CS-342
Operating Systems
Section: 1
PROJECT 1 REPORT

Mehmet Hasat Serinkan

21901649

Mehmet Eren Balasar

22001954

20.10.2023

Notes:

- Experiment computer has 6 CPU cores. This is why one of the data points is 6. We wanted to see the breaking point.
- For Part B, we used both incrementally and randomly generated numbers as inputs for experiments. The largest number is 10 000.
- In Part B's experimentation, we saw that randomly generated number sets and incrementally generated number sets do not make a difference because of the "10 000" limit. So, we only used randomly generated numbers for Part A's experiments. Plus, to get an accurate comparison between threads and processes we had to use the "10 000" limit also for Part A's experiment inputs. However, Part A code works without a max number limit.

Part A Data (Processes + Message Queue)

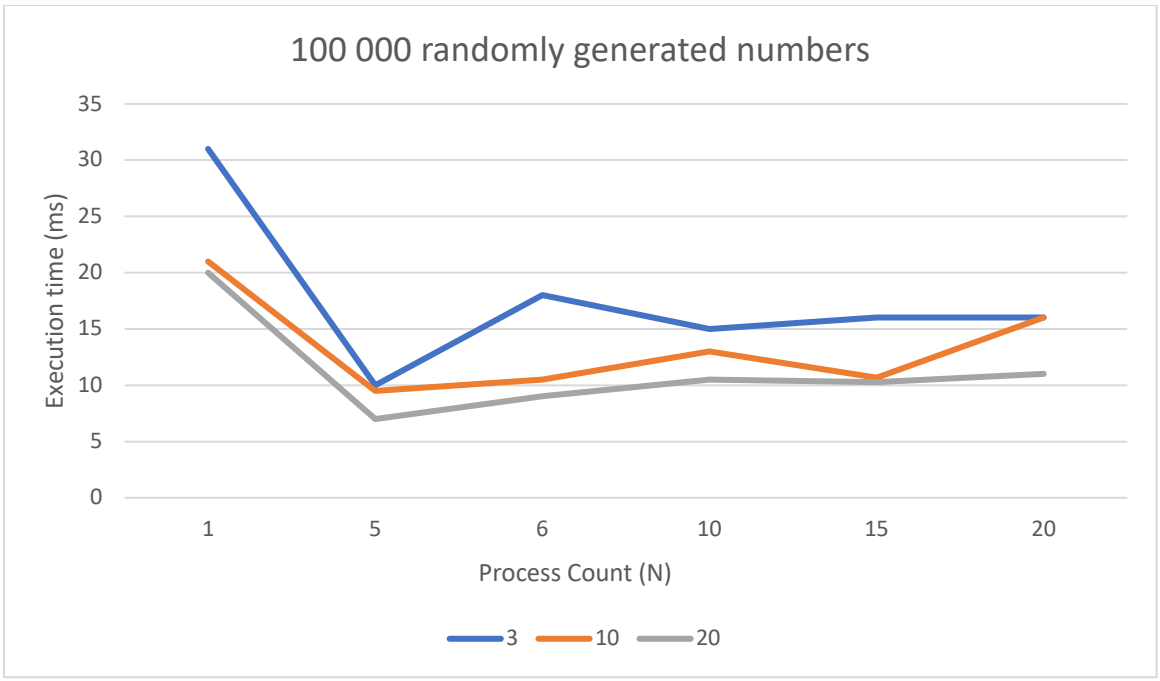
- 100 thousand randomly generated numbers (execution times are in ms)

Process Count (N) ⟹ / Message Size (Number of Primes)(M) ⇩	1	5	6	10	15	20
3	Min:27.1 Max:35.2	Min:7.2 Max:14.0	Min:14.0 Max:22.6	Min:11.9 Max:17.9	Min:12.6 Max:19.9	Min:12.2 Max:19.8
10	Min:19.6 Max:22.4	Min:8.2 Max:11.7	Min:8.9 Max:12.3	Min:9.0 Max:17.0	Min:8.2 Max:13.5	Min:11.7 Max:22.0
20	Min: 19.2 Max: 21.5	Min: 4.7 Max: 10.4	Min: 7.4 Max: 12.0	Min: 8.4 Max: 10.5	Min: 8.2 Max: 12.5	Min: 10.3 Max: 12.2

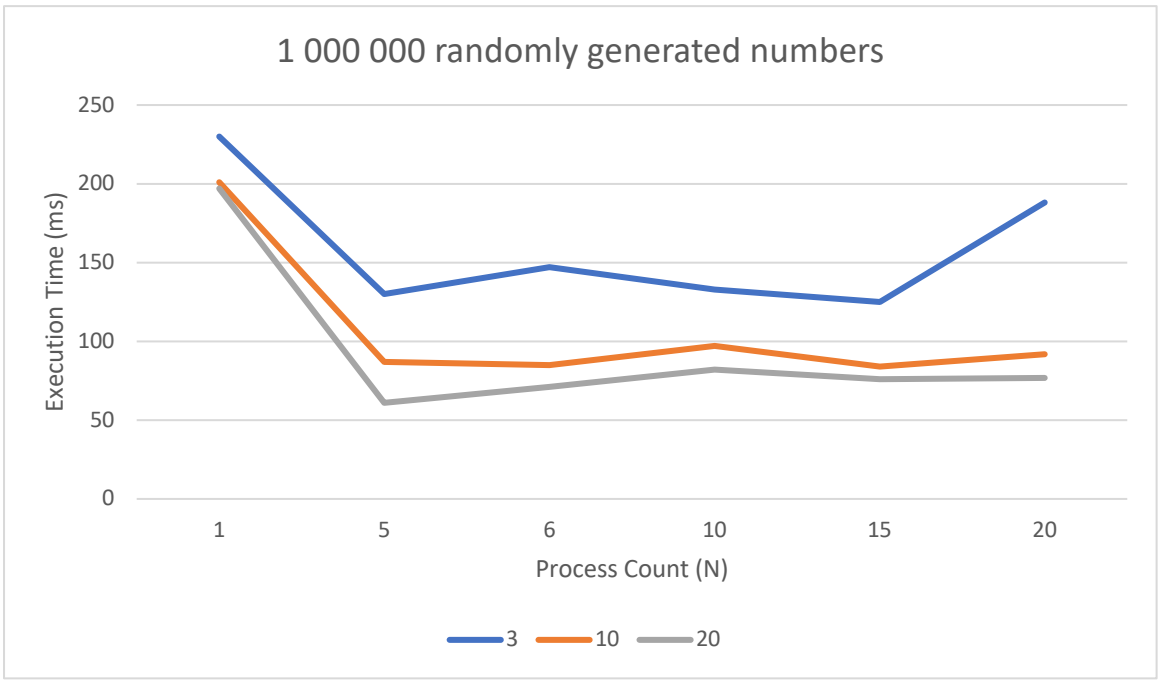
- 1 million randomly generated numbers (execution times are in ms)

Process Count (N) ⟹ / Message Size (Number of Primes)(M) ⇩	1	5	6	10	15	20
3	Min:212.9 Max:255.5	Min:110.3 Max:162.4	Min:134.3 Max:160.8	Min:105.6 Max:160.8	Min:108.5 Max:143.9	Min:150.2 Max:227.4
10	Min:198.3 Max:205.8	Min:72.4 Max:107.2	Min:75.1 Max:95.9	Min:82.1 Max:114.6	Min:64.5 Max:105.3	Min:79.8 Max:106.7
20	Min: 189.1 Max: 205.7	Min: 55.6 Max: 67.1	Min: 69.4 Max: 72.8	Min: 78.9 Max: 86.5	Min: 61.1 Max: 91.3	Min:69.1 Max: 86.7

- 100 thousand randomly generated numbers graph (different lines are different M values)

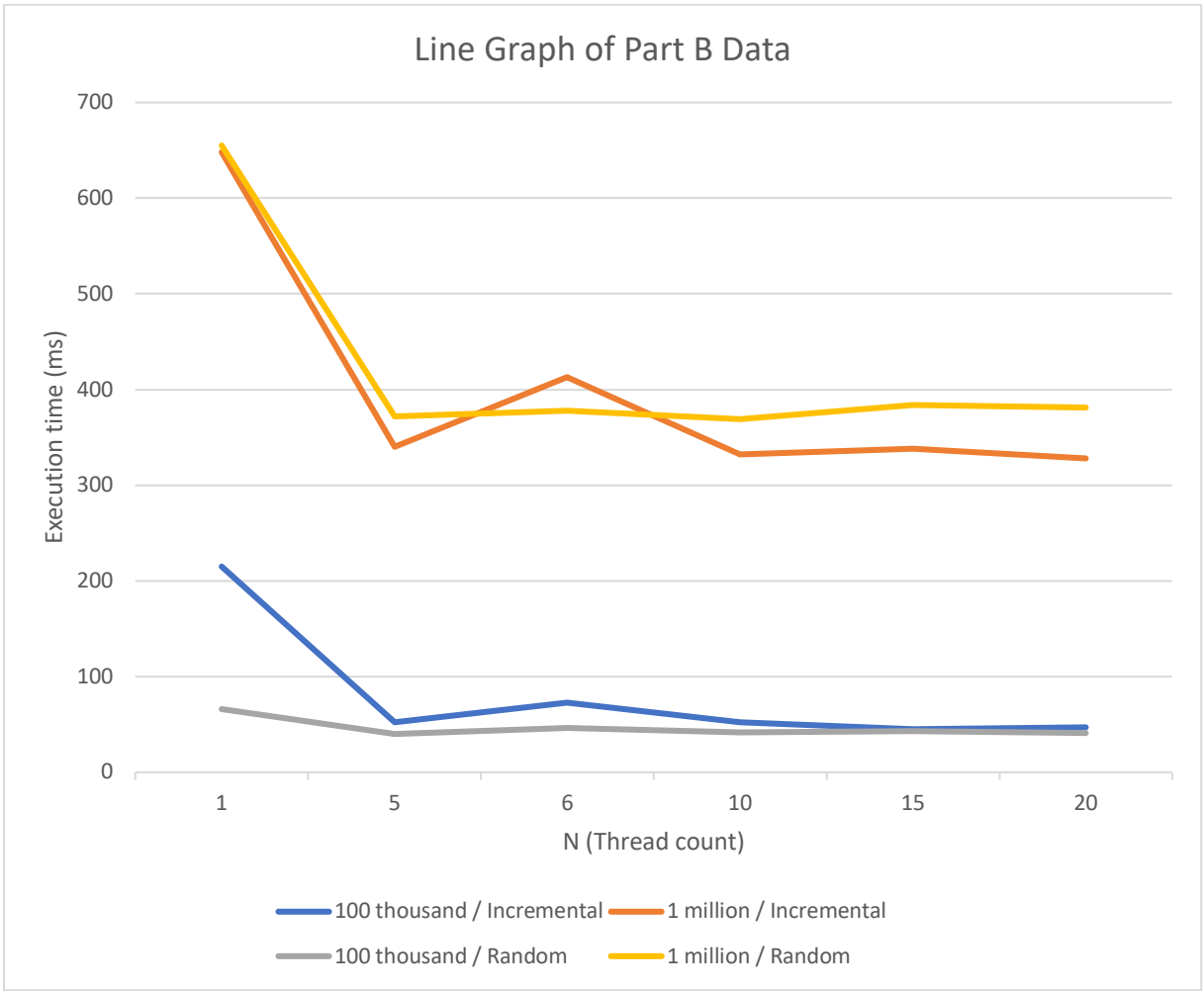


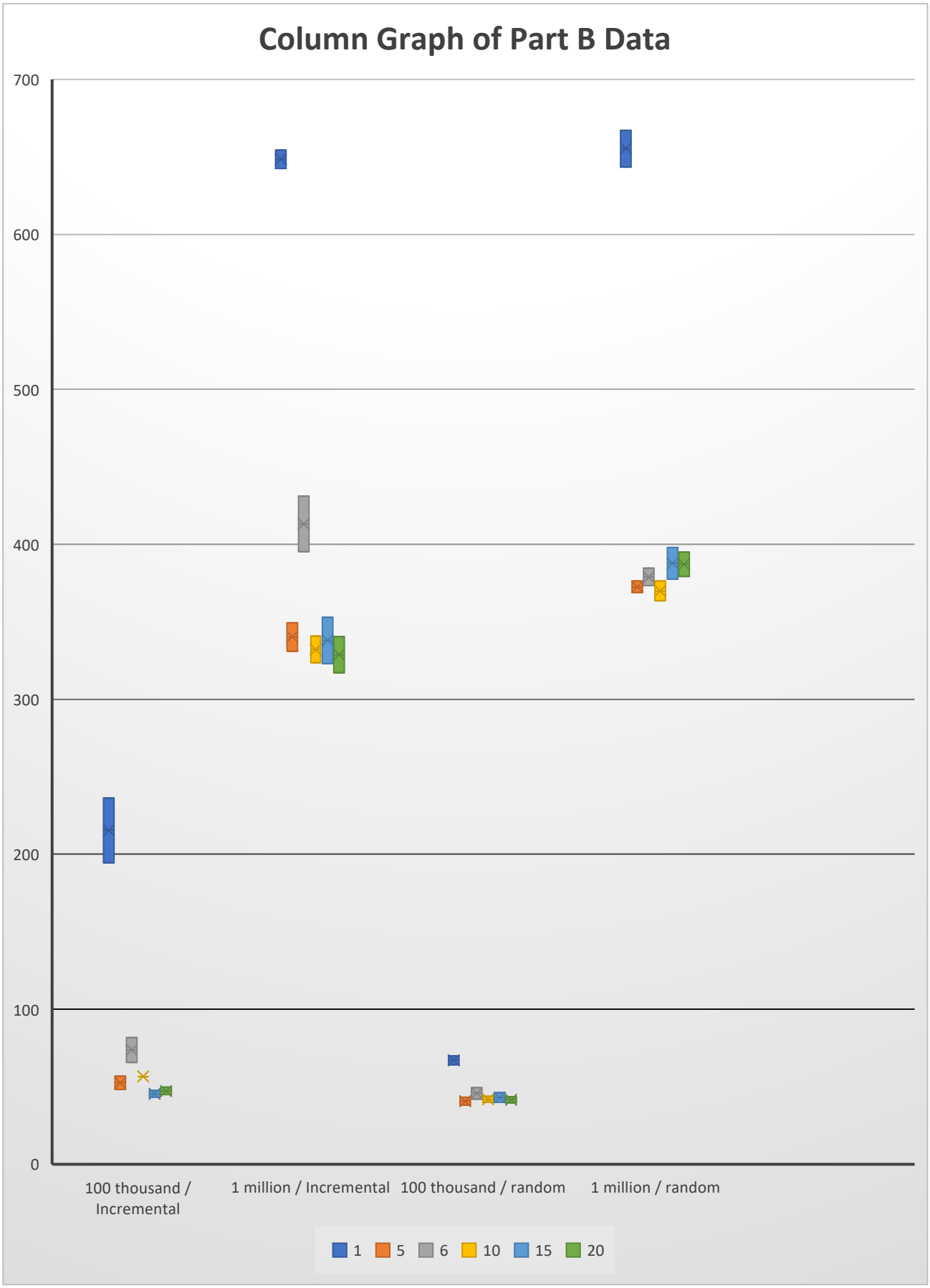
- 1 million randomly generated numbers graph (different lines are different M values)



Part B Data (Threads)

Thread Count ⇨ / Number Count and Type ⇩	1	5	6	10	15	20
100 thousand / Incremental	Min:194.5 Max:236.3	Min:48.2 Max:56.9	Min:65.7 Max:81.7	Min:48.7 Max:56.6	Min:43.1 Max:47.6	Min:44.8 Max:49.5
1 million / Incremental	Min:642.6 Max:654.5	Min:331.1 Max:349.4	Min:395.3 Max:431.1	Min:323.5 Max:340.8	Min:323.0 Max:352.9	Min:317.1 Max:340.4
100 thousand / random	Min: 64.3 Max: 69.9	Min: 38.2 Max: 43.2	Min: 42.0 Max: 49,3	Min: 39.8 Max: 43.6	Min: 39.8 Max: 46.2	Min: 39.6 Max: 43.5
1 million / random	Min: 643.4 Max: 667.3	Min: 369.0 Max: 376.2	Min: 373.5 Max: 384.5	Min: 362.6 Max: 376.2	Min: 377.6 Max: 398.0	Min: 379.4 Max: 395.2




Column Graph of Part B Data
700
600
500
400
300
200
100
0
100 thousand / Incremental
1 million / Incremental
100 thousand / random
1 million / random
1
5
6
10
15
20

Summary and Analysis:

It's evident that when a single thread is employed, the program requires the most time to complete. However, when the thread count is increased to six, there's a significant reduction in execution time. This decrease isn't precisely proportional to the increase in threads, as the creation of threads and other ancillary tasks consume additional time. Yet, on a computer with six CPU cores, this reduction is both acceptable and desirable.

Nevertheless, when the thread count exceeds five or six, there's no further improvement in performance. This is due to the lack of available CPU cores, leading to more frequent thread scheduling and thus, less concurrent running of threads. The maximum number of threads that can operate concurrently is six, although it's likely that the kernel doesn't allocate all cores to this C program, meaning only four to five threads may run concurrently (probably this is also the reason why the 6-threaded program does not run faster than the 5-threaded program). In fact, when we increase the thread count unnecessarily (to more than 6), the program runs slightly slower since all the scheduling, context switches etc. take time.

We can say the same things also for the processes. A decrease in the execution time can be seen when N goes from 1 to 6. After that, there is no performance increase.

Also, the first program (the one utilizing processes) is much faster than the second in all cases. This is mainly because of the different nature of the two programs. In the first one, all the processes, including the parent, run concurrently. Parent writes the data and childs sends the data at the same time. However, in the second program, main thread waits for the workers to finish.

About different M values in Part A:

When we increase M, almost always program executes faster. We believe that the reason for that is the decreasing number of send and receive executions. It is expensive to access the message queue. When we increase the number of primes that can fit in a message, there are less messages sent and received.