COL334 Assignment 3

Tript Sudhakar (2021CS10110)
Parth Patel (2021CS10550)

October 15, 2023

§1 Logic of the client program

§1.1 High level overview and design

- We have designed the program using multi-threading (two threads).
- First thread (sending thread) handles sending requests. We are sending one request per `sleep_time`. Second thread (receiving thread) handles receiving data.
- We first wait for the second thread to complete (i.e. receive all the data) and only after this, is the first thread finished.
- At the end, the client sends the MD5 hash value for the complete data received and we get the appropriate feedback from the server.

§1.2 Receiving data reliably

- Whenever the sending thread sends too much requests (even though the leaky bucket had no tokens left), the receiving thread will receive squished data. Whenever this happens we have ensured to increment the `sleep_time` by `0.005s`
- Our program is correct (ensures correctness) and robust. We have NOT used locks since there is no shared resource being modified by both the threads. This ensures no weird behaviour for the multi-threaded application.

§2 Code structure and implementation

§2.1 Basic constructs

- Commands like `send_size`, `submit`, `message` and variables like `size`, `sleep_time`, `lines_recv`, `index`, `lock`, `datastring`, `hashval`, `bytestr`, `feedback`
- Data structures : `offset_list` list, `received` list, `data_list` list

§2.2 Receiving and Sending Threads

- The threads `sending_thread` and `receiving_thread` handle concurrent communication with the server, associated with the functions `sending_process` and `receiving_process`

- `receiving_process` receives the messages from the server, processes it and stores it in `data_list` list. If the message contains "squished", it increases the `sleep_time` by `0.005s`
- `sending_process` sends requests to the server and waits for `sleep_time` amount of time after each request. In case of congestion (leaky bucket), the `sleep_time` will be incremented by `receiving_process`, hence ensuring robustness.

§2.3 Main program

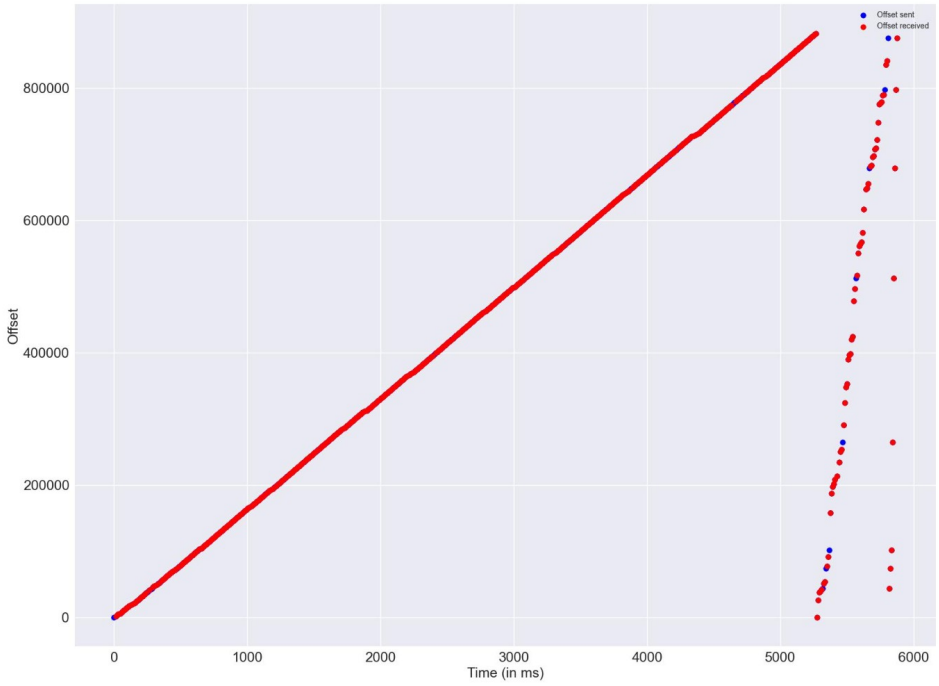
- The order of execution of threads is
  1. `sending_thread.start()`
  2. `receiving_thread.start()`
  3. `receiving_thread.join()`
  4. `sending_thread.join()`
- No locks are required because there are no resources which are being modified by both the thread.
- The main program finally constructs the final string and computes its MD5 hash value to and submits that to the server.

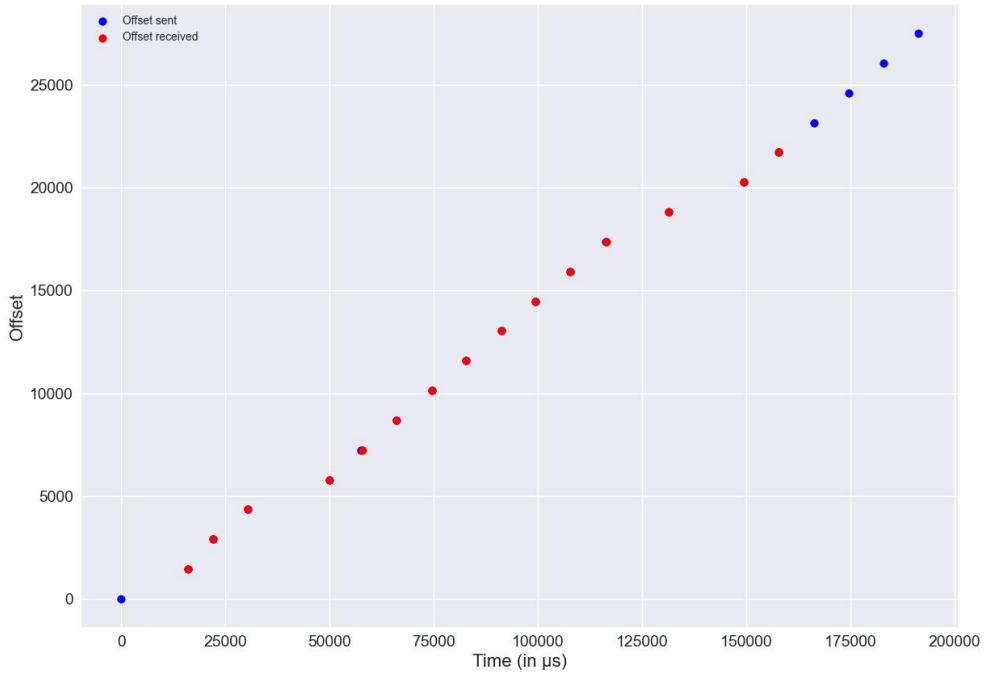
§3 Results and Analysis

We performed the analysis for 3 possible values of total number of lines - 10000, 50000, 100000.
We have assumed that the server is constant rate for now, even if its not, we

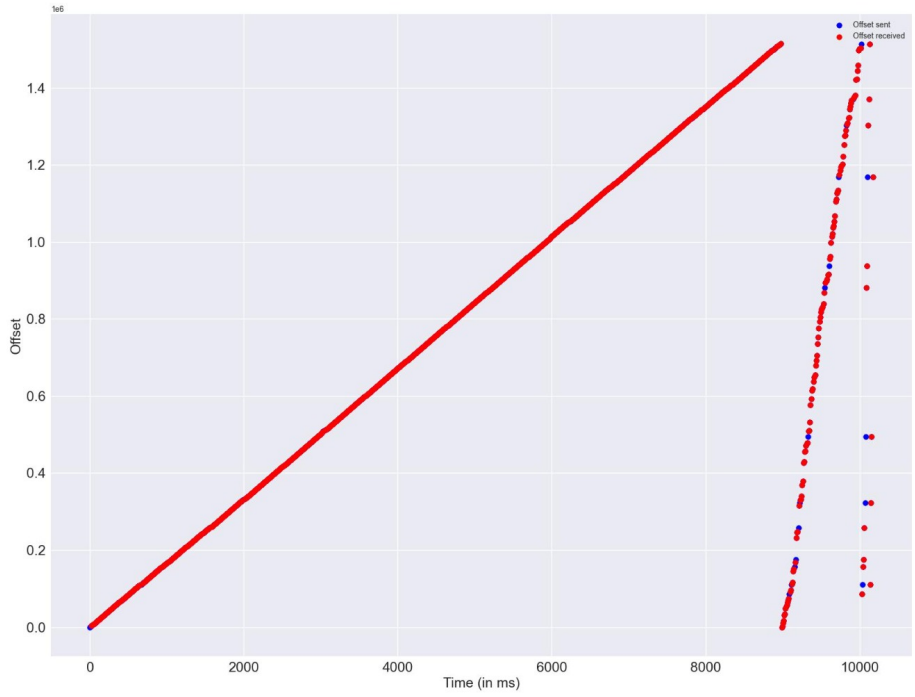
§3.1 Sequence-number trace

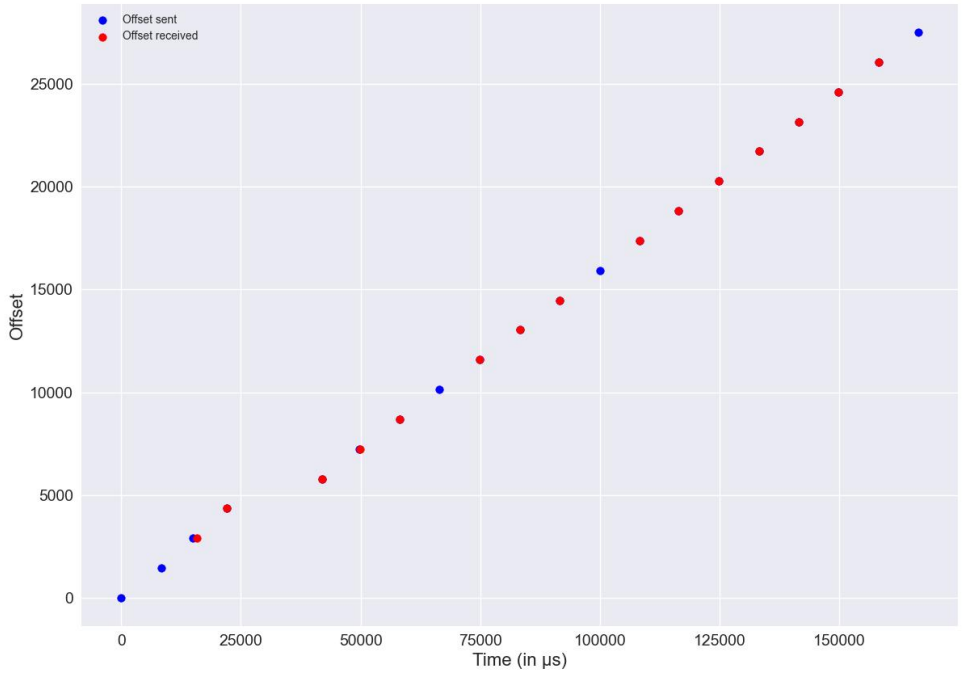
1. **10000 lines**



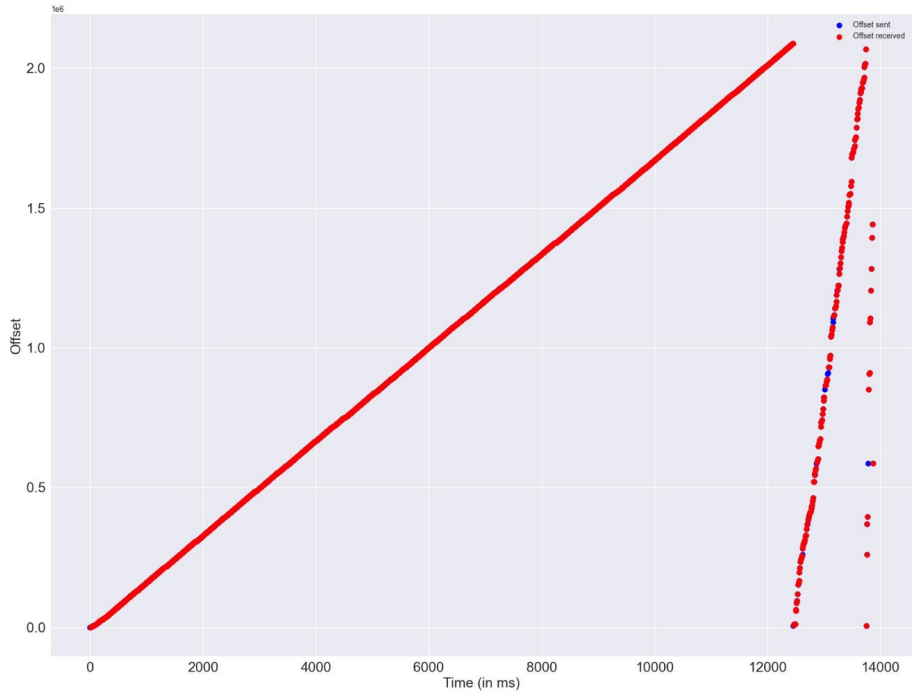


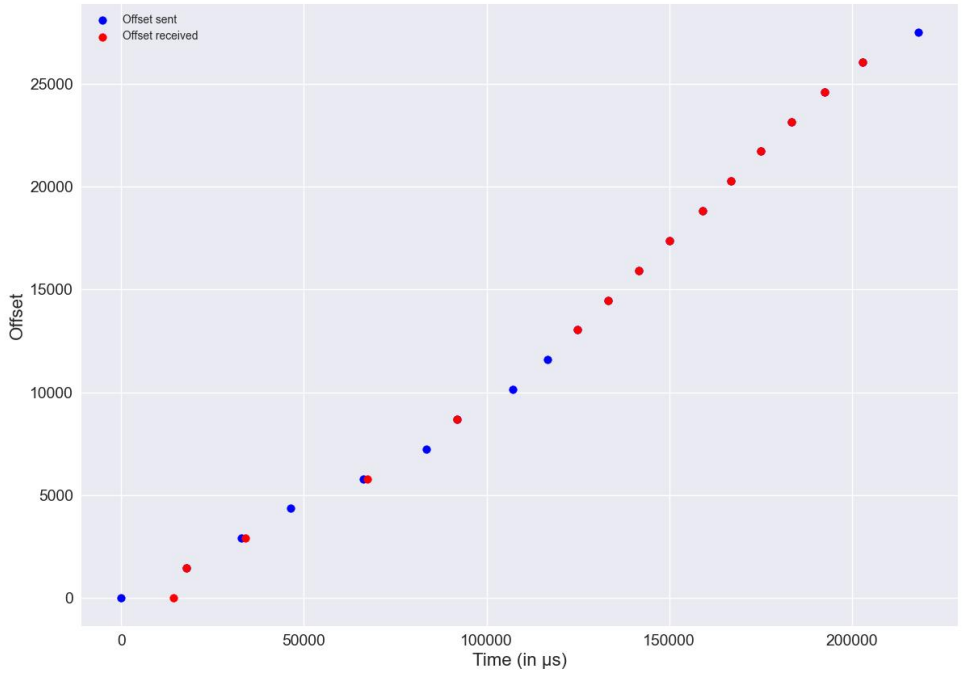
2. **20000 lines**



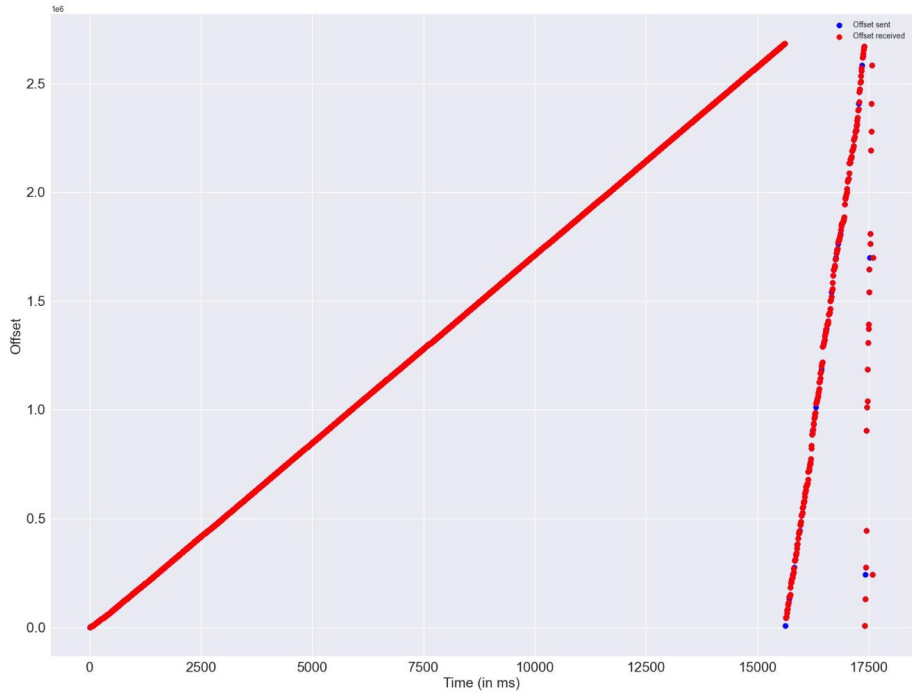


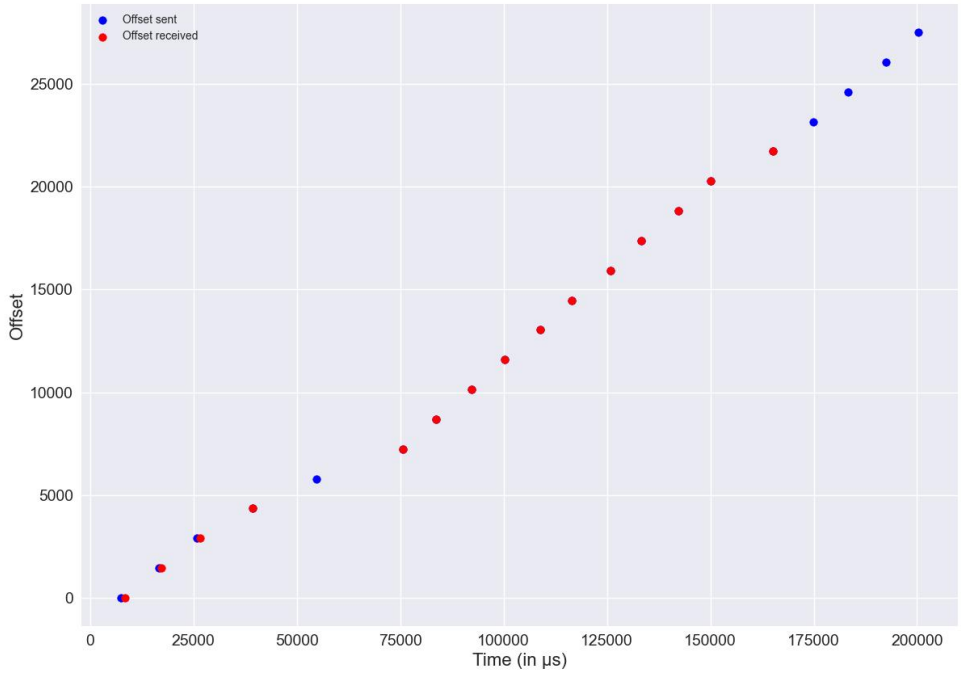
3. **30000 lines**



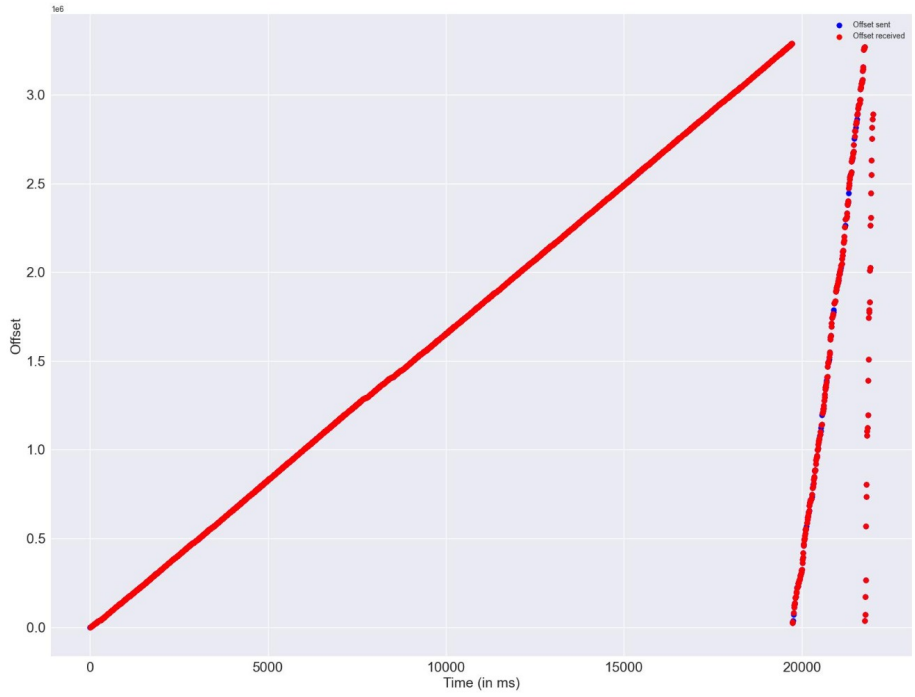


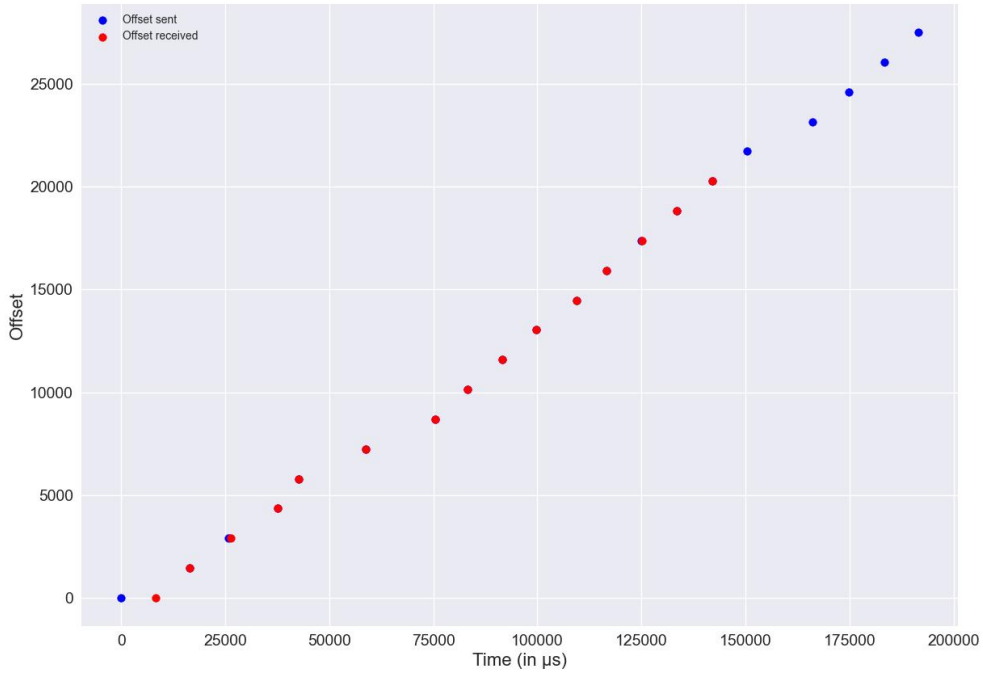
4. **40000 lines**





5. **50000 lines**



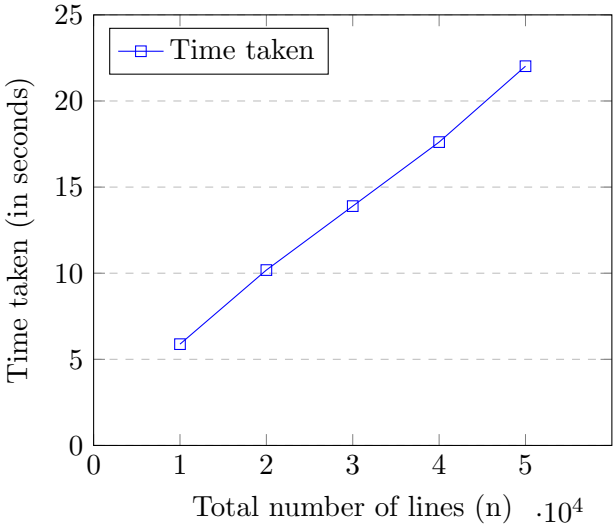


6. **Observation** In almost 3 iterations, all the lines are getting fetched from the server. The second figure in each of the above plots is a zoomed in version. We can observe that most red dots overlap with the blue dots but there are sometimes "no" red dots for the blue dot, meaning that that line couldn't be fetched successfully. We have used a linux system with both client and server running on the same PC. If the server would say be another system or vayu, the distance between the red and blue dot would no longer be subtle, and some separation would be seen between them.

§3.2 Time taken by varying total number of lines

Total number of lines	Time taken (in s)
10000	5.881
20000	10.183
30000	13.896
40000	17.613
50000	22.020

Time taken for different total number of lines



- We observe that the time increase linearly with the number of lines, which is what we expect.

§4 Naive Approach

We have also implemented a single threaded program. We have also attached it in the assignment zip.

§5 Contribution

Parth Patel (2021CS10550) - 10
Tript Sudhakar (2021CS10110) - 10