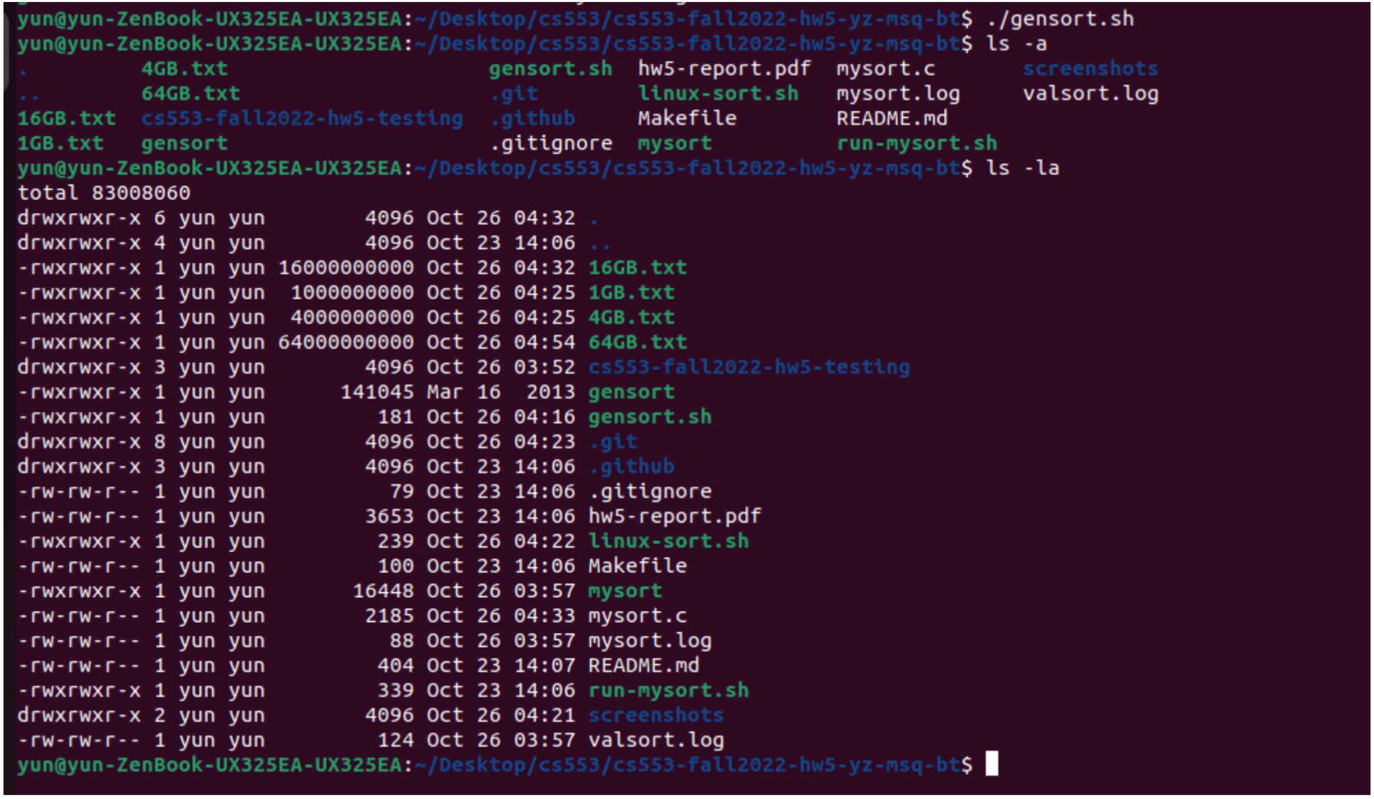
Team members:
Yun Zi
Bhavesh Rajesh Talreja
Muhammad Sheheryar Qureshi

Design Document:

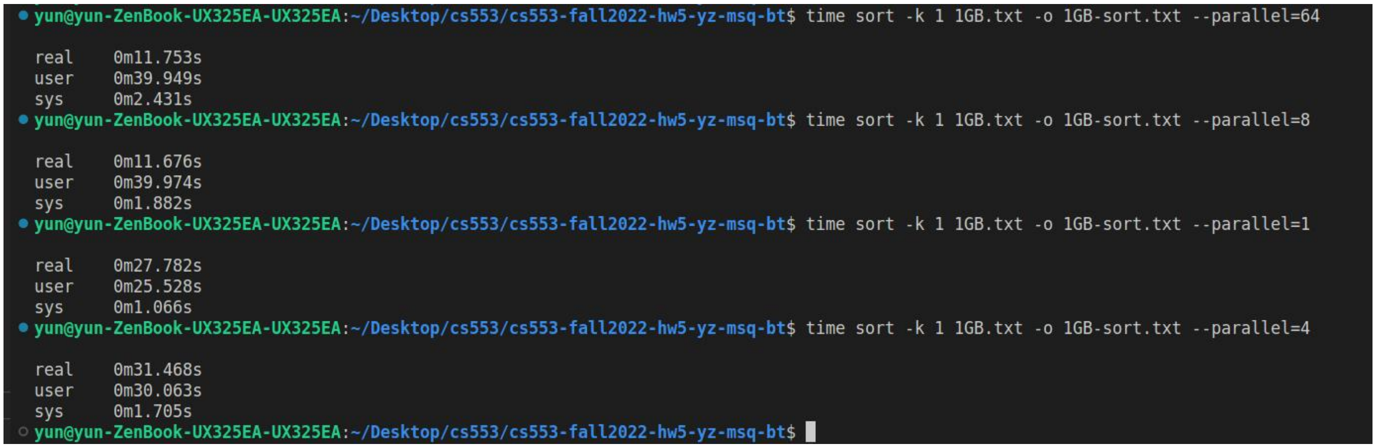
1. gensort to create test data which we have a script: ./gensort.sh to create 1GB, 4GB, 16GB and 64GB test data, to see screenshots:



```
yun@yun-ZenBook-UX325EA-UX325EA:~/Desktop/cs553/cs553-fall2022-hw5-yz-msq-bt$ ./gensort.sh
yun@yun-ZenBook-UX325EA-UX325EA:~/Desktop/cs553/cs553-fall2022-hw5-yz-msq-bt$ ls -a
.         4GB.txt                      gensort.sh  hw5-report.pdf  mysort.c       screenshots
..        64GB.txt                     .git        linux-sort.sh   mysort.log     valsort.log
16GB.txt  cs553-fall2022-hw5-testing   .github     Makefile        README.md
1GB.txt   gensort                      .gitignore  mysort          run-mysort.sh
yun@yun-ZenBook-UX325EA-UX325EA:~/Desktop/cs553/cs553-fall2022-hw5-yz-msq-bt$ ls -la
total 83008060
drwxrwxr-x 6 yun yun        4096 Oct 26 04:32 .
drwxrwxr-x 4 yun yun        4096 Oct 23 14:06 ..
-rwxrwxr-x 1 yun yun 16000000000 Oct 26 04:32 16GB.txt
-rwxrwxr-x 1 yun yun  1000000000 Oct 26 04:25 1GB.txt
-rwxrwxr-x 1 yun yun  4000000000 Oct 26 04:25 4GB.txt
-rwxrwxr-x 1 yun yun 64000000000 Oct 26 04:54 64GB.txt
drwxrwxr-x 3 yun yun        4096 Oct 26 03:52 cs553-fall2022-hw5-testing
-rwxrwxr-x 1 yun yun      141045 Mar 16  2013 gensort
-rwxrwxr-x 1 yun yun         181 Oct 26 04:16 gensort.sh
drwxrwxr-x 8 yun yun        4096 Oct 26 04:23 .git
drwxrwxr-x 3 yun yun        4096 Oct 23 14:06 .github
-rw-rw-r-- 1 yun yun          79 Oct 23 14:06 .gitignore
-rw-rw-r-- 1 yun yun        3653 Oct 23 14:06 hw5-report.pdf
-rwxrwxr-x 1 yun yun         239 Oct 26 04:22 linux-sort.sh
-rw-rw-r-- 1 yun yun         100 Oct 23 14:06 Makefile
-rwxrwxr-x 1 yun yun       16448 Oct 26 03:57 mysort
-rw-rw-r-- 1 yun yun        2185 Oct 26 04:33 mysort.c
-rw-rw-r-- 1 yun yun          88 Oct 26 03:57 mysort.log
-rw-rw-r-- 1 yun yun         404 Oct 23 14:07 README.md
-rwxrwxr-x 1 yun yun         339 Oct 23 14:06 run-mysort.sh
drwxrwxr-x 2 yun yun        4096 Oct 26 04:21 screenshots
-rw-rw-r-- 1 yun yun         124 Oct 26 03:57 valsort.log
yun@yun-ZenBook-UX325EA-UX325EA:~/Desktop/cs553/cs553-fall2022-hw5-yz-msq-bt$ 
```

2. External Sort

- Memory limit: 4GB, Because the test laptop memory is just 7.47GB(8G RAM), and ubuntu system have to use 1.1GB, and if the program running, it also has to use 1GB, so I chose 4GB as the upper limitation of data in memory.
- Multi-thread pools: We created a multi-threaded pool which supports 8 threads(through the use of command line parameters (1, 2, 4, 16)), but we achieved the best performance using 8 threads as shown below.



```
● yun@yun-ZenBook-UX325EA-UX325EA:~/Desktop/cs553/cs553-fall2022-hw5-yz-msq-bt$ time sort -k 1 1GB.txt -o 1GB-sort.txt --parallel=64

  real    0m11.753s
  user    0m39.949s
  sys     0m2.431s
● yun@yun-ZenBook-UX325EA-UX325EA:~/Desktop/cs553/cs553-fall2022-hw5-yz-msq-bt$ time sort -k 1 1GB.txt -o 1GB-sort.txt --parallel=8

  real    0m11.676s
  user    0m39.974s
  sys     0m1.882s
● yun@yun-ZenBook-UX325EA-UX325EA:~/Desktop/cs553/cs553-fall2022-hw5-yz-msq-bt$ time sort -k 1 1GB.txt -o 1GB-sort.txt --parallel=1

  real    0m27.782s
  user    0m25.528s
  sys     0m1.066s
● yun@yun-ZenBook-UX325EA-UX325EA:~/Desktop/cs553/cs553-fall2022-hw5-yz-msq-bt$ time sort -k 1 1GB.txt -o 1GB-sort.txt --parallel=4

  real    0m31.468s
  user    0m30.063s
  sys     0m1.705s
○ yun@yun-ZenBook-UX325EA-UX325EA:~/Desktop/cs553/cs553-fall2022-hw5-yz-msq-bt$ 
```

- Based on the 4GB memory limit, if we need to sort data smaller than 4GB, we don't need to use external sort, we just need to use in-memory sort.
- For external sort:
    - we read the data, split it, and used qsort() to sort the data in memory, and wrote to the temporary file. For example, if we sort a 16GB file using 8 threads, we use:
        - the max memory of each thread - 0.5GB.
        - we have to use (filesize / 0.5GB) temporary files. // since each file is of 0.5 GB, 16 GB will give us 32 temporary files

    Sorting algorithm:

    quick-sort, because if use merge sort, we have to use extra space.
- k-way merge sort: each time read one line from all of the temporary files. and pick up smallest line and write to the output file, then choose the next line on minimum line file.

3. Testing

```
linux test: time sort -k 1 16GB.txt -o 16GB-sort.txt --parallel=8
mysort test: ./mysort 16GB.txt out.txt 8 >> mysort16GB.log
results: mysortXX.log, linsortXX.log
```

4. Suggestions to optimize

1. k-way merge can be implemented using multi-threading: 32 files -> 4 files (using 8 threads) -> 1 file (we have to test multiple times by external read/write performances)
2. qsort() can be implemented by manually instead of using the C library qsort().
3. For k-way merging, we can use Min-Heapify to optimize the performance, rather than picking up the smallest line.
4. Improve the hardware memory limitation (8GB), our laptop supports the threshold of 4GB.

Log files:

The command line output of mysort.c is used for creating log files which contains details about the name of input file, output file, number of threads, execution time.

Experiment	Shared Memory (1 GB)	Linux Sort (1 GB)	Shared Memory (4 GB)	Linux Sort (4 GB)	Shared Memory (16 GB)	Linux Sort (16 GB)	Shared Memory (64 GB)	Linux Sort(64 GB)
Number of Threads	1 or 8	8	1 or 8	8	8	8	8	8
Sort Approach	in-memory	in-memory	in-memory	in-memory	External	External	External	External
Sort Algorithm	Quick sort	Merge sort	Quick sort	Merge sort	Quick sort	Quick sort	Quick sort	Quick sort
Data Read (GB)	0.125 GB per thread(8 thread)	0.125 GB per thread	0.5 GB per thread	0.5 GB per thread	0.5 GB per thread	0.5 GB per thread	0.5 GB per thread	0.5 GB per thread

Data Write (GB)	1 GB (1 thread write)	0.125 GB per thread	0.5 GB per thread	0.5 GB per thread	0.5 GB per thread	0.5 GB per thread	0.5 GB per thread	0.5 GB per thread
Sort Time (sec)	12.105	17.994	73.088	148.82	573.978	614.562	1868.534	1353.937
Overall I/O Throughput (MB/sec)	83.333	93.601	54.728	26.878	27.875	26.034	35.073	47.26
Overall CPU Utilization (%)	99.74%	55.57%	56.532%	112.71%	52.427%	137.66%	76.383%	140.34%
Average Memory Utilization (GB)	1.157GB	0.826GB	4.611GB	0.983G	2.379GB	1.258GB	3.434GB	2.609GB

From table, we know:

1. For small amount of data sorting, that is, 1GB and 4GB, the performance of our in-memory algorithm is worse than the sorting algorithm of linux system, I think it is mainly because some specific code optimization is not done well, but the time complexity is the same.

2. And for large amount of data sorting, that is, 16GB and 64GB, the performance of our external sorting algorithm is better than that of the linux system itself, mainly because we fully use the performance of the machine itself (the test machine itself is a high computing power cpu is i7, but the memory is only 8G machine).

Linux sort benchmarks against different data sizes

```
time sort -k 1 1GB.txt -o 1GB-sort.txt --parallel=8
real    0m17.994s
user    0m42.101s
sys     0m1.665s

time sort -k 1 4GB.txt -o 4GB-sort.txt --parallel=8
real    2m28.820s
user    3m6.515s
sys     0m8.836s

time sort -k 1 16GB.txt -o 16GB-sort.txt --parallel=8
real    10m14.562s
user    13m59.796s
sys     0m31.369s

time sort -k 1 64GB.txt -o 64GB-sort.txt --parallel=8
real    50m2.503s
```

```
user    67m48.721s
sys     2m42.319s
```

<u>mysort benchmarks against different data sizes</u>

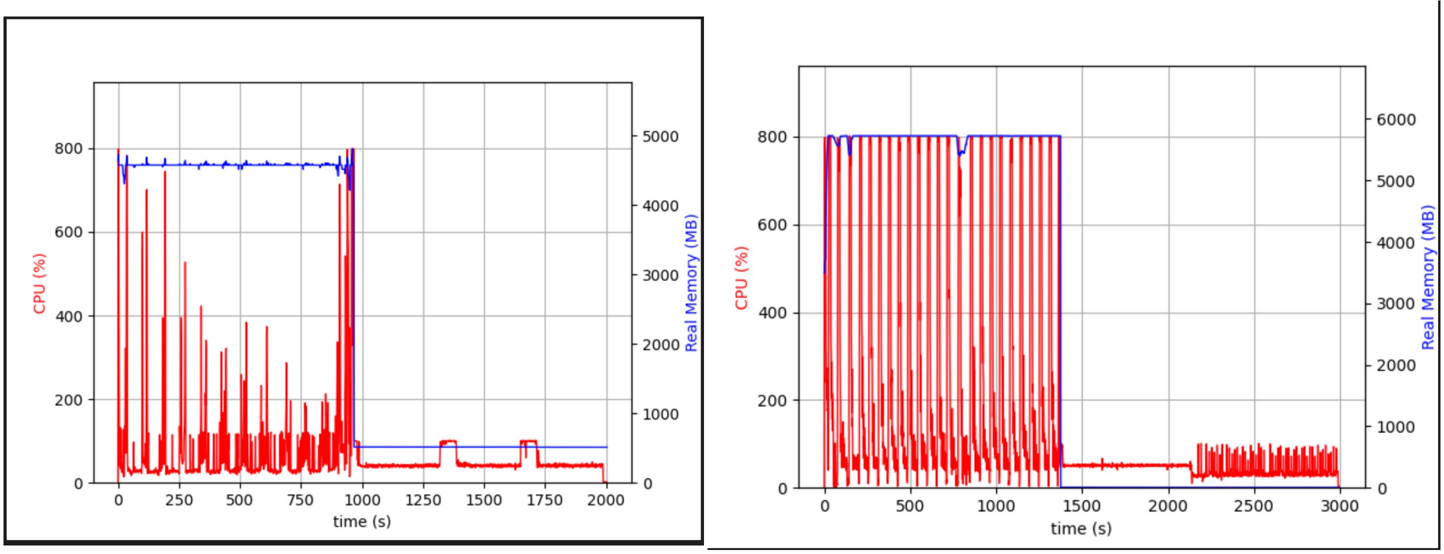
1GB Data file:
input file: 1GB.txt
output file: out.txt
number of threads: 8
execution time: 12.105986

4GB Data file:
input file: 4GB.txt
output file: out.txt
number of threads: 8
execution time: 73.088546

16GB Data file:
input file: 16GB.txt
output file: out.txt
number of threads: 8
execution time: 573.978294

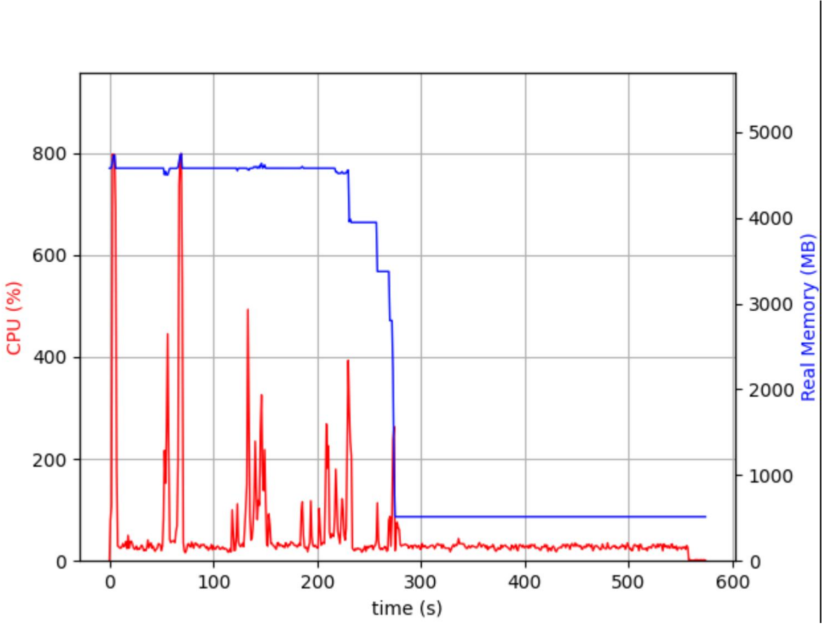
64GB Data file:
input file: 64GB.txt
output file: out.txt
number of threads: 8
execution time: 1868.532462

CPU utilization plots:
For 64GB(mysort) vs 64gb(linux sort)

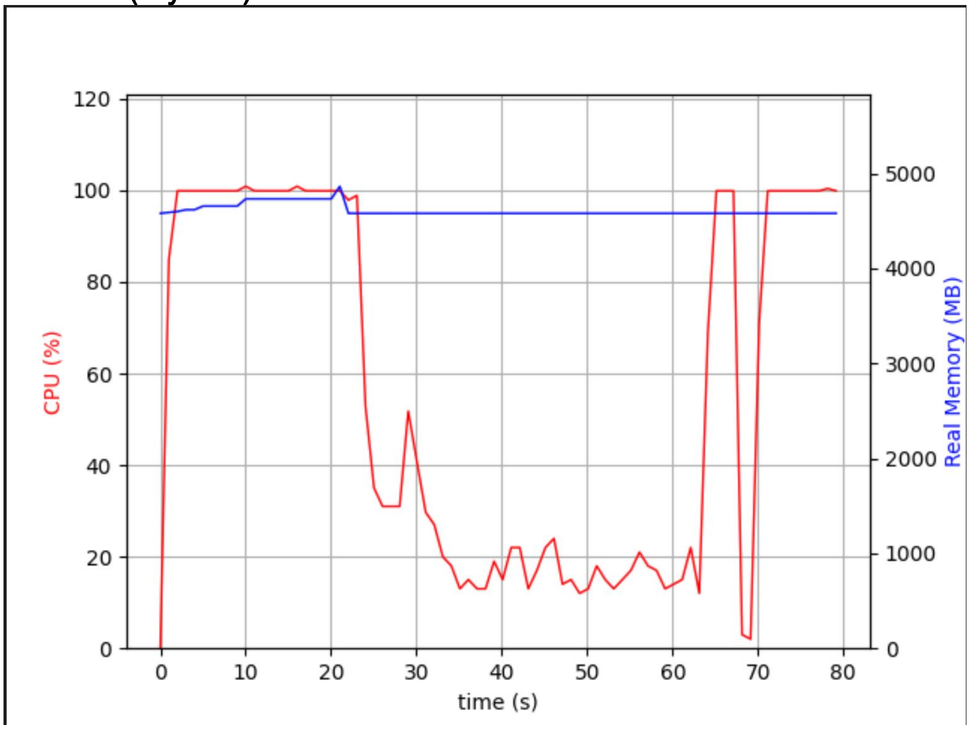


From the graph, the overall CPU usage of linux's own sort is better than our mysort, but the memory usage is not enough, which is why its performance is not as good as mysort's. The bottleneck of external sort lies in the long time to read and write files, if we can optimize this piece, we can improve the performance better.

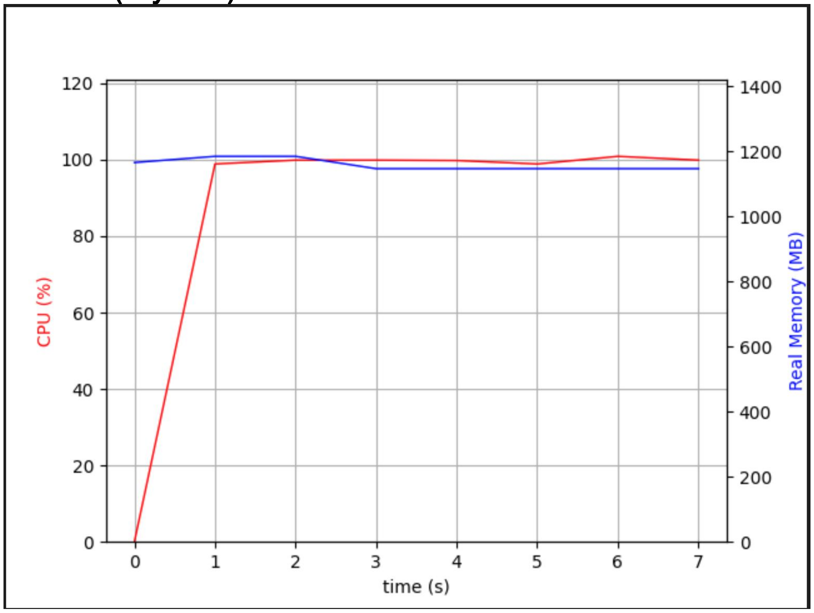
For 16GB(mysort):



For 4GB(mysort):



For 1GB(mysort):


CPU (%)
time (s)
Real Memory (MB)