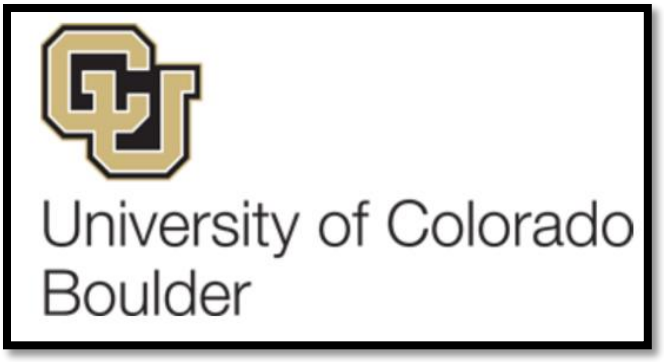


LAB - 1

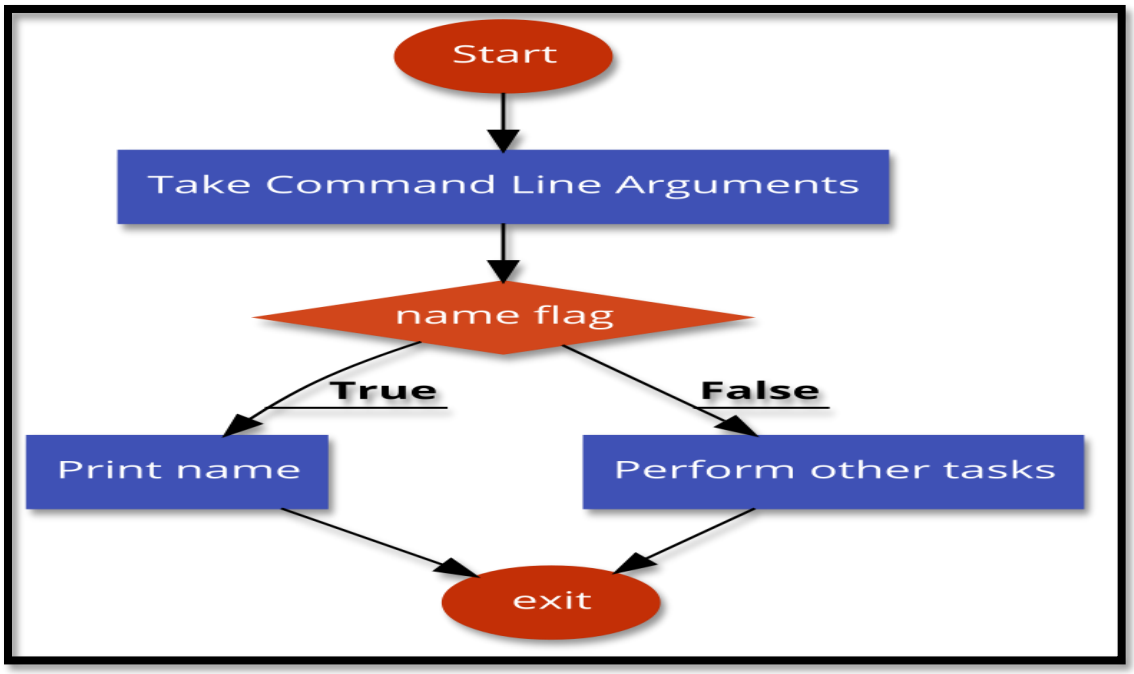
CONCURRENT PROGRAMMING (ECEN 5033)

FALL 2019

REPORT BY:

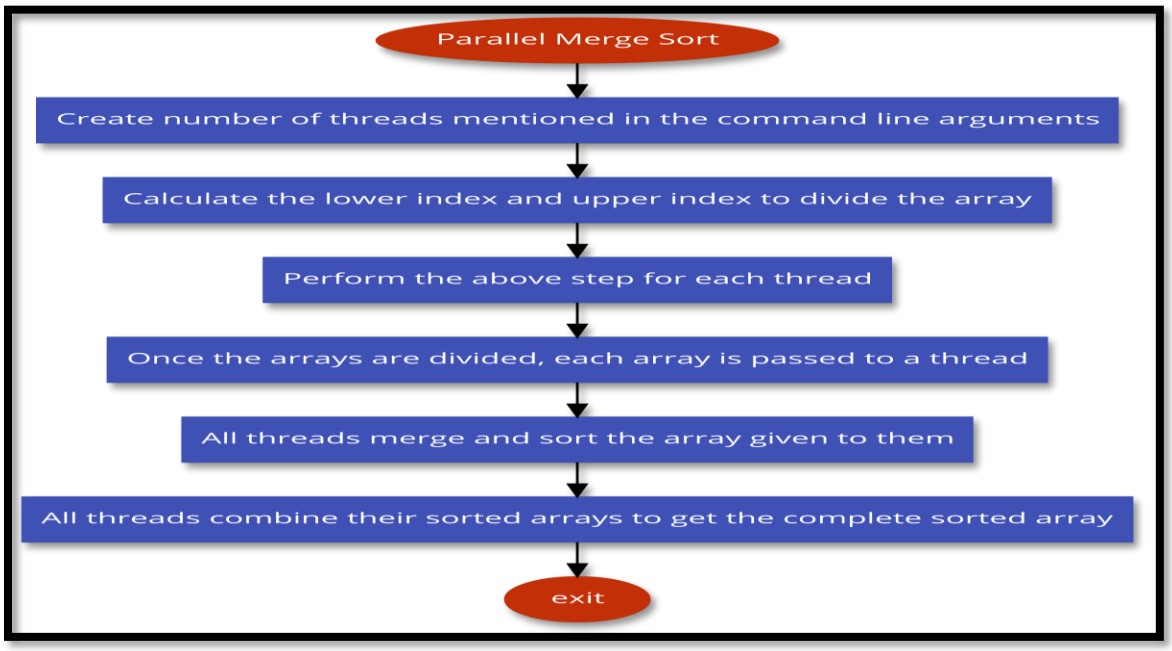
OM RAHEJA

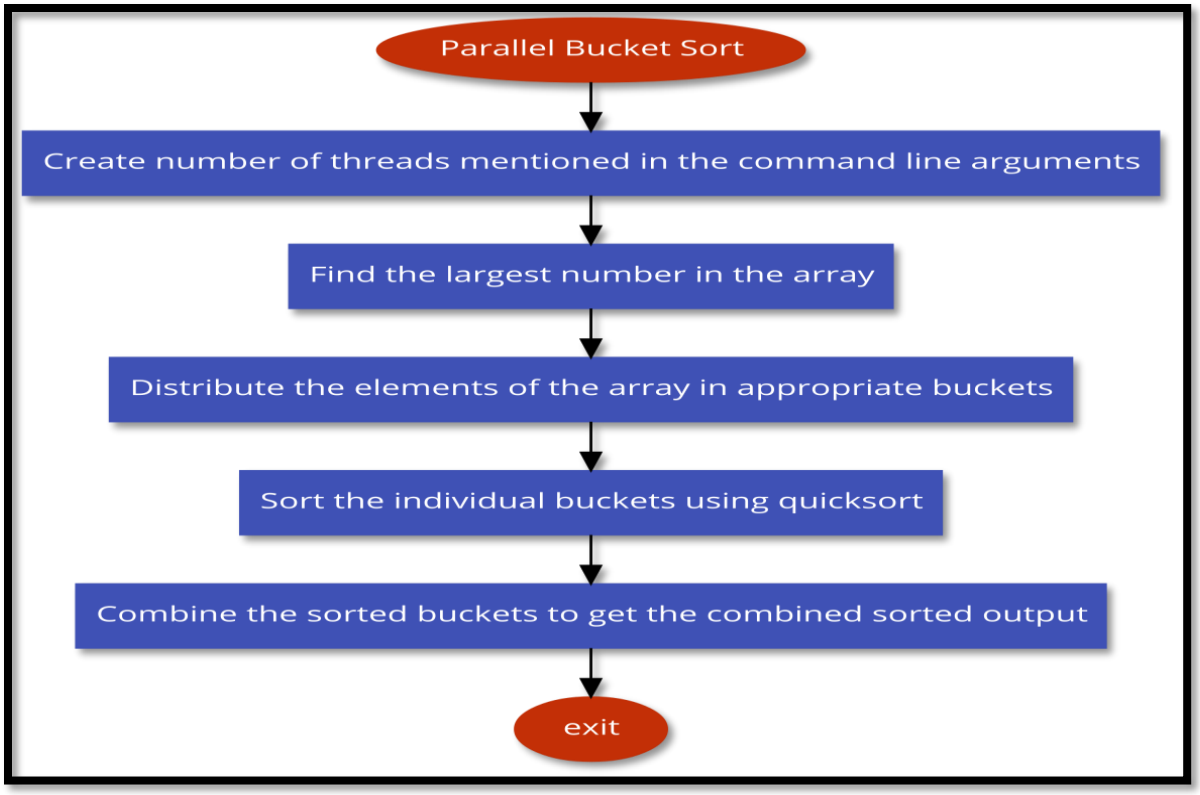
PARALLELIZATION TECHNIQUES



The program takes command line arguments and depending upon the arguments, carries out the tasks. The “Perform other tasks” mentioned in the above flowchart either performs a parallelized merge sort or a parallelized bucket sort. The decision of either performing the merge sort or the bucket sort depend on the parameters passed in the command line arguments.

The below flowchart will give a brief flow of the two parallelized algorithms, i.e. The parallelized merge sort and the parallelized bucket sort.



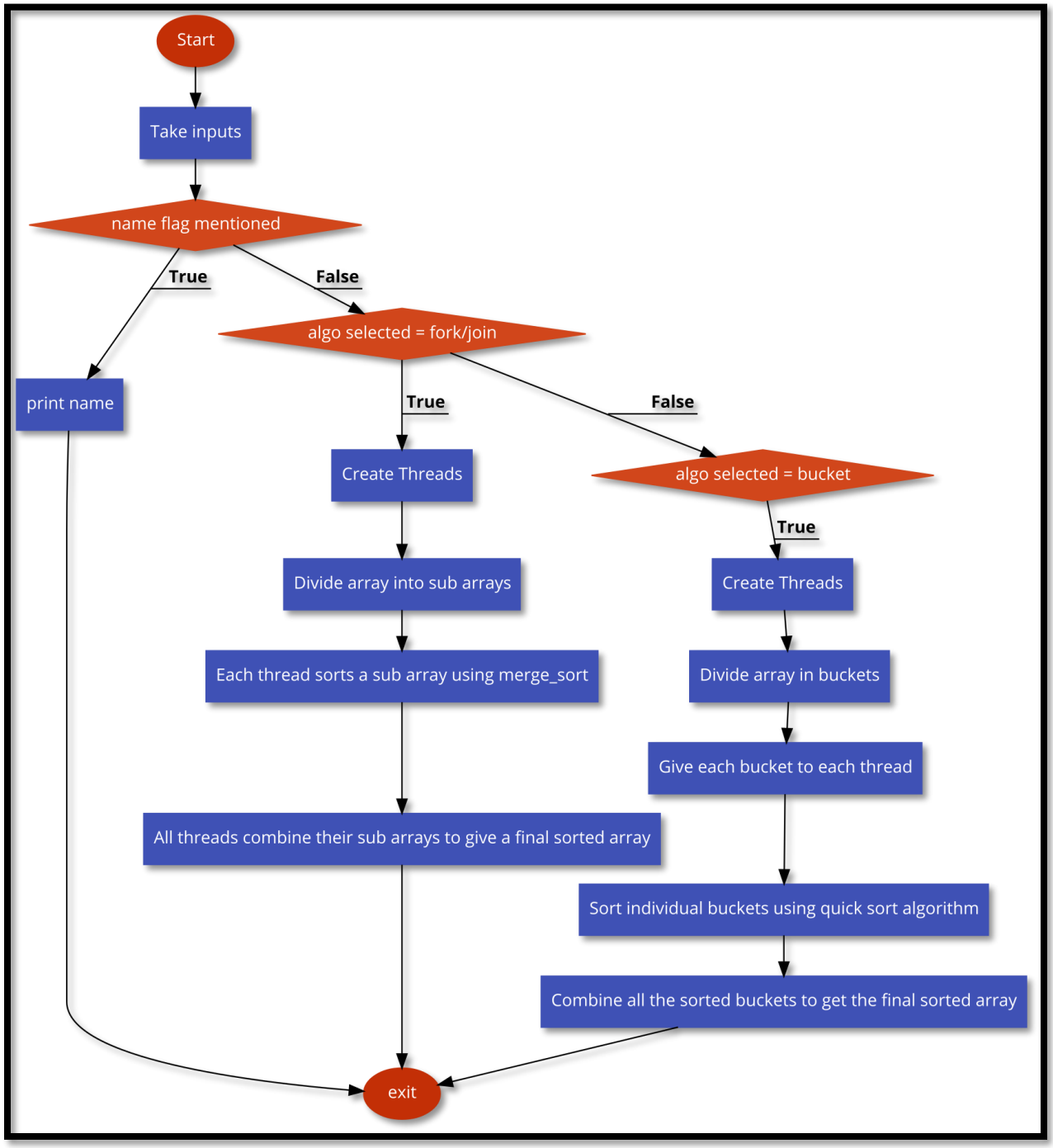


The above two flowcharts describe the parallelizing strategy. In the parallelized merge sort, arrays are divided in to sub arrays and given to each thread for sorting. The threads from 0 to n-1 get equal number of elements while the last thread taking the remaining load. Each thread sorts the array and waits at the barrier until all the other threads finish their sorting. Once, all the threads have finished sorting, they are merged to give the final sorted output.

In the parallelized bucket sort, the elements of the array are divided into buckets (one for each thread) and the individual buckets are sorted using quick sort. Once all the elements are sorted, they are combined to give a final sorted output.

CODE ORGANIZATION

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



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 “Lab1”. Lab1 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> -o <output_file_name.txt> -t 15 –alg=bucket

Output:
##Prints Time Elapsed in second and nanoseconds

SAMPLE SCREENSHOTS

Command Run:

```
omraheja@omraheja-VirtualBox:~/Concurrent_Programming_ECEN_5033/Lab1/src$ ./mysort test.txt -o output.txt -t 3 --alg=fj
Creating thread: 2
Creating thread: 3
Thread 2: 1 ; Thread_Part = 1
Thread 2 reporting for duty
Thread 1: 2 ; Thread_Part = 0
Thread 1 reporting for duty
Thread 3: 3 ; Thread_Part = 2
Thread 3 reporting for duty
Joined thread 2
Joined thread 3
Elapsed (ns): 460398
Elapsed (s): 0.000460
omraheja@omraheja-VirtualBox:~/Concurrent_Programming_ECEN_5033/Lab1/src$
```

Input File:

Output File:

REFERENCES

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

[2] References taken for codes have been mentioned in the header comments of the respective files.