quick sort algorithm in the form of flowchart.

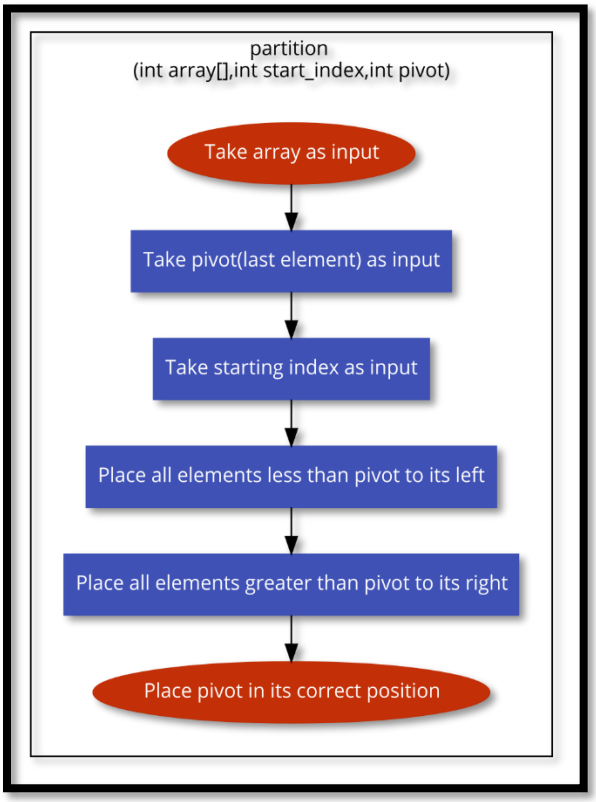


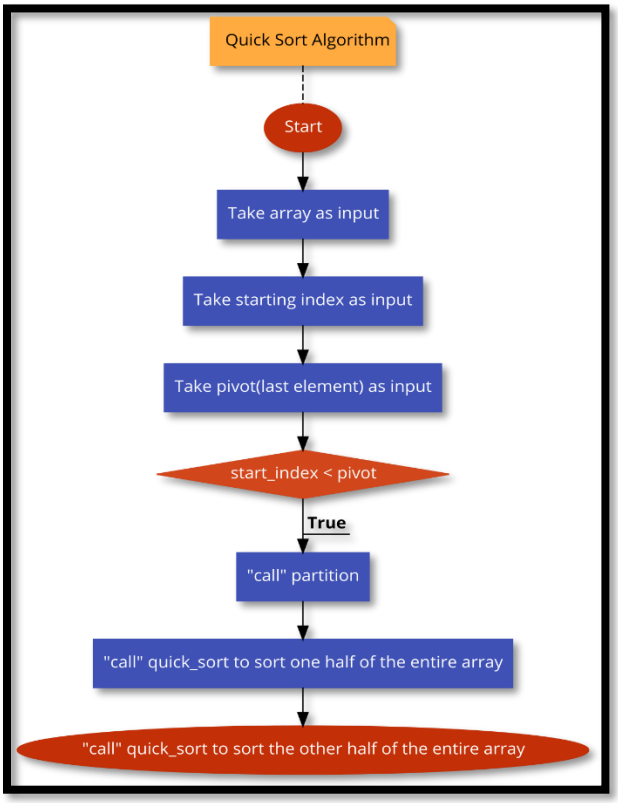
EXPLANATION OF QUICK SORT ALGORITHM:

The quick sort algorithm uses a divide and conquer method.

Steps:

1. Select the last element as the pivot.
2. Partition the entire array such that all elements less than the pivot are placed to the left of the pivot and all elements greater than the pivot are placed to the right to the pivot.
3. This way, the pivot is positioned at its correct location.
4. Repeat this until all the elements are sorted.





DESCRIPTION OF FILES SUBMITTED

1. **mysort.c/mysort.h**
   This file contains the main function. In the main, all the arguments mentioned in the command line are parsed and actions are carried out accordingly.
2. **print_in_file.c/ print_in_file.h**
   This file contains a function that takes the file name, array and the number of elements in the array as the input and prints the sorted array in the file.
3. **print_on_console.c/ print_on_console.h**
   This file contains a function that takes the file name, array and the number of elements in the array as the input and prints the sorted array on the console.

4. **quick_sort.c/ quick_sort.h**
   This file contains the implementation of quick sort algorithm. It contains two functions, one to partition the array and other to carry out the sorting.
5. **Makefile**
   Run this to create an executable.

COMPILATION & EXECUTION INSTRUCTIONS

1. The submission folder contains one folder named “Lab0”. Lab0 contains two sub folders in which all the header files are included in the folder named “inc” and all the source files including the Makefile are located in the “src” folder.
2. To run the code, go to the src folder and write the “make” command.
3. Once the executable is made follow the following commands to test the code.

Case 1:
$ make
$ ./mysort -n

Output:
Name = Om Raheja

Case 2:
$ make
$ ./mysort --name

Output:
Name = Om Raheja

Case 3:
$ make
$ ./mysort -n testfile.txt

Output:
Name = Om Raheja

Case 4:
$ make
$ ./mysort --name testfile.txt

Output:
Name = Om Raheja

Case 5: (a sample testfile is included for testing purpose)
$ make
$ ./mysort testfile.txt

Output:
Array:
-126 -67 -23 0 0 10 11 15 45 56 78 89 97

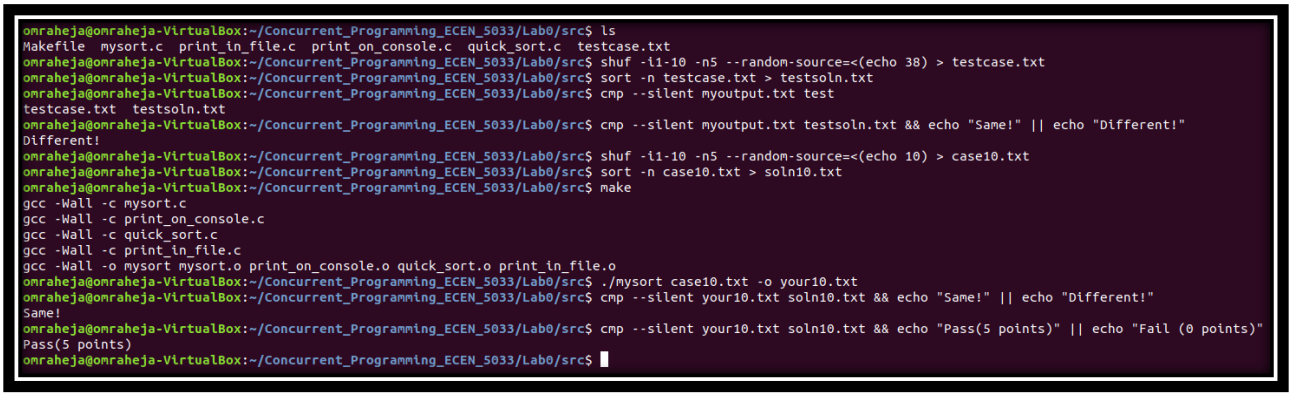
Case 6: (a sample testfile is included for testing purpose)
$ make
$ ./mysort testfile.txt -o output.txt
$ cat output.txt

Output:
-126
-67
-23
0
0
10
11
15
45
56
78
89
97

Case 7: (a sample testfile is included for testing purpose)
$ make
$ ./mysort-n testfile.txt -o output.txt

Output:
Name = Om Raheja

A sample screenshot of the commands executed as directed in the Lab0 document



REFERENCES

[1] Quick Sort vs Merge Sort : https://www.geeksforgeeks.org/quick-sort-vs-merge-sort/

[2] *Grokking Algorithms*, by *Aditya Bhargava*

[3] **GitHub Link**: https://github.com/omraheja/Concurrent_Programming_ECEN_5033