Assignment 3

Will Gantt and Hannah An

April 24, 2020

1 Overview

The following sections describe the experiments we ran using our `msort` and `bsort` program. The provided test schema was used across all experiments. Details of our implementation can be found in the `README` in our submission.

2 Msort Experiments

2.1 File Size Experiments

In this first set of experiments, we fixed k to 10 and the allowed memory to 2860 bytes. The latter number was chosen because:

1. The length of a record in memory is 26 bytes—25 bytes collectively for the attribute values and an additional byte for the null-terminating character in the string.
2. We wanted each buffer to be able to hold 10 records. With $k = 10$ input buffers, this requires $26 * 10 * 10 = 2600$ bytes, plus an additional 260 bytes for the output buffer, which brings the total requirement to 2860.

We varied input file sizes from 1000 records ($\approx$ 30KB) to 200,000,000 ($\approx$ 5.6GB) and sorted records on the `cgpa` attribute. The experiments were run locally on Will's laptop (a 2018 MacBook Pro). The specific file sizes and their runtimes (in real time) are shown in Table 1 and Figure 1 below.

File size (# records)	Runtime (seconds)
10^3	0.03
10^4	0.3
10^5	3.12
10^6	33.49
10^7	366.82
10^8	—
$2 * 10^8$	—

Table 1: `msort` runtime by file size.

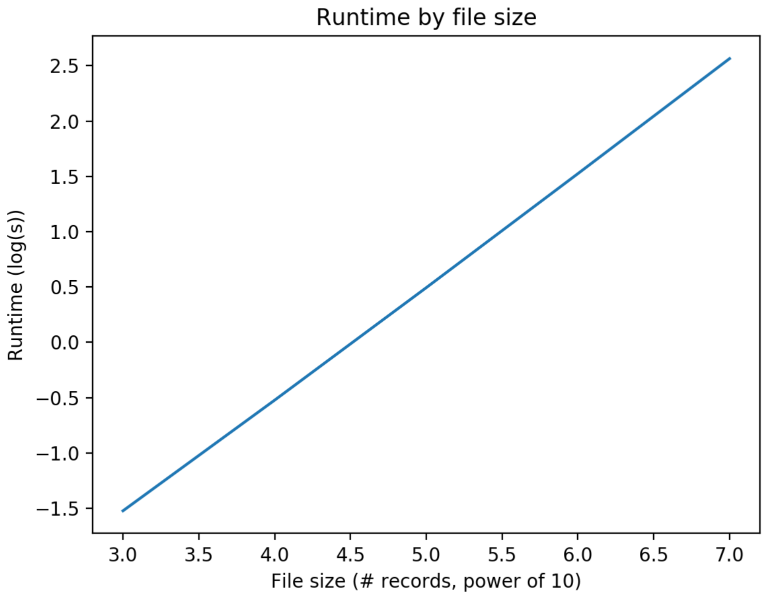

Figure 1: `msort` runtime by file size. Note that the y-axis is log-transformed to account for the exponential steps on the x-axis.

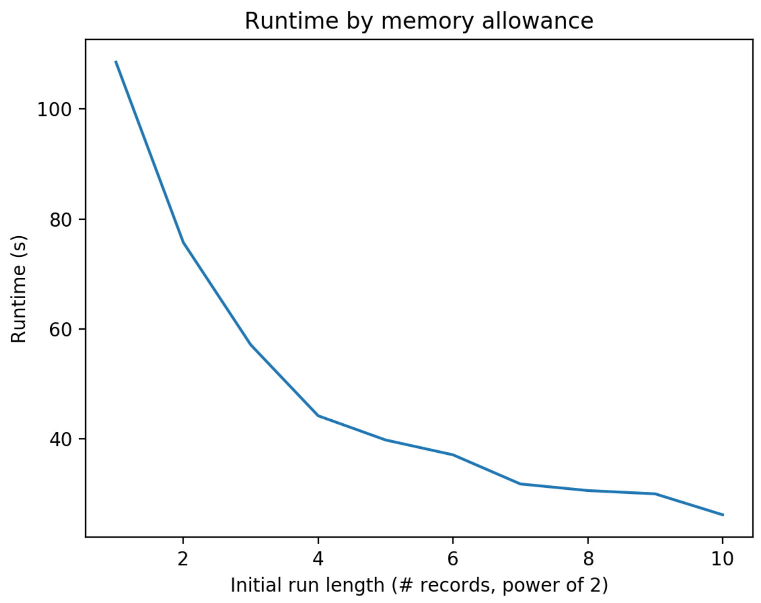

Figure 2: `msort` runtime by memory capacity, as measured by initial run length.

In retrospect, 10 was probably a needlessly small value to use for k. However, we nonetheless observed an almost perfectly linear relationship between file size and program runtime, as evidenced in Table 1. This makes sense, given that we know from class that the runtime of 2PMMS is $3B$ disk I/Os (or $4B$, counting the final write), where B is the size of the file, which is plainly linear. Unfortunately, our program exited with an error roughly 25 minutes into the runs for files of size 10^8 and $2 * 10^8$ bytes, and so we were not able to collect data points for them. We realized that 2860 bytes is too little memory to enable sorting of a file of that size, but even after retrying with memory of 28600 bytes and $k = 100$, we still encountered the same problem. Unfortunately, we were unable to identify the source of the error before the submission deadline.

2.2 Memory Allowance Experiments

For our second experiment, we considered different allowances for total memory. We fixed file size to 1 million records, k to 10, and sorted on the `cgpa` attribute. We allowed memory capacity to vary such that the first pass run length varied from 2 to 1024 by powers of 2. Thus, the memory requirement exactly doubles between experiments. The results are shown in Table 2 and Figure 2.

With the file size and k fixed, the only thing that the memory capacity changes is the number of records that may be stored in each buffer, which in turn reduces the frequency with which the output buffer must be flushed to disk and also may reduce the number of passes required. Unsurprisingly, we observe significant performance gains as the memory capacity is increased, and particularly as fewer passes are needed.

Memory capacity (KB)	Initial run length	Passes required	Runtime (seconds)
0.572	2^1	6	108.5
1.144	2^2	6	75.7
2.288	2^3	6	57.1
4.576	2^4	5	44.2
9.152	2^5	5	39.8
18.304	2^6	5	37.1
36.608	2^7	4	31.8
73.216	2^8	4	30.6
146.432	2^9	4	30.0
292.864	2^{10}	3	26.2

Table 2: `msort` runtime by memory capacity.

2.3 Buffer Number Experiments

In this final experiment, we evaluated the impact of the choice of k on `msort`'s performance. We fixed file size to 1 million records, memory capacity to 26260B (allowing for a buffer of 10 records when $k = 100$) and once again sorted on the `cgpa` attribute. We varied k from 0 to 100 in increments of 5. The results in Table 3 (and plotted in Figure 3) are rather unexpected. For fixed memory capacity M and for a number of input buffers k, we know from class that the cost of 2PMMS can be expressed as $2M(\lceil \log_k M \rceil + 1)$. Recall the rule of logarithms that states that:

$$\log_b(x) = \frac{\log_c(x)}{\log_c(b)}$$

for any bases b and c. We can therefore re-express the runtime using some alternative base (say, 2) as follows:

$$2M\left(\left\lceil \frac{\log_2 M}{\log_2 k} \right\rceil + 1\right)$$

Thus, we would expect runtime to decrease logarithmically in k, which is not clearly born out in our data. However, we ran `msort` just once for each value of k, and it's possible that are results are somewhat noisy.

3 Bsort Experiments

3.1 Sorting Comparison Experiments

In this experiment, we compared the performance of `msort` against `bsort`. To evaluate different ways of sorting, we used `bsort`, and `msort` with a poorly chosen parameter as well as a well-tuned parameters. For the poorly chosen parameter for `msort`, we used $k = 2$ and 572 bytes of the allowed memory. For the well-tuned parameter for `msort`, and fixed k to 100 and allowed 28600 bytes of memory capacity. For all three aforementioned ways of sorting, we also varied input file sizes from 1000 records ($\approx$ 30KB) to $10,000,000$ ($\approx$ 300MB) based on the results from Section 2.1. The specific file sizes and their runtimes (in real time) of different ways of sorting are shown in Table 4

k	Runtime (seconds)
5	41.9
10	43.6
15	32.5
20	31.6
25	32.0
30	34.8
35	29.4
40	30.5
45	29.9
50	31.3
55	32.9
60	32.5
65	30.4
70	31.2
75	32.4
80	34.0
85	34.3
90	35.9
95	36.3
100	35.6

Table 3: `msort` runtime by value of k.

and Figure 4 below.

`bsort` always had better performance than `msort`. As Table 4 shows, `bsort` ran faster than `msort` with all the various file we tested and was able to handle larger files that `msort` could not. It wasn't surprising that `msort` with a poorly-chosen parameter terminated with an error for the files with more than 1M records. However, it was `msort` with a well-tuned parameter that showed unexpectedly poor performance. Even with $k = 100$ and 28600 bytes of the allowed memory, `msort` also stopped after around 1M records, which is worse than what `msort` with $k = 10$ and allowed memory 2860 bytes could handle as we reported in Section 2.1. Unfortunately, we did not have time to analyze and fix the error on the larger files.

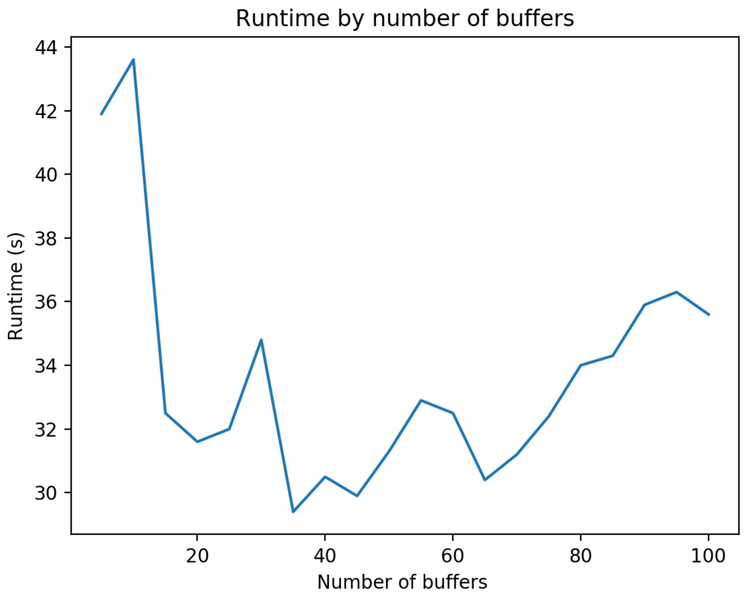

Figure 3: `msort` runtime by value of k

File size	Runtime (seconds)		
(# records)	bsort	msort poorly chosen	msort well-tuned
10^3	0.056	0.072	0.014
$10^{3.5}$	0.059	0.244	0.056
10^4	0.086	0.878	0.183
$10^{4.5}$	0.186	3.382	0.607
10^5	0.484	11.288	1.950
$10^{5.5}$	1.389	40.537	8.409
10^6	4.303	134.523	25.199
$10^{6.5}$	12.301	—	—
10^7	40.173	—	—

Table 4: `bsort` vs. `msort` runtime by file size.

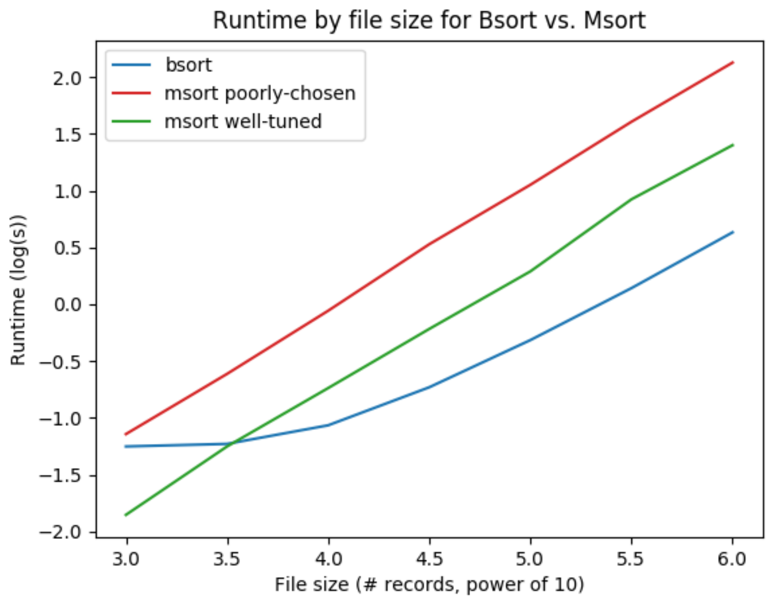

Figure 4: `bsort` vs. `msort` runtime by file size. Note that the y-axis is log-transformed to account for the exponential steps on the x-axis.