**LAB3 REPORT**

**Shreya Chakraborty**

**PROGRAM DESCRIPTION:**

The program implements mergesort algorithm. It tries to emulate 'sort -n' linux command.

It takes command line arguments of 3 types

a. --name -> it's a flag. Causes the program to return my name regardless of any other argument after it.

b. sourcefile.txt -> non option ARG element. This is the file to sort. If it’s not followed by anything else then the program outputs the sorted numbers one per line on the stdout.

c. outputfile.txt -> optional argument. If present, the program writes the sorted integers into the output file.

./mysort --name [sourcefile.txt] -o [outputfile.txt]

The program parses the command line arguments using the wrapper function ParseCommandline(...) which internally uses getopt\_long() in parse.c. It checks for the name flag, if found, the function prints my name and returns 1. The main then checks if the return value is 1, then returns 0. The parse function also stores the names of the input and output files from the command line by virtue of double pointer source and output file names in its parameter. The program then checks if the input file exists, if it does, then it opens it and reads it's content line by line using the API getline() until end of file, it converts the string in each line into integer and adds it to the input array. The API OMP\_merge() calls the OMP\_mergesort internally. Here the actual OpenMP parallelised sorting takes place. All the input and output files are available in the folder called 'files'

Lastly we compare our the output of our mergesort algorithm with the output of 'sort -n'. They should be same.

**PARALLELIZATION STRATEGY**

OpenMP supports multi-platform [shared memory](https://en.wikipedia.org/wiki/Shared_memory_architecture) [multiprocessing](https://en.wikipedia.org/wiki/Multiprocessing) programming in C and C++.

OpenMP uses #pragmas for thread creation, parallelization ,workload distribution and synchronization. The following pragmas have been used in the mergesort algorithm.

The section construct is used to distribute different tasks to different threads.The parallel construct creates a team of threads which execute in parallel, sections mark different blocks executing in parallel and distributes tasks to exisiting threads. Omp parallel sections directive is a combination of both constructs omp parallel and omp sections directives. we define a parallel region containing a single section directive in one step. Inside the mergesort function, if the number of threads is 1, then the normal iterative mergesort takes place. If the number threads is greater than 1, we apply openMP parallelization through pragmas. The #pragma omp parallel sections applies to the block of code that contains, recursive mergesort of sub arrays which are in turn inside the omp sections pragmas. Hence these blocks of codes are distributed to different threads in parallel. There is no fixed order of execution. The omp\_set\_nested is used to enable nested parallelism. The number of threads are set using omp\_set\_thread\_nums().

**CODE ORGANISATION:**

1. bin : contains the executable of the program "mysort"

2. doc : contains "output.txt" which has the entire standard output of the bashcript.

It also contains image of the corresponding input called bash\_input along with the Lab report.

3. files : contains all the testfiles, sorted testfiles by both mergesort as well as sort -n

4. includes : contains the header files (.h) of the Lab3

5. obj : contains the object files (.o) generated during make

6. script : contains the bashcript to test the entire program framework.

7. src : contains all the source files (.c) of Lab3

8. Makefile : supports make and make clean only

9. Readme : lab details

**BASH SCRIPT:**

The bashscript.sh in folder 'script' was my testing framework. It has an interactive and easy to use ui with 7 options. It can be used to generate random files for sorting. It uses the linux command 'shuf.' Based on the user input of range size and seed, the user can generate the required TestFile/TestFiles. All the TestFiles are stored in 'files'. Next it can also be used to sort the TestFile by both my sorting method and 'sort -n'. In my sorting method, it gives further options to choose algorithm (only mergesort has been attempted)and 'makes' it. It provides sub option like

sort for a single file or sort for all the 14 random generated TestFiles. The random generated testfiles and sorted and saved as 'mySortedTestFile'. The sorted files from sort -n option are saved as 'SortedTestFile'

We then compare the sorted files from both the options to see if the results match. For this we use the 'cmp' command line function. Finally we also have an option to delete all the text files in folder 'file' and make clean the code.

Folder 'doc' contains output.txt which is output of the bashcript and the image conatins the user input.

**COMPILATION INSTRUCTIONS:**

Makefile is in Lab0 folder. To compile just do $ make

and to clean just $ make clean

**EXECUTION INSTRUCTIONS:**

To execute normally go to the folder bin in Lab0. Thats where the executable mysort gets stored after make.

Then you can enter any of the configurations

$ ./mysort --name

My name is Shreya

$ ./mysort --name input.txt (name flag immediately returns)

My name is shreya

$ ./mysort input.txt

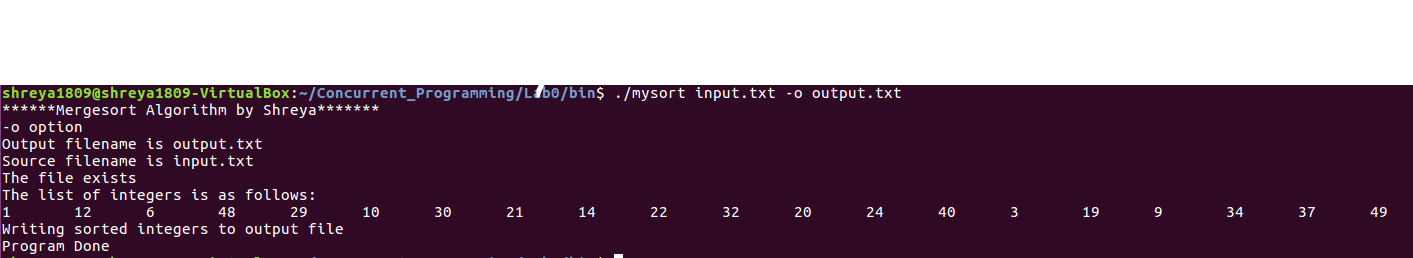
$ ./mysort input.txt -o output.txt

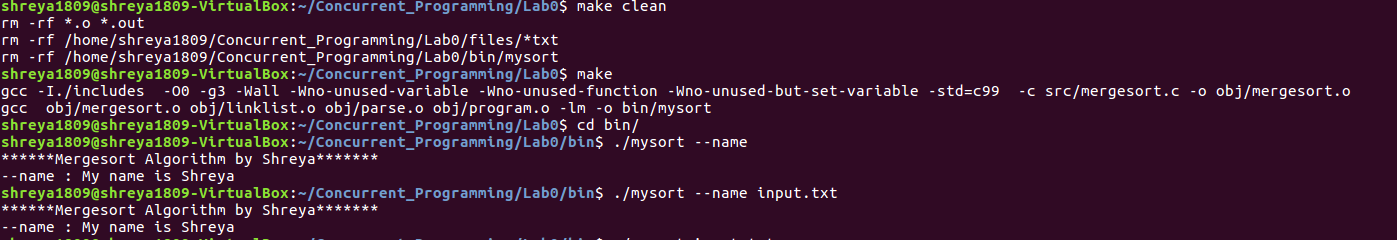
Alternatively you can use bashscript to automate everything. Go to in the folder 'script'.

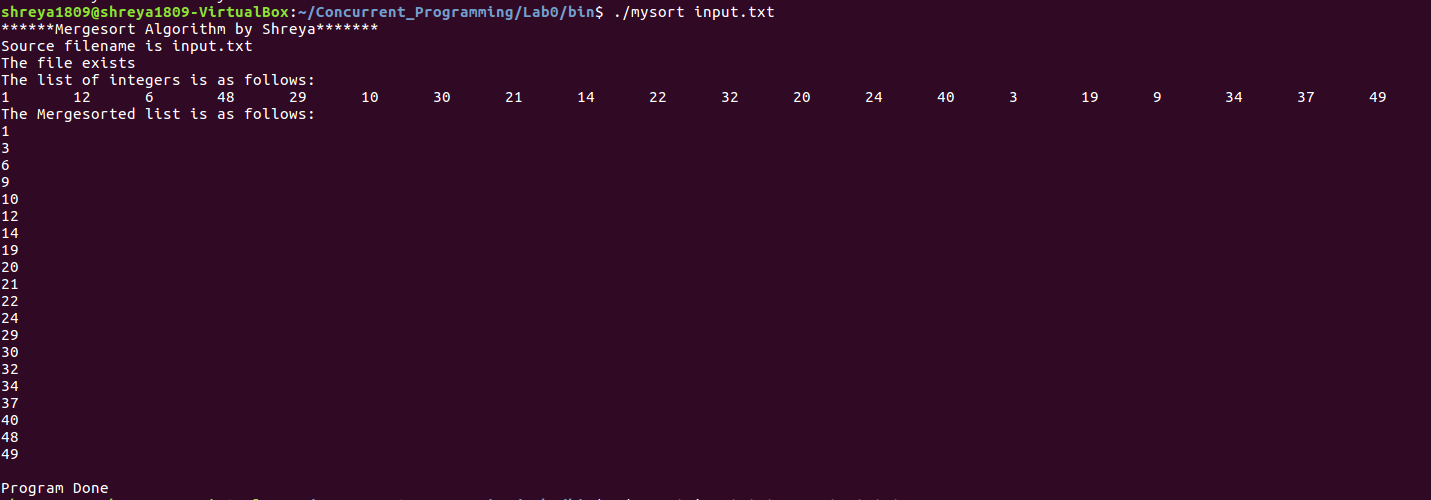
set permission and make it executable -> $ chmod +x bashcript.sh

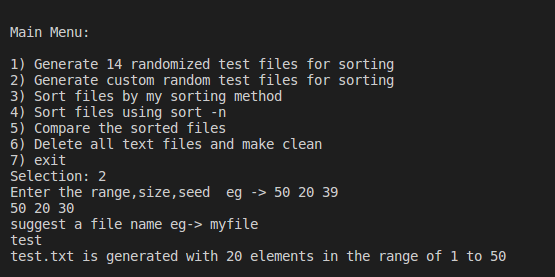
then -> $ ./bashcript.sh

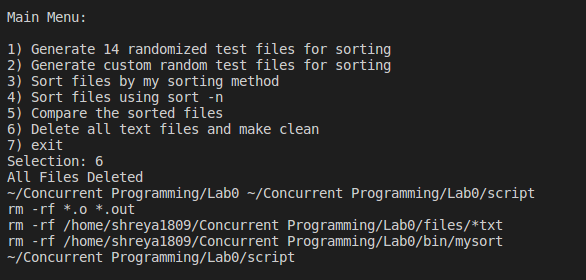
APPENDIX:

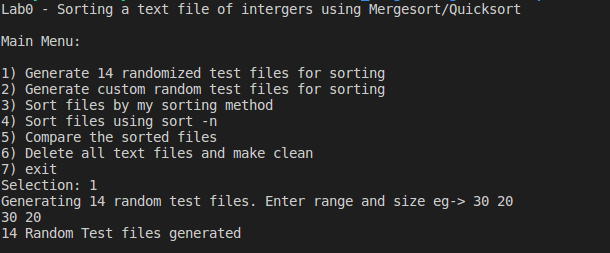


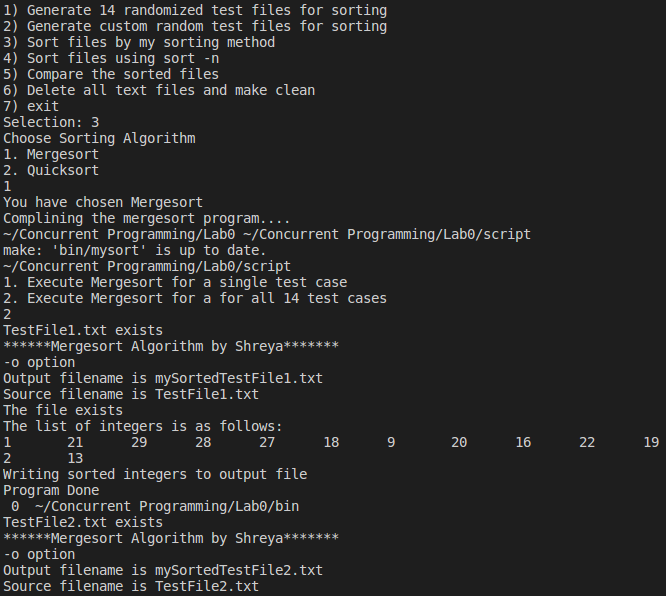


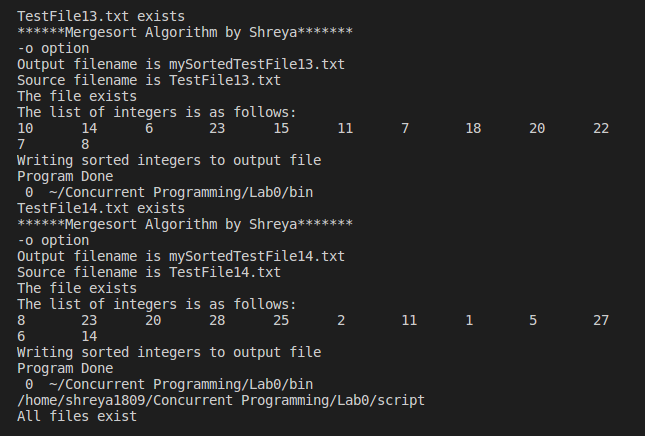


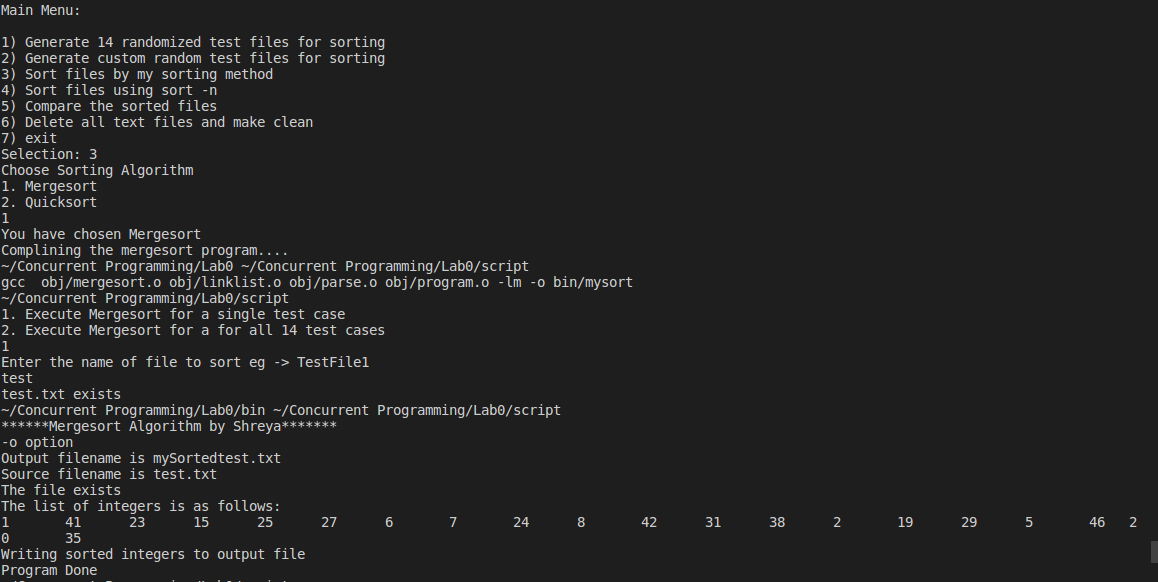


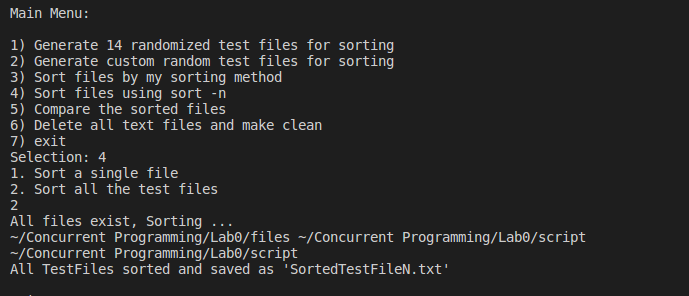


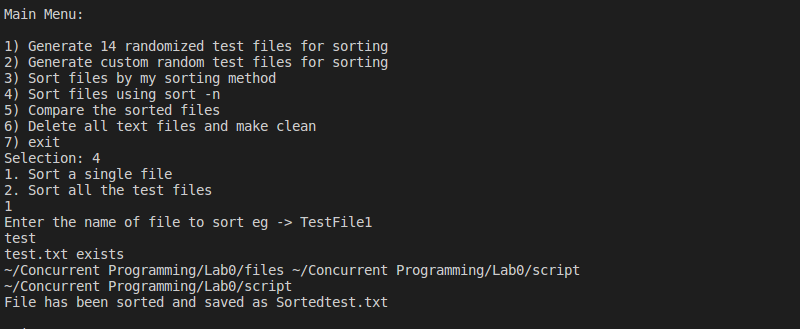


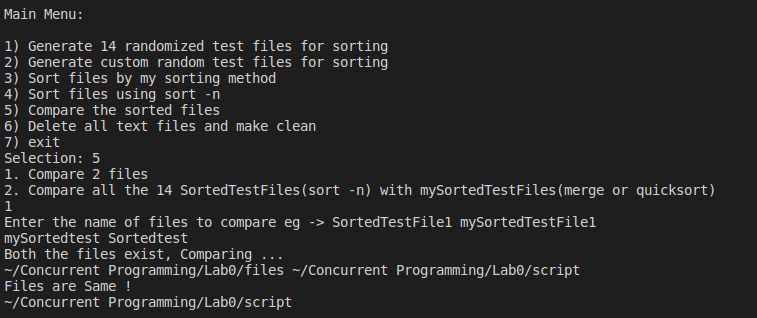


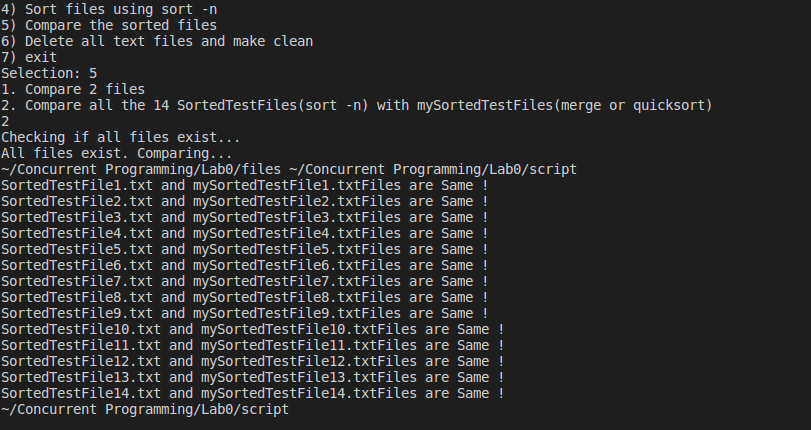


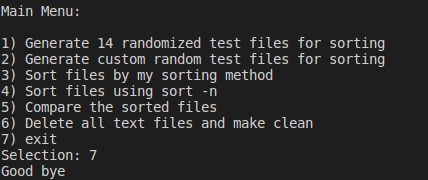












References:

1. <https://linuxconfig.org/bash-scripting-tutorial>
2. <https://stackoverflow.com/questions/30182086/how-to-use-goto-statement-in-shell-script?lq=1>
3. <https://www.gnu.org/software/libc/manual/html_node/Getopt-Long-Option-Example.html>
4. <https://stackoverflow.com/questions/30573481/path-include-and-src-directory-makefile/30602701>
5. <https://www.geeksforgeeks.org/merge-sort-for-linked-list/>
6. <https://hiltmon.com/blog/2013/07/03/a-simple-c-plus-plus-project-structure/>
7. <https://www.geeksforgeeks.org/iterative-merge-sort/>
8. <http://madhugnadig.com/articles/parallel-processing/2017/02/25/parallel-computing-in-c-using-openMP.html>
9. <http://ianfinlayson.net/class/cpsc425/notes/23-sorting>