Concurrent Programming

Lab 0 Report

Author: Sarayu Managoli

Program Overview:

The program is intended to implement quick and merge sort and the execution of these sorting algorithms is dependent upon the inputs provided by the user. This program is expected to function like “sort -n” Linux command and output the sorted values onto the required file.

Brief Introduction to Quick Sort:

As many sources quote, Quick sort involves the method of “divide-and-conquer”. In this algorithm, a value is first selected and named as a “pivot” value. The easiest and the simple way to select a pivot value is by selecting the first element as the pivot. This element is used to divide the whole array into 2. Quick sort partitions and calls itself twice to sort both the subarrays. At the end of the two sorts, one subarray holds the values lower than pivot and the other subarray holds the values greater than pivot. These two arrays are later merged to form the main array.

Brief Introduction of Merge Sort

It is said to be one of the most efficient sorting algorithms. Similar to quick sort, this too works on the principle of divide-and-conquer. It breaks down an array into smaller arrays until each of the arrays consists of only one element. Until the arrays are down to one element, merge sort is called. These single elements are later merged into forming a single sorted array. Here, the array is divided exactly from the middle element. Here additional storage is used as compared to quick sort. Merge sort does not work on the right array until the left array is sorted.

Difference between Quick Sort and Merge Sort

- Partition in quick sort is done in any ratio unlike merge sort which divides the array into exact half.
- Merge sort requires additional space since it is not in place. This is not the same in case of quick sort.
- Quick sort is more suitable for arrays while merge sort is for linked list.
- Quick sort is faster than merge sort.
- Data is stored in the main memory in the case of quick sort. Merge sort requires additional external storage.
- Quick sort is inefficient in case of larger arrays. Merge sort exhibits more efficiency.

Deliverables

As a part of the deliverables, the following artefacts are present in the ZIP folder:

1. Lab write-up
2. Source File (in folder Code)
3. Makefile (in folder Code)

Source File

This file contains the source code for the implementation of Lab 0. The following functions are included as a part of the source file:

1. read_from_file(char *read_file_name,vector<int> &readarray) - Reads the integer values from the file specified in the command line argument.
2. write_to_file(char *write_file_name,vector<int> &writearray) - Writes the integer values from the arrays and stores it in a file as mentioned.
3. main(int argc, char **argv) - Initialises the variables and calls other functions.

Merge sort functions:

4. mergefunction(vector<int>& mergearray, int left, int mid, int right) - Merges the left and the right subarrays onto the main array.
5. sort_merge(vector<int>& mergearray, int left, int right) - Implements merge sort by arranging the elements in an ascending order.

Quick sort functions:

6. quick_swap(int* a, int* b) - Swaps the input values referenced by their addresses
7. quick_partition (vector<int>& quickarray, int low, int high) - This function segregates the array by partitioning all the elements lesser than pivot on one side and all the other elements on the other side of pivot.
8. sort_quick(vector<int>& quickarray, int low, int high) -Implements quick sort by arranging the elements in an ascending order.

Makefile

Makefile consists of two targets, all and clean. The steps to execute are also printed in the file.

Compilation steps:

Open command prompt in the folder Lab0/Code, type "**make**". This will create the executable "mysort". To delete the executable, type "**make clean**".

Execution steps:

To print the author's name, type "**./mysort --name**"

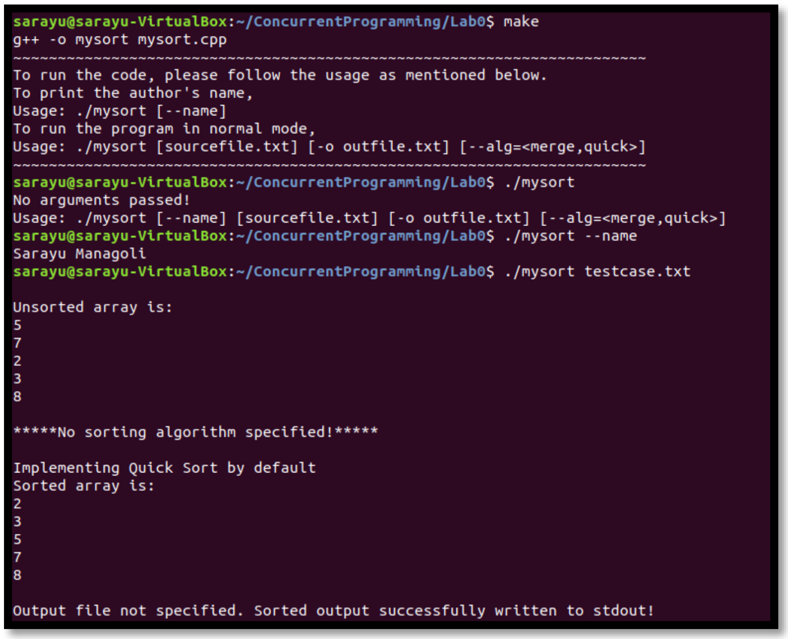
To run the program in normal mode, type "**./mysort [sourcefile.txt] [-o outfile.txt] [--alg=<merge,quick>]**"

Error handling and extant bug:

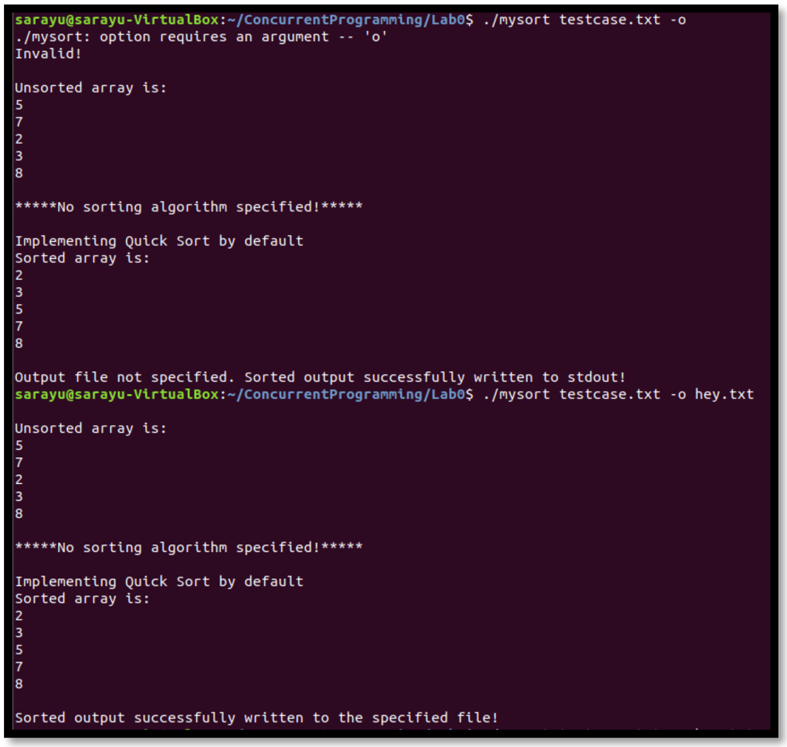
As indicated in the outputs below, error handling has been done if incremental inputs are missing.

Extant bugs: The program does not handle the errors related to the inputs from the user being out of order. This can be considered as future scope of work.

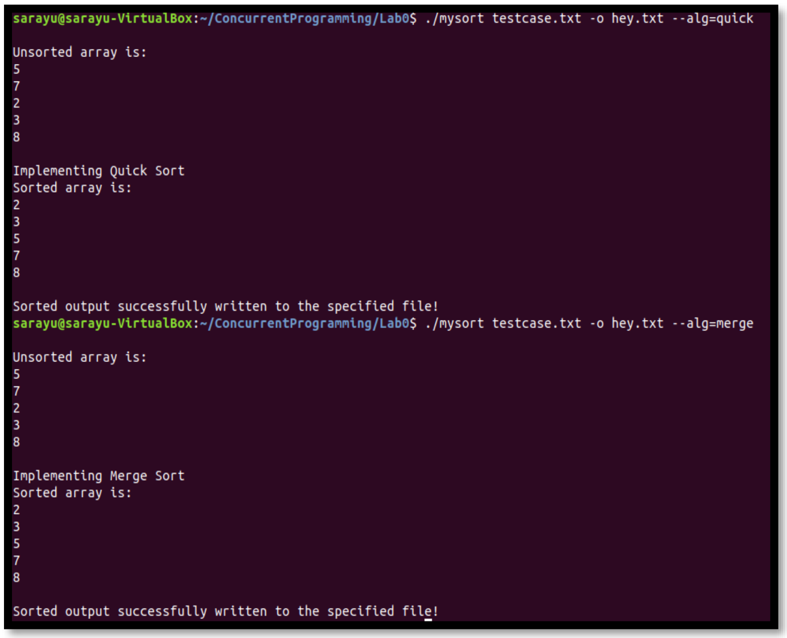
Output:



```
sarayu@sarayu-VirtualBox:~/ConcurrentProgramming/Lab0$ make
g++ -o mysort mysort.cpp
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
To run the code, please follow the usage as mentioned below.
To print the author's name,
Usage: ./mysort [--name]
To run the program in normal mode,
Usage: ./mysort [sourcefile.txt] [-o outfile.txt] [--alg=<merge,quick>]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
sarayu@sarayu-VirtualBox:~/ConcurrentProgramming/Lab0$ ./mysort
No arguments passed!
Usage: ./mysort [--name] [sourcefile.txt] [-o outfile.txt] [--alg=<merge,quick>]
sarayu@sarayu-VirtualBox:~/ConcurrentProgramming/Lab0$ ./mysort --name
Sarayu Managoli
sarayu@sarayu-VirtualBox:~/ConcurrentProgramming/Lab0$ ./mysort testcase.txt

Unsorted array is:
5
7
2
3
8

*****No sorting algorithm specified!*****

Implementing Quick Sort by default
Sorted array is:
2
3
5
7
8

Output file not specified. Sorted output successfully written to stdout!
```



```
sarayu@sarayu-VirtualBox:~/ConcurrentProgramming/Lab0$ ./mysort testcase.txt -o
./mysort: option requires an argument -- 'o'
Invalid!

Unsorted array is:
5
7
2
3
8

*****No sorting algorithm specified!*****

Implementing Quick Sort by default
Sorted array is:
2
3
5
7
8

Output file not specified. Sorted output successfully written to stdout!
sarayu@sarayu-VirtualBox:~/ConcurrentProgramming/Lab0$ ./mysort testcase.txt -o hey.txt

Unsorted array is:
5
7
2
3
8

*****No sorting algorithm specified!*****

Implementing Quick Sort by default
Sorted array is:
2
3
5
7
8

Sorted output successfully written to the specified file!
```



```
sarayu@sarayu-VirtualBox:~/ConcurrentProgramming/Lab0$ ./mysort testcase.txt -o hey.txt --alg=quick

Unsorted array is:
5
7
2
3
8

Implementing Quick Sort
Sorted array is:
2
3
5
7
8

Sorted output successfully written to the specified file!
sarayu@sarayu-VirtualBox:~/ConcurrentProgramming/Lab0$ ./mysort testcase.txt -o hey.txt --alg=merge

Unsorted array is:
5
7
2
3
8

Implementing Merge Sort
Sorted array is:
2
3
5
7
8

Sorted output successfully written to the specified file!
```

References

https://stackoverflow.com/questions/25833541/reading-numbers-from-file-c

https://www.geeksforgeeks.org/quick-sort/

https://www.geeksforgeeks.org/merge-sort/

https://codeyarns.com/2015/01/30/how-to-parse-program-options-in-c-using-getopt_long/