2022-2023 Spring

CS 342 – Operating Systems

Project #1 — Report

Berk Çakar - 22003021 - Section 2

Kutay Tire - 22001787 - Section 3

Contents

Introduction

In this report, the total execution times of the programs `proctopk` and `threadtopk` were investigated under various N and K values. Six tests were conducted for each program (3 for fixed N values and 3 for fixed K values) to comprehend the effects of different N and K values.

As our test input source, *The Iliad* was used from the Project Gutenberg digital library [1]. We selected the first 200000 words from the book and split them into ten text files (since the maximum value for N is stated as 10), where each file contains 20000 words. Furthermore, in the Ubuntu Desktop 64-bit 22.04 testing environment, we disabled the file system caching using a command-line tool to obtain better results as it was introducing more bias than expected (i.e., the first run for a particular test case always takes significantly more time, and the remaining runs take very little due to caching) [2]. Overall, the results of the tests were as anticipated, and the graphs successfully reflected the desired outcomes with some minor deviations.

Part 1 - Testing `proctopk`

a) Testing with various N values

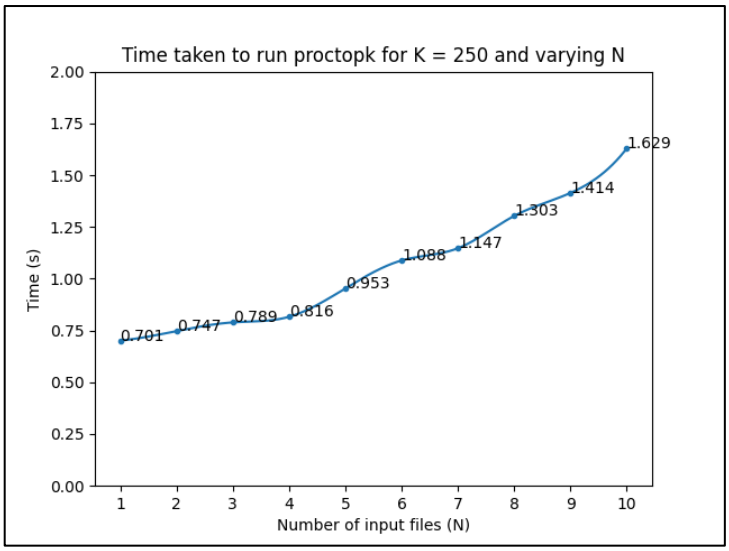

Figure 1: Relationship between `proctopk` execution time and number of input files for K = 250

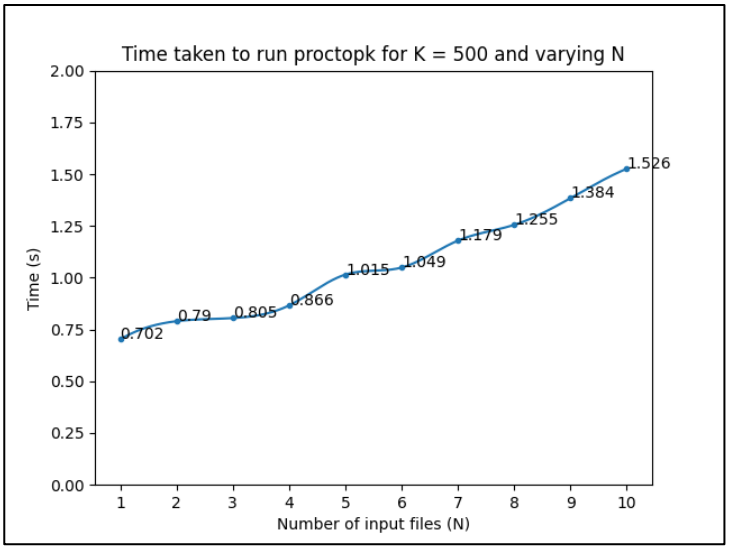

Figure 2: Relationship between `proctopk` execution time and number of input files for K = 500

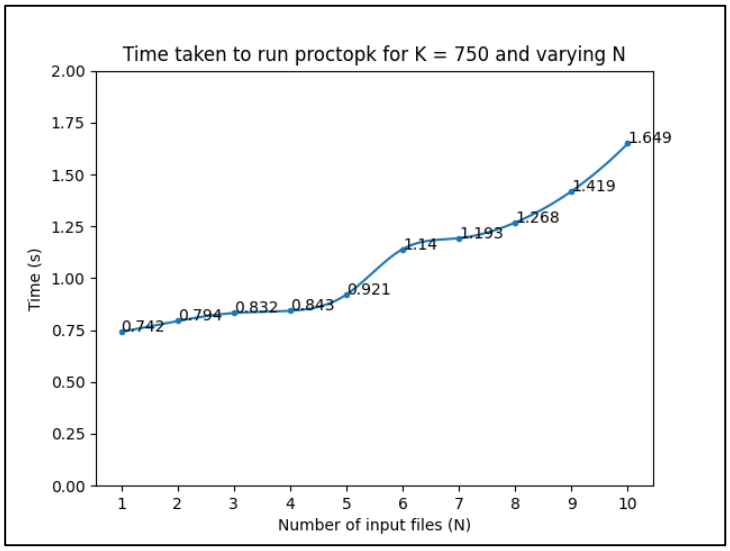

Figure 3: Relationship between `proctopk` execution time and number of input files for K = 750

In the first section of the first part, three different tests were conducted with fixed K values. As can also be seen from the graphs, 250, 500, and 750 were used as the fixed K values, whereas the number of files (N) ranged from 1 to 10. The graphs demonstrate a linear correlation between the time in seconds and the number of files. This is because for each file, the parent process creates a new child process, and the parent process has to wait until all child processes terminate. Therefore, as the N values increase, total execution time also increases.

b) Testing with various K values

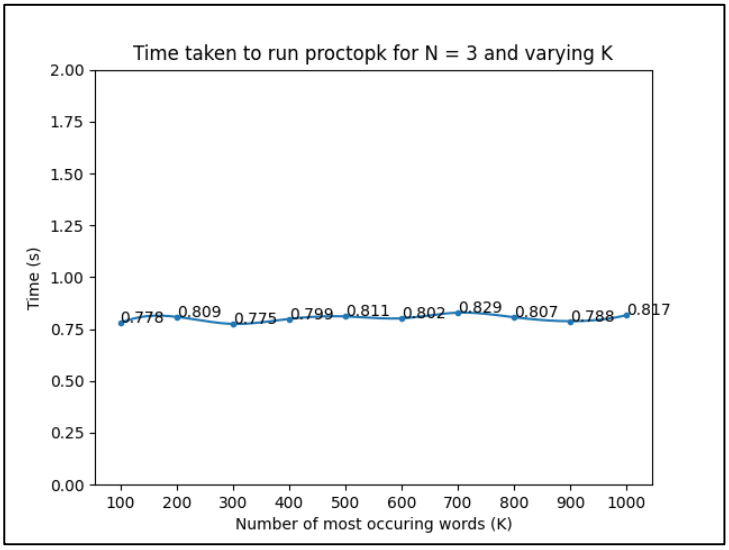

Figure 4: Relationship between `proctopk` execution time and number of most occuring words for N = 3

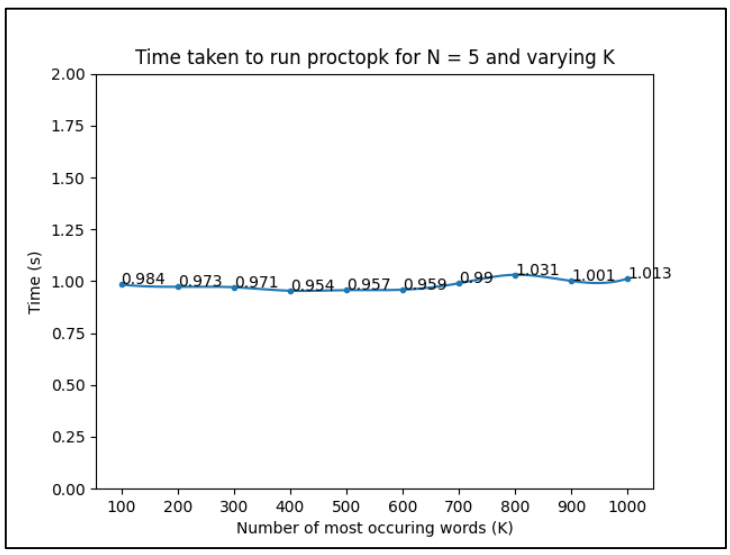

Figure 5: Relationship between `proctopk` execution time and number of most occuring words for N = 5

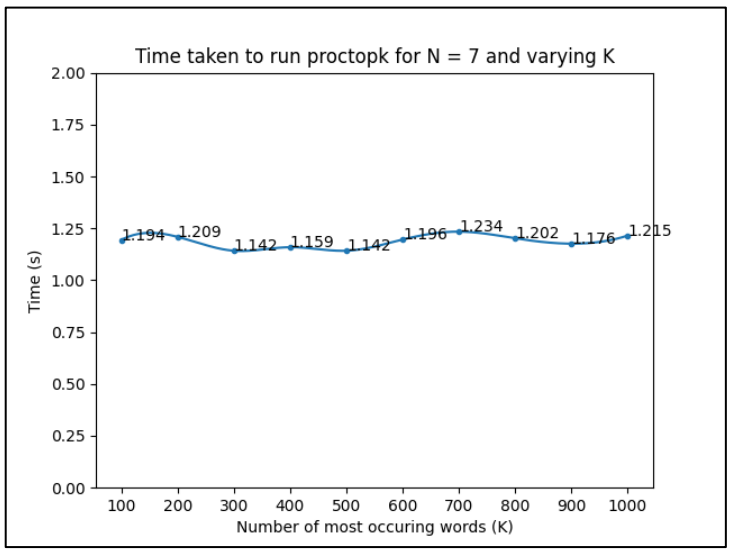

Figure 6: Relationship between `proctopk` execution time and number of most occuring words for N = 7

In parallel with the previous section, three different tests were conducted with fixed N values. Notice that we chose ten equally spaced K values between 100 and 1000, both inclusive. Unlike the graphs of part (a), the graphs were not linearly increasing this time. Although there are some minor fluctuations, it can be observed that the total execution time tends to be the same for different K values for a fixed number of N files. This is relevant to the implementation of the `proctopk` program. Before fetching the top-K words, the table that holds the words with corresponding frequencies is sorted, and the same sorting algorithm is used regardless of the K values. Hence, fetching those K values from the sorted table becomes the only difference-making factor. As the cost of fetching data is relatively small compared to sorting, total execution times for varying K values are pretty close to each other.

Part 2 - Testing `threadtopk`

a) Testing with various N values

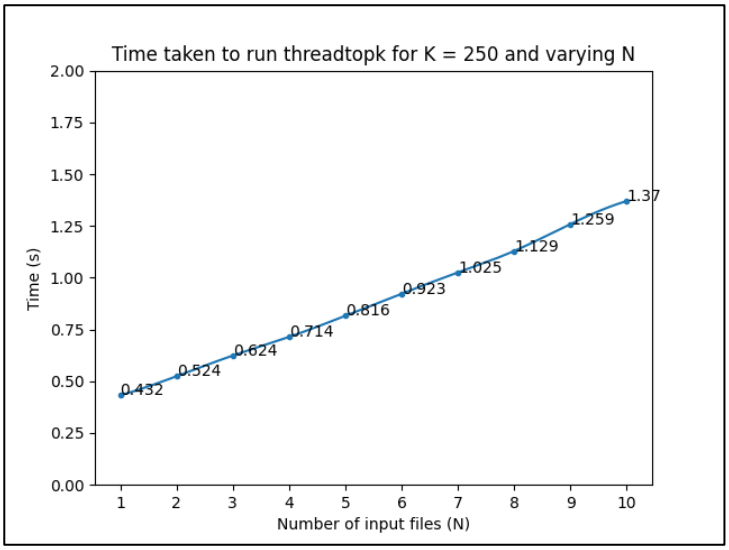

Figure 7: Relationship between `threadtopk` execution time and number of input files for K = 250

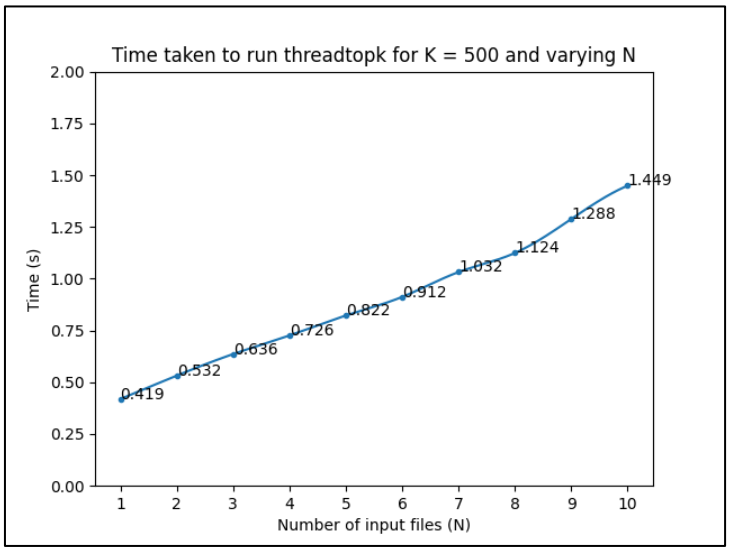

Figure 8: Relationship between `threadtopk` execution time and number of input files for K = 500

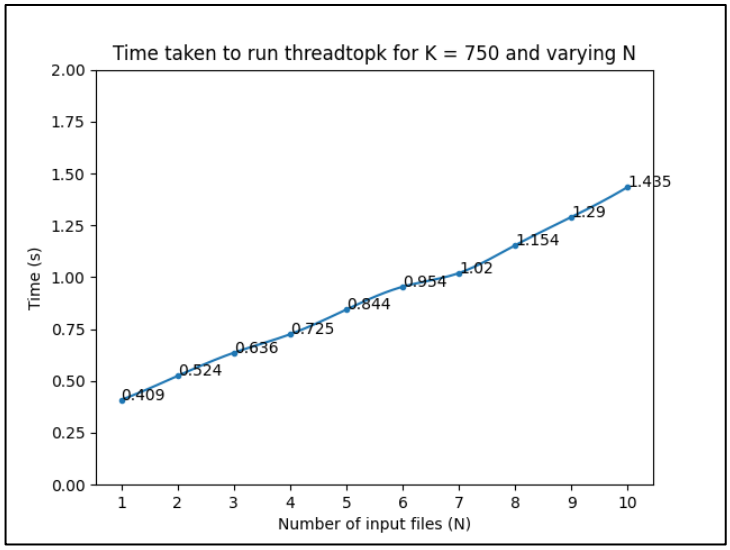

Figure 9: Relationship between `threadtopk` execution time and number of input files for K = 750

As can be seen from the graphs, there is a positive correlation between the number of input files and total execution times for fixed K values. Since `threadtopk` waits for more threads to finish as N goes from 1 to 10, the execution time increases. Notice that for each file, we create a new thread for processing that file. However, it is also clear that the execution times of `threadtopk` are less than the execution times of `proctopk` for the same fixed K values. This is because process creations and executions require more time and space, whereas thread creations and executions are handled in the main process. This enables a thread to process a file and execute the code at a shorter time than a forked child process which can also be verified by comparing the corresponding execution time graphs for `threadtopk` and `proctopk`.

b) Testing with various K values

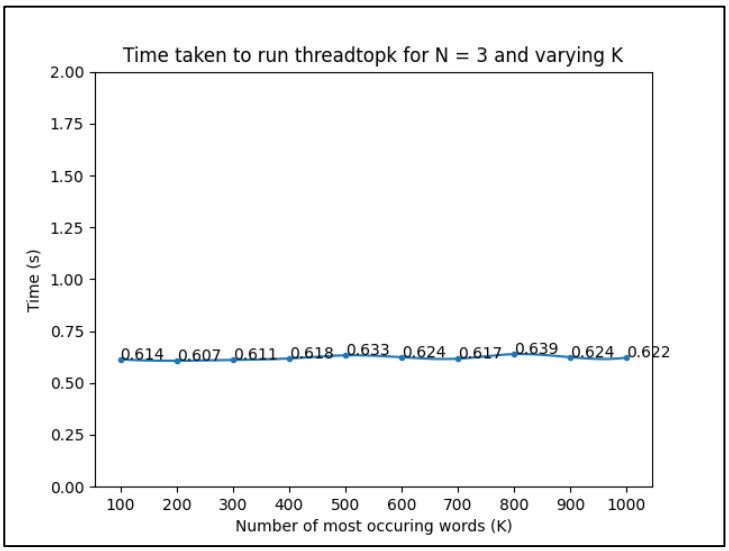

Figure 10: Relationship between `threadtopk` execution time and number of most occuring words for N = 3

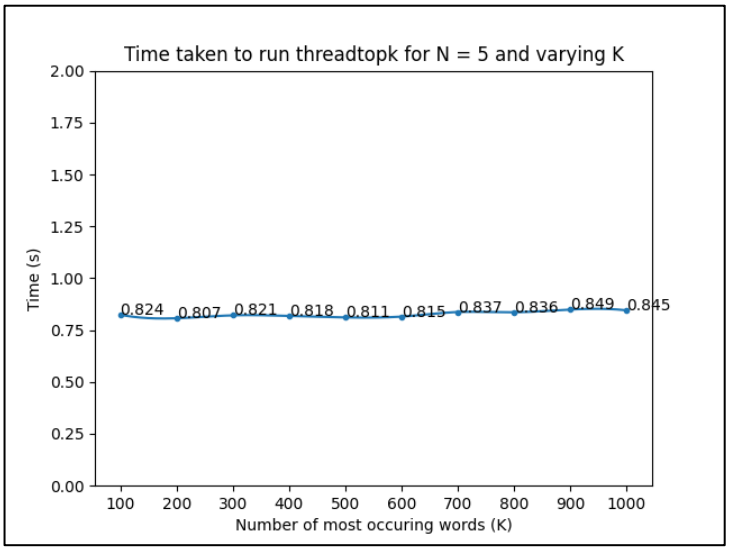

Figure 11: Relationship between `threadtopk` execution time and number of most occuring words for N = 5

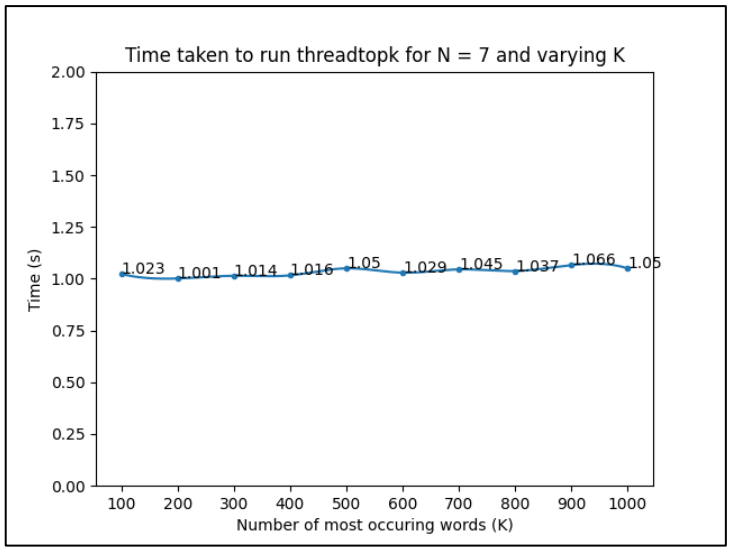

Figure 12: Relationship between `threadtopk` execution time and number of most occuring words for N = 7

In part (b), the execution times with various K values were measured when N was fixed to 3, 5, and 7, respectively. Similar to the case of `proctopk`, the execution times were quite close to each other with little fluctuations. Again, this was due to the reason that the same sorting algorithm was used by each thread regardless of the K value. The only difference-maker was reading the values from the sorted table, which had minimal effect on the total time. Other than that, it is also apparent that `threadtopk` takes less time to run compared to `proctopk` when we test with various K values. Again, as explained before, since threads cause less overhead on creation and execution, we are observing this difference.

Data Tables

To summarize our experiments, the tables below review the results of our tests as a whole.

`proctopk` execution times for various N values (in seconds)	N = 1	N = 2	N = 3	N = 4	N = 5	N = 6	N = 7	N = 8	N = 9	N = 10
K = 250	0.701	0.747	0.789	0.816	0.953	1.088	1.147	1.303	1.414	1.629
K = 500	0.702	0.79	0.805	0.866	1.015	1.049	1.179	1.255	1.384	1.526
K = 750	0.742	0.794	0.832	0.843	0.921	1.14	1.193	1.268	1.419	1.649

Table 1: The table of `proctopk` execution times under various N values

`proctopk` execution times for various K values (in seconds)	K = 100	K = 200	K = 300	K = 400	K = 500	K = 600	K = 700	K = 800	K = 900	K = 1000
N = 3	0.778	0.809	0.775	0.799	0.811	0.802	0.829	0.807	0.788	0.817
N = 5	0.984	0.973	0.971	0.954	0.957	0.959	0.99	1.031	1.001	1.013
N = 7	1.194	1.209	1.142	1.159	1.142	1.196	1.234	1.202	1.176	1.215

Table 2: The table of `proctopk` execution times under various K values

`threadtopk` execution times for various N values (in seconds)	N = 1	N = 2	N = 3	N = 4	N = 5	N = 6	N = 7	N = 8	N = 9	N = 10
K = 250	0.432	0.524	0.624	0.714	0.816	0.923	1.025	1.129	1.259	1.37
K = 500	0.419	0.532	0.636	0.726	0.822	0.912	1.032	1.124	1.288	1.449
K = 750	0.409	0.524	0.636	0.725	0.844	0.954	1.02	1.154	1.29	1.435

Table 3: The table of `threadtopk` execution times under various N values

`threadtopk` execution times for various K values (in seconds)	K = 100	K = 200	K = 300	K = 400	K = 500	K = 600	K = 700	K = 800	K = 900	K = 1000
N = 3	0.614	0.607	0.611	0.618	0.633	0.624	0.617	0.639	0.624	0.622
N = 5	0.824	0.807	0.821	0.818	0.811	0.815	0.837	0.836	0.849	0.845
N = 7	1.023	1.001	1.014	1.016	1.05	1.029	1.045	1.037	1.066	1.05

Table 4: The table of `threadtopk` execution times under various K values

References

[1] Homer, *The Iliad*. [Online]. Available: https://www.gutenberg.org/ebooks/6130. [Accessed: Mar. 18, 2023]

[2] J. Plenz, “nocache.” Mar. 14, 2023 [Online]. Available: https://github.com/Feh/nocache. [Accessed: Mar. 18, 2023]