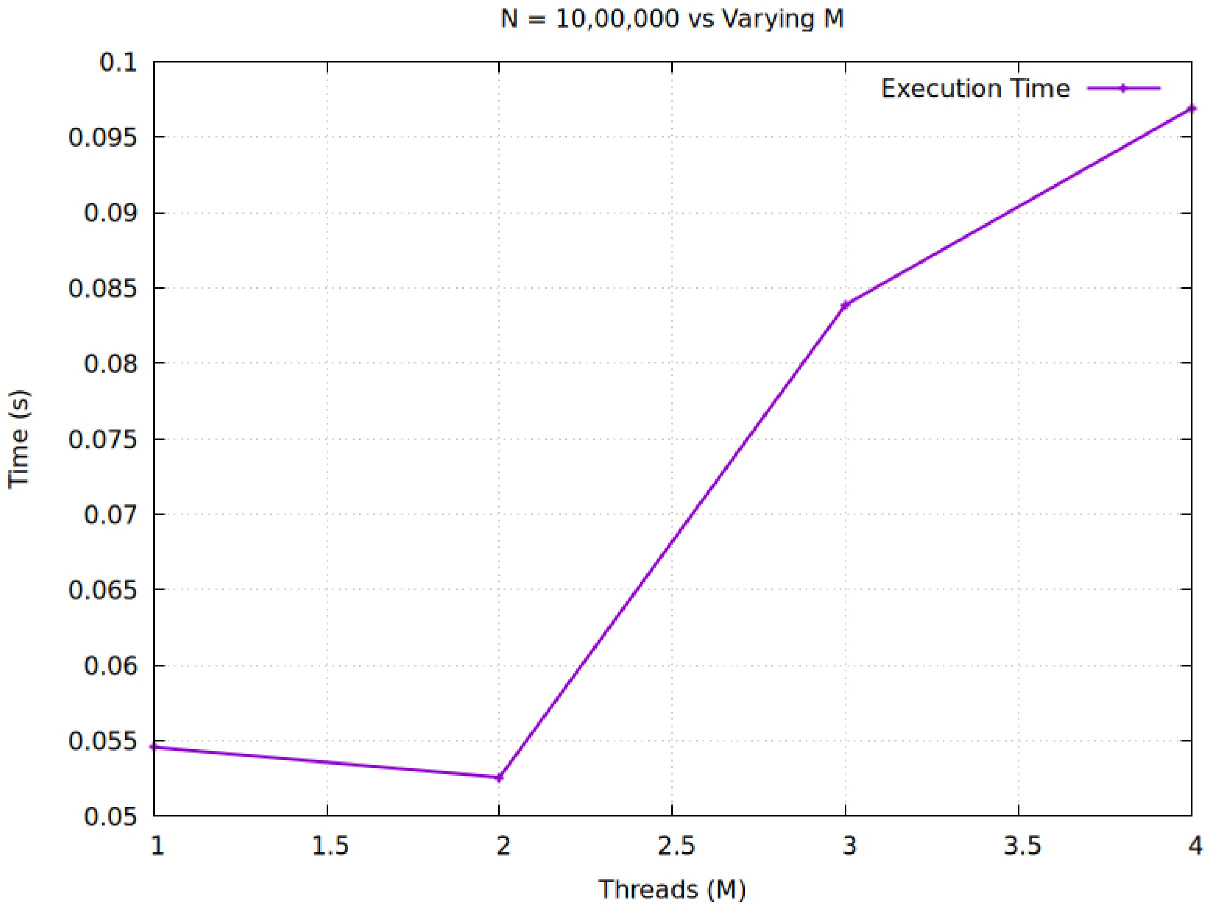
Week 5
(Introduction to Concurrent Programming) Report

Case A:

Graph:



Observation:

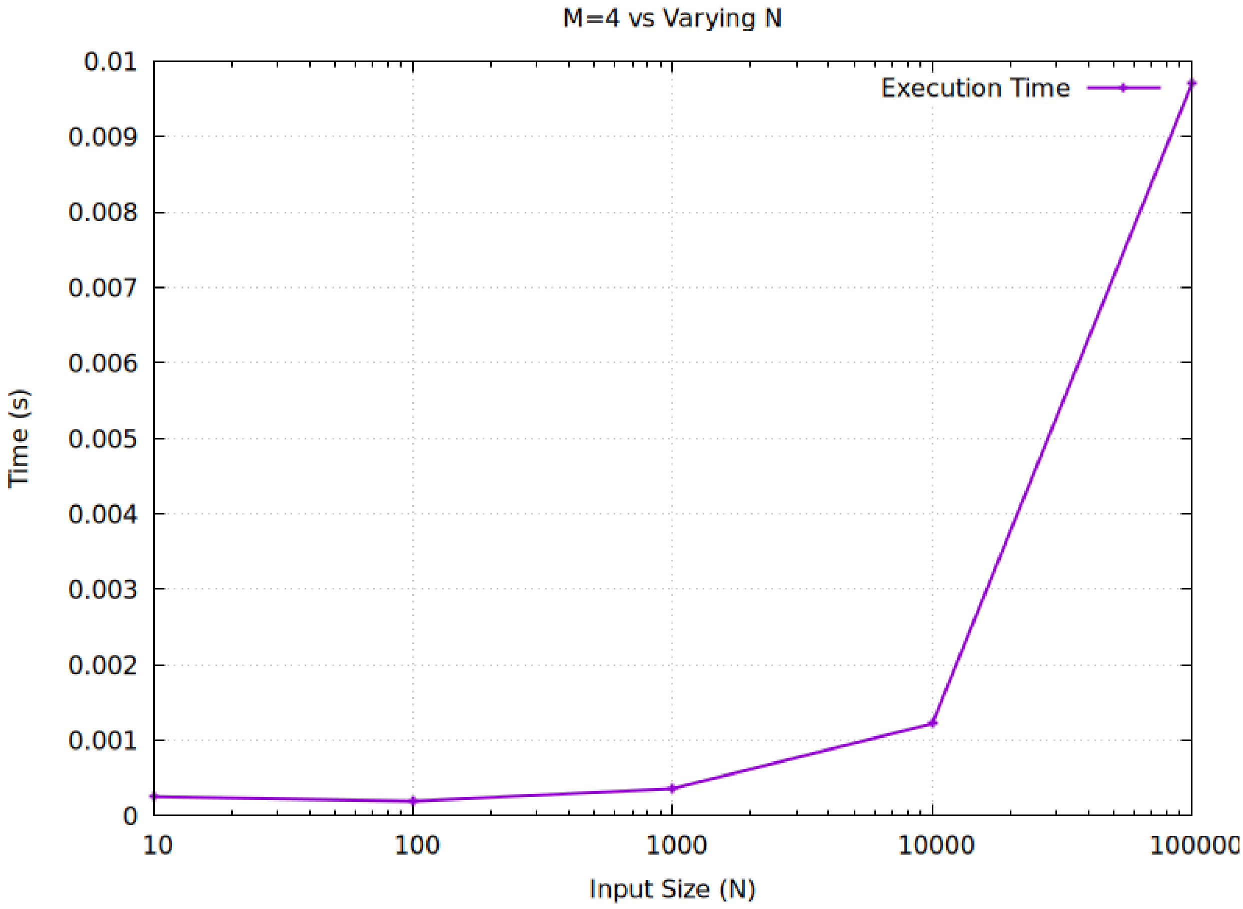
- The runtime decreases a bit from 1 to 2 threads then increases for 3 and 4 threads.

Analysis:

- **Synchronization Bottleneck:** The atomic variable currIdx is shared by all threads, and each update to that variable requires atomic operations. This requires serialized access which results in bottlenecks.
- **Linked-List Order:** Due to insertion with order preservation, the threads can’t run concurrently, which leads to internal sequential execution.
- **Cache and Memory Contention:** With an increase in threadcount, the cores start competing for memory bandwidth and cache lines, resulting in decreased performance

Case B:

Graph:



Observation:

- For smaller values of N the execution times remain low and fairly constant.
- The execution starts increasing linearly around N = 10000 to 100000.

Analysis:

- **Overhead of Thread Creation:** For small values of N, the execution time is dominated by creation and management of the 4 threads, hence the graph doesn't scale.
- **Complexity of Insertions:** With increasing values of N, the cost of insertion takes over as the dominating factor compared to the fixed overhead hence the graph starts scaling linearly.