CSC443 Assignment 1 Part 1&2 Report

Fangzhou Yu (yufangzh)

Yu Xie (xieyu1)

Bar Chart Read Guide:

read\_block\_seq.txt -> read from disk sequentially

read\_block\_rand.txt -> read from disk randomly

read\_ram\_rand.txt -> read from RAM randomly

read\_ram\_seq.txt -> read from RAM sequentially

write\_ram\_rand.txt -> write to RAM randomly

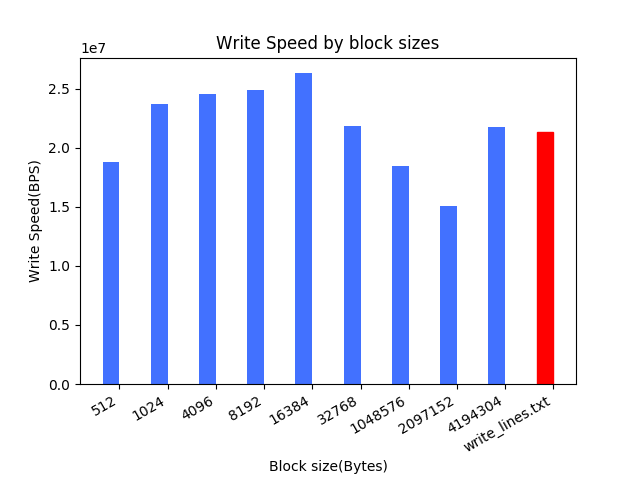
write\_block\_rand.txt -> write to disk randomly

write\_block\_seq\_opt.txt -> write to disk sequentially

Experiment 1



my Linux OS system block size 1024 bytes



The speed for writing in lines is 20.322MBPS.

Q1: Optimal block size to my experiment is?

A1: 16KB

Q2: Does it correspond to the system disk block size?

A2: No

Q3: Is there a block size when further increase does not contribute to better performance?

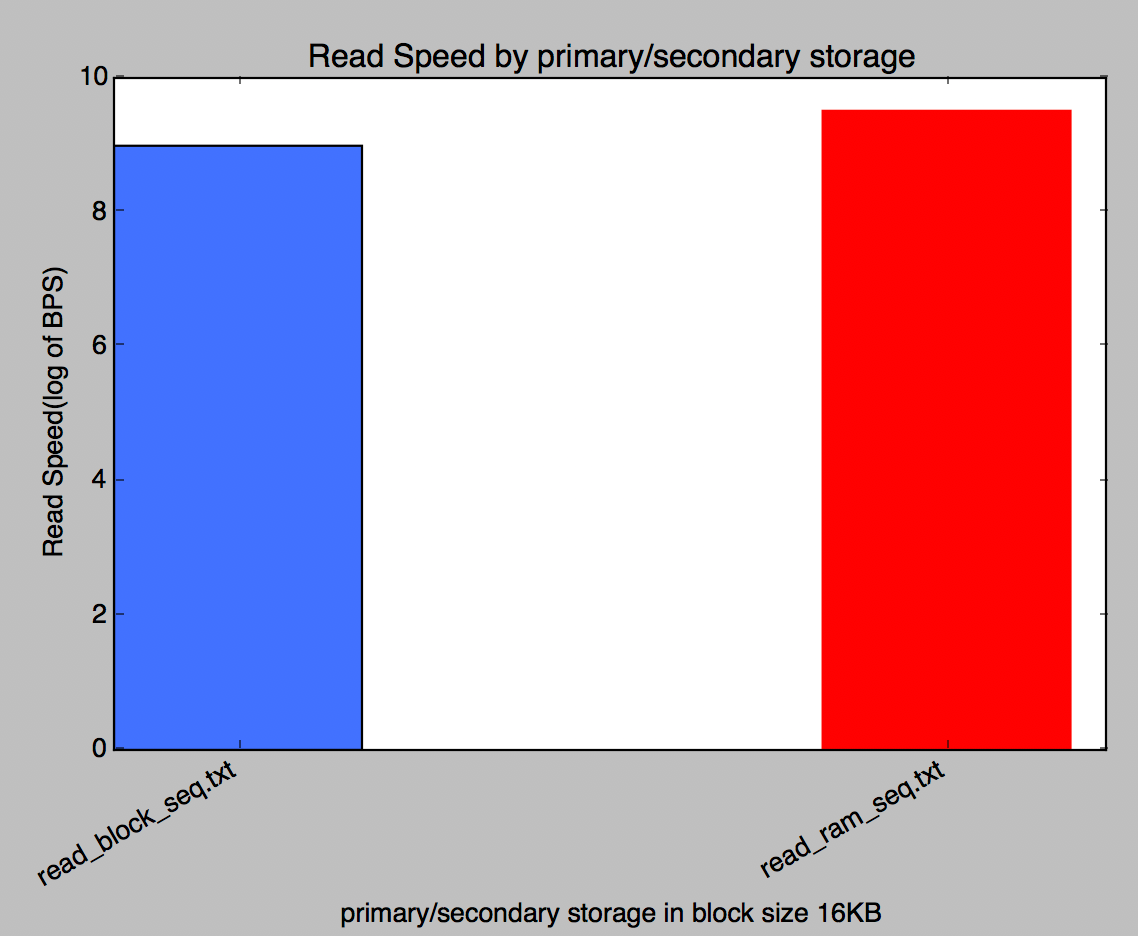
A3: 16KB

Summary of Experiment 1:

As we can see from the bar chart, process writing in lines is slower than process writing in blocks. Process writing in blocks is more efficient. From the lecture, we learned that using block to write data to disk can reduce disk I/O and it will be more efficient. The results we got shows that point, process writing in blocks doing more work in RAM, but process writing in lines has a lot of disk I/O actions.

The optimal block size we got for our experiment is 16KB which is different than the system block size. Because the block size used by OS system is not optimal. 16KB is the better block size to use in this case.

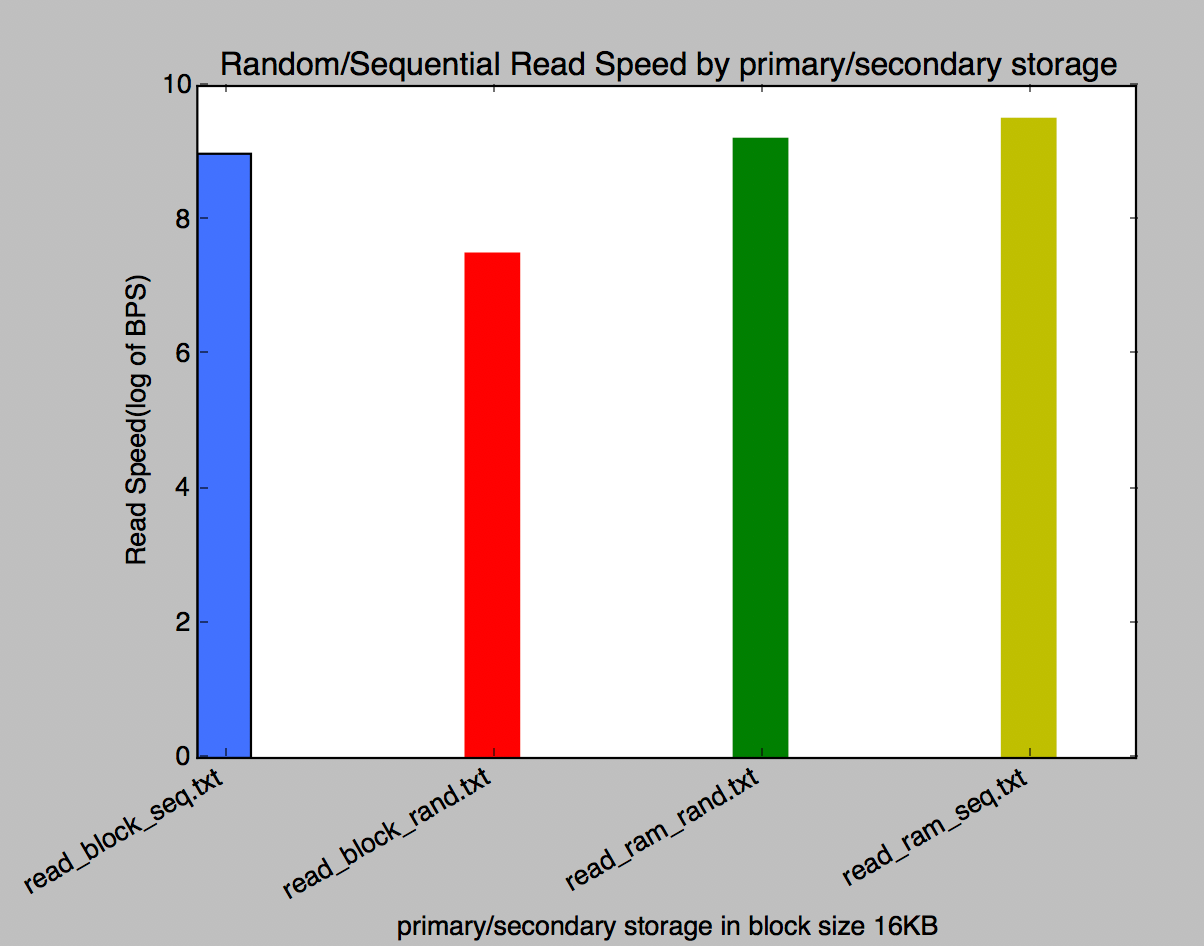
Experiment 2



Q: What is the ratio of sequential read rate for secondary storage and for RAM? Does it correspond to the ratio discussed in class? If not, what do you think is the reason?

A: The ratio discussed in class is around 10^8.8/Sec (sequential RAM) and around 10^7.6/Sec (sequential DISK). The ratio we got from the data is different, around 10^9.1/Sec (sequential DISK) and 10^9.6/Sec (sequential RAM). Reason we come up with is the graph we see in the lecture slides is the result from 2009, now the sequential DISK must be upgrade with better I/O performance.

Part 2



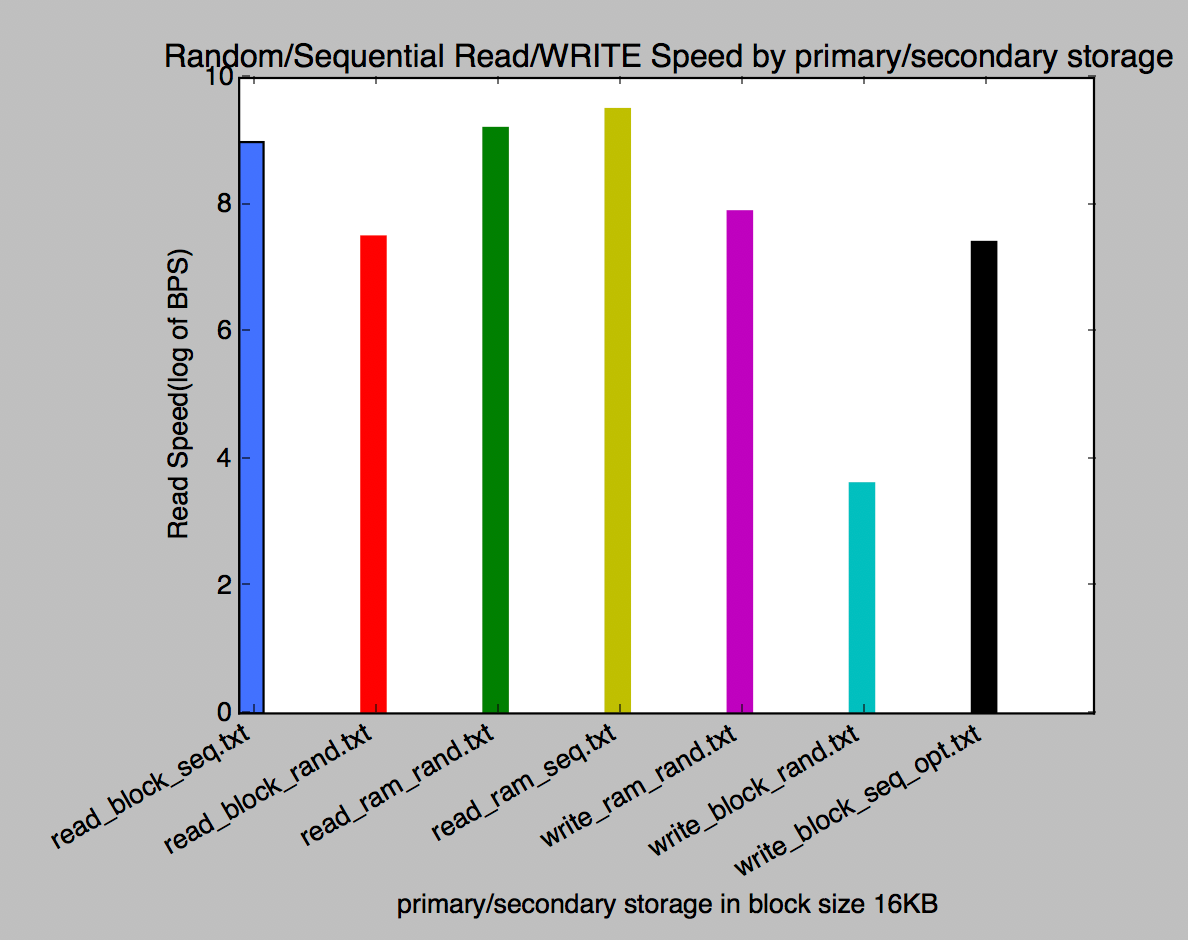
Q: Discuss differences in speed and make a conclusion about reading rates (sequential and random reads) for different memories.

A: Read from RAM sequentially is always the fast one, and then is read from RAM sequentially, read from block sequentially and read from disk randomly. From this we can see read from RAM is always faster than read from disk, discard random read or sequential read. Also, when we read from disk, using sequentially read is much more efficient than random read. Reading from RAM randomly and sequentially are almost the same, we think this is because search data position in RAM also very fast.

Summarize:

The results we got from this experiment is very like what we learned from the lecture. The lecture slides said random disk access need seek time plus rotational delay and transfer delay to get data, but sequential disk access only has transfer delay. That states why our read block rand process has very slow read speed compare the rest.

Experiment 3



From the bar chart, we can clearly see that read is always faster than write. Write to RAM randomly is faster than write to disk randomly. Write to disk sequentially is more efficient than randomly.

**Summary:**

The access pattern for different memory type is:

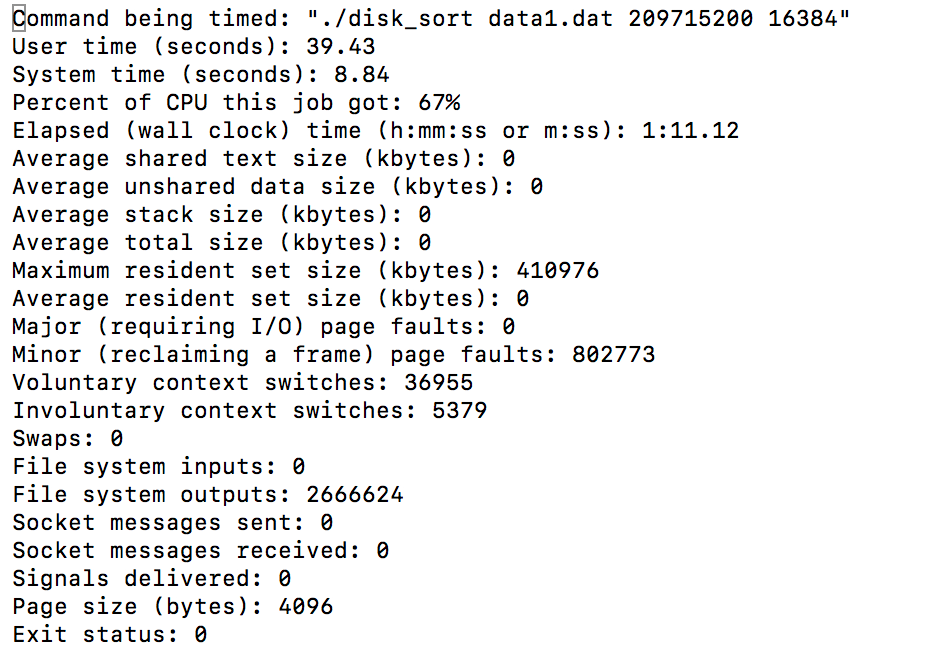
1. data access from RAM is faster than data access from disk,
2. Sequential read/write is always faster than random read/write for disk memory type
3. Sequential read and random read are not very different in RAM.
4. Read is faster than write in both RAM and disk.

Overall the results of those experiments do persuade us that we need to design different algorithms for primary and for secondary storage.

Because for primary storage, we want an algorithm that access the disk as less as possible. Disk I/O is very expensive from the results we got for those experiments. But when access data in RAM, we can access sequentially or randomly. The results from the experiments show there is not a huge difference between sequential read and random read from RAM. For secondary storage, we want algorithms that access data sequentially, random access is very slow in disk.

A1.2

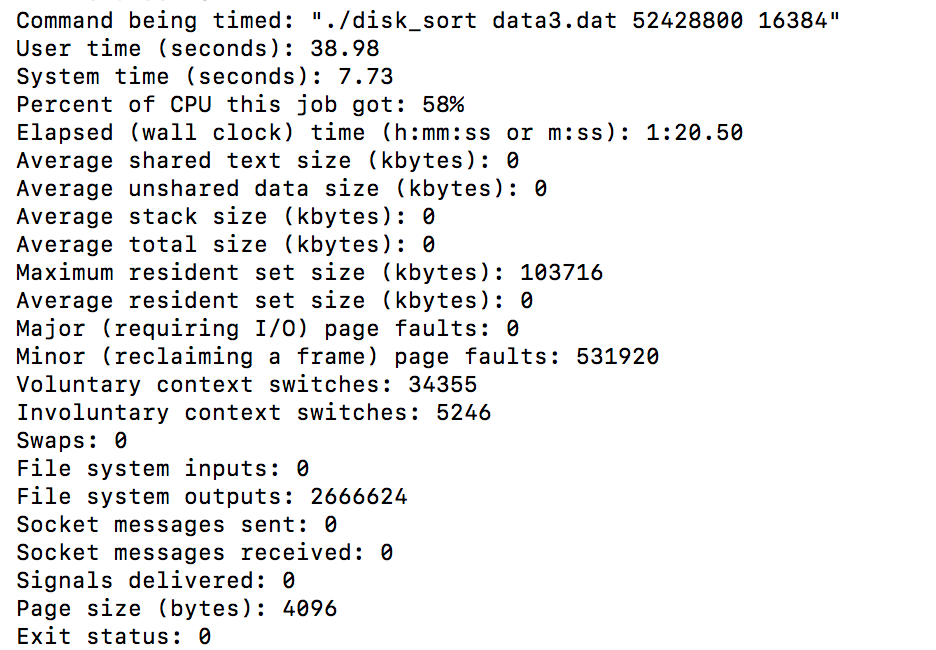
2.1:

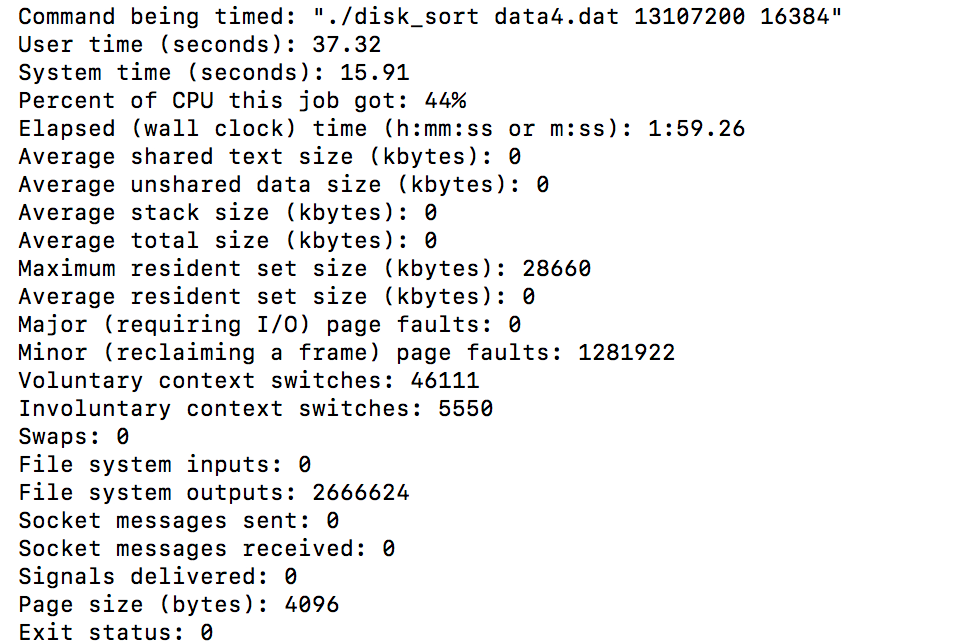


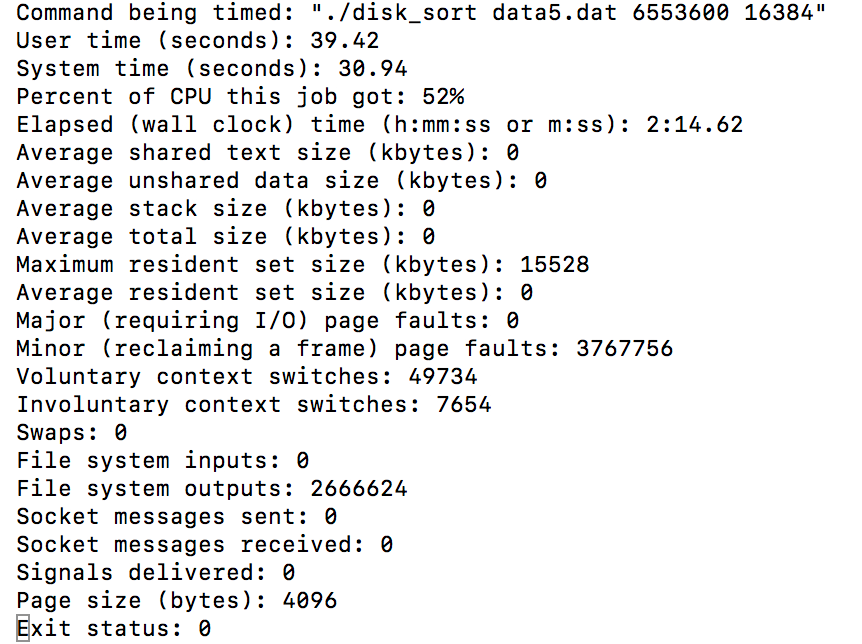
2.2:

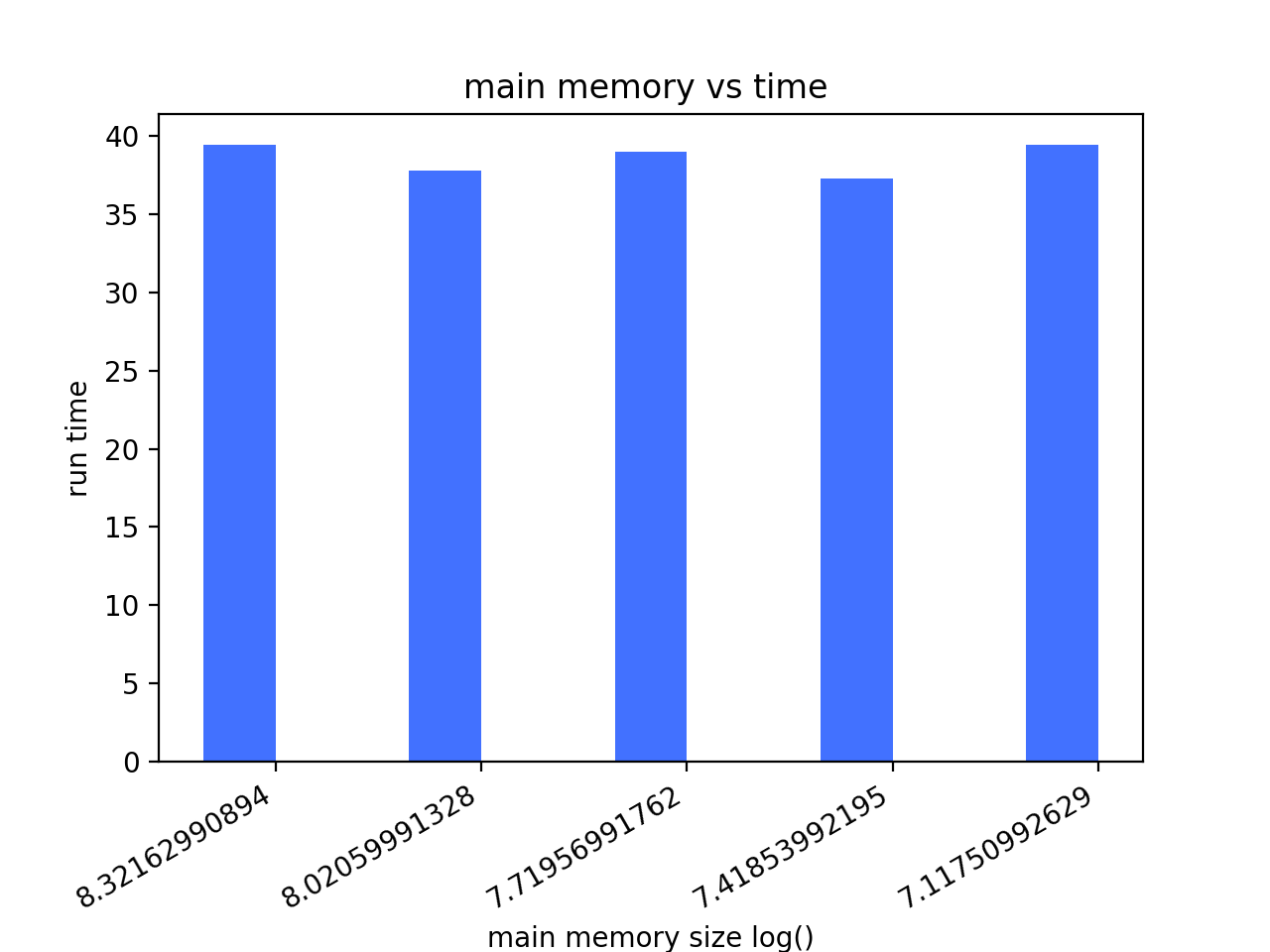
Results from the experiment







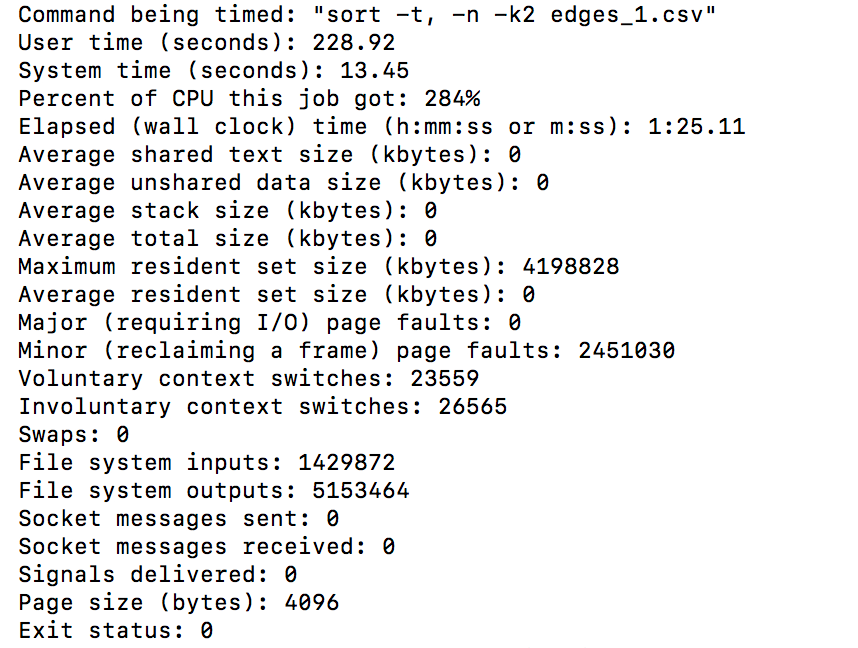




Q: Is there any difference in performance in your experiments? Explain why there is a difference or why there is no difference.

A: There is no difference in performance. All the run time we got are very similar. Because the block size we use is fixed and we are sorting from the same data. Each block is read from disk twice and write to disk twice. Then the running time is not depended on the number of runs, and the size of the input and output buffers.

2.3



Q: Which program is faster: your implementation or Unix sort? Which one uses less memory? Explain the difference (or the lack of difference) in performance. If there is a difference - what in your opinion could explain it?

A: Our implementation is faster than Unix sort. And our implementation uses less memory. The main difference we can see from Unix sort and ours is the memory use. Also, Unix sort uses a large amount of CPU. We notice that there is a parameter –k to tell Unix which column in the file to sort, maybe every time Unix needs look up this column first. But in our implementation, we turn everything into binary first. Then perform the sort, no need to do look up for the sorting key every time.

Summary:

For 2PMMS, the total available main memory would not affect the performance of sorting if the total available main memory is large enough to perform 2PMMS.

After compare to Unix sort, we found out that there are other situations that might affect 2PMMS performance. We learned that Unix sort also using 2PMMS, but it has worse performance than ours. We thought the type of data might also affect the performance. Our implementation is sorting on binary data, which is quicker to read. But Unix sort is sorting on CSV file, it needs to find the sorting key first before each comparison. Also, I tried to let Unix sorting buffer a little bit data in memory, which improves the performance.