Assignment 2a Report

1. Results from the multithreaded counter program

Thread number	1	100	1000	10000
Final counter value	10	1000	10000	100000
Time taken(ms)	0	4	27	278

2. Results from the collection 1 program

Time taken to add elements to Vector (ms)	3
Time taken to add 100k elements to ArrayList (ms)	1

3. Results from the collection 2 program

Thread number	1	100
Time taken to add elements to HashTable (ms)	7	10
Time taken to add elements to HashMap (ms)	4	14
Time taken to add elements to ConcurrentHashMap (ms)	22	15

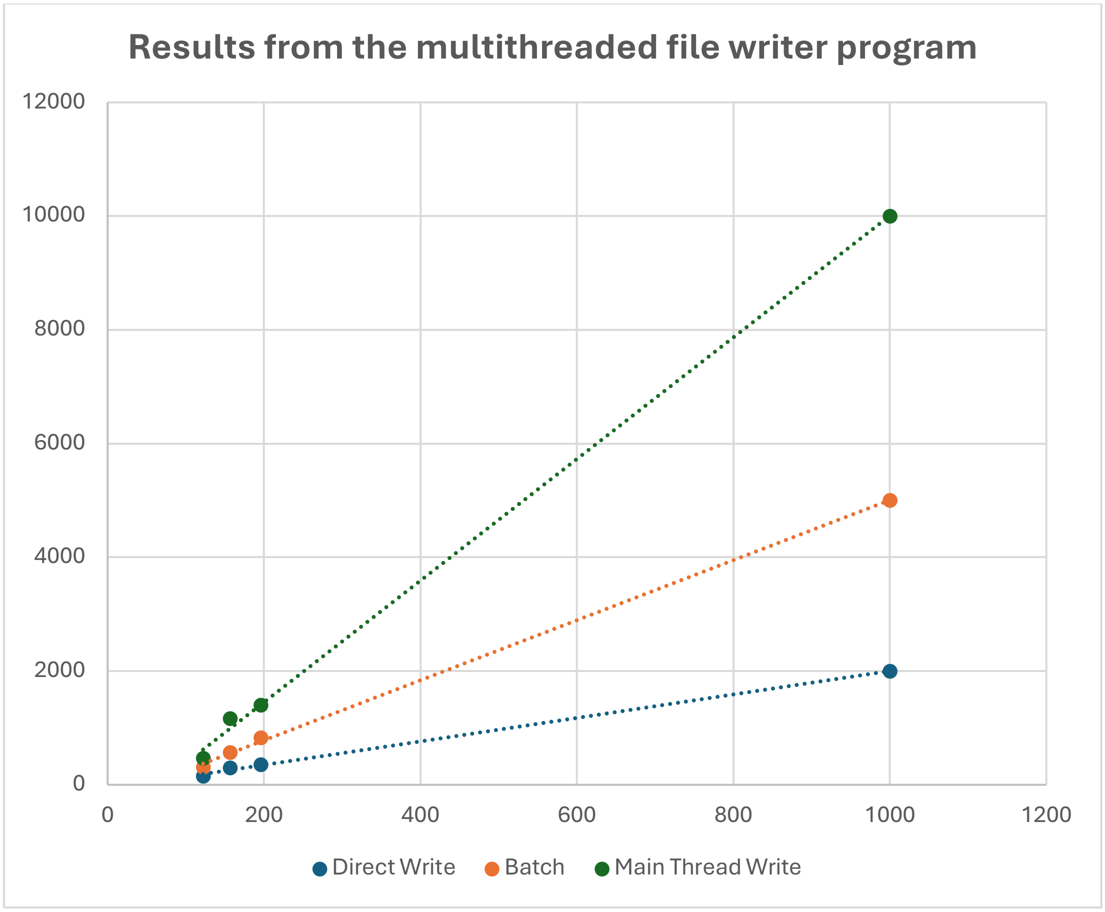
4. Results from the multithreaded file writer program

This program runs with 1000, 2000, 5000, and 10,000 threads, and implements three approaches to writing to a file using multiple threads:

1. direct write in each thread
2. bath write after thread completion
3. main thread write

Number of threads generated	1000	2000	5000	10000
Time taken for Approach 1 (Direct Write) (ms)	157	291	561	1165
Time taken for Approach 2 (Batch Write) (ms)	122	155	306	468
Time taken for Approach 3 (Main Thread Write) (ms)	196	348	828	1395

Below is a graph on the results:



Insights and takeaways:

1. Performance aspect:

- Main thread write took the longest time to write to the file. It may be that this approach accumulates data in memory and writes all at once, resulting in a significant overhead.
- Direct write was faster than main thread write, may due to the frequent input output operations
- Batch write was the fastest, as it writes data in batches after all threads have completed their tasks. This approach reduces the overhead of frequent I/O operations, and synchronization overhead.

2. Scalability:

- Main thread suffers the most as the number of threads increases due to the large amount of data stored in memory and synchronization overhead during collection.
- Direct write also suffers as the number of threads increases, but not as much as main thread write.
- Batch write is the most scalable approach, as it writes data in batches after all threads have completed their tasks.

Alternative implementations:

1. Only one thread writes to the file while the threads are generating the strings.

2. Write data in an ascending timestamp order

Results from the alternative implementation:

- Producer Threads: Time taken = 1169 ms
- Consumer Thread: Time taken = 2591 ms
- Total Time (Producer + Consumer): Time taken = 2591 ms

Pros and Cons of the alternative implementation:

- Pros:
  - The PriorityBlockingQueue ensures ascending timestamp order.
  - The generation and writing processes are independent.
  - Only one writer ensures thread safety
- Cons:
  - One single writer may become a bottleneck when the number of threads increases.
  - The use of PriorityBlockingQueue requires additional memory and adds complexity, which may introduce overhead.