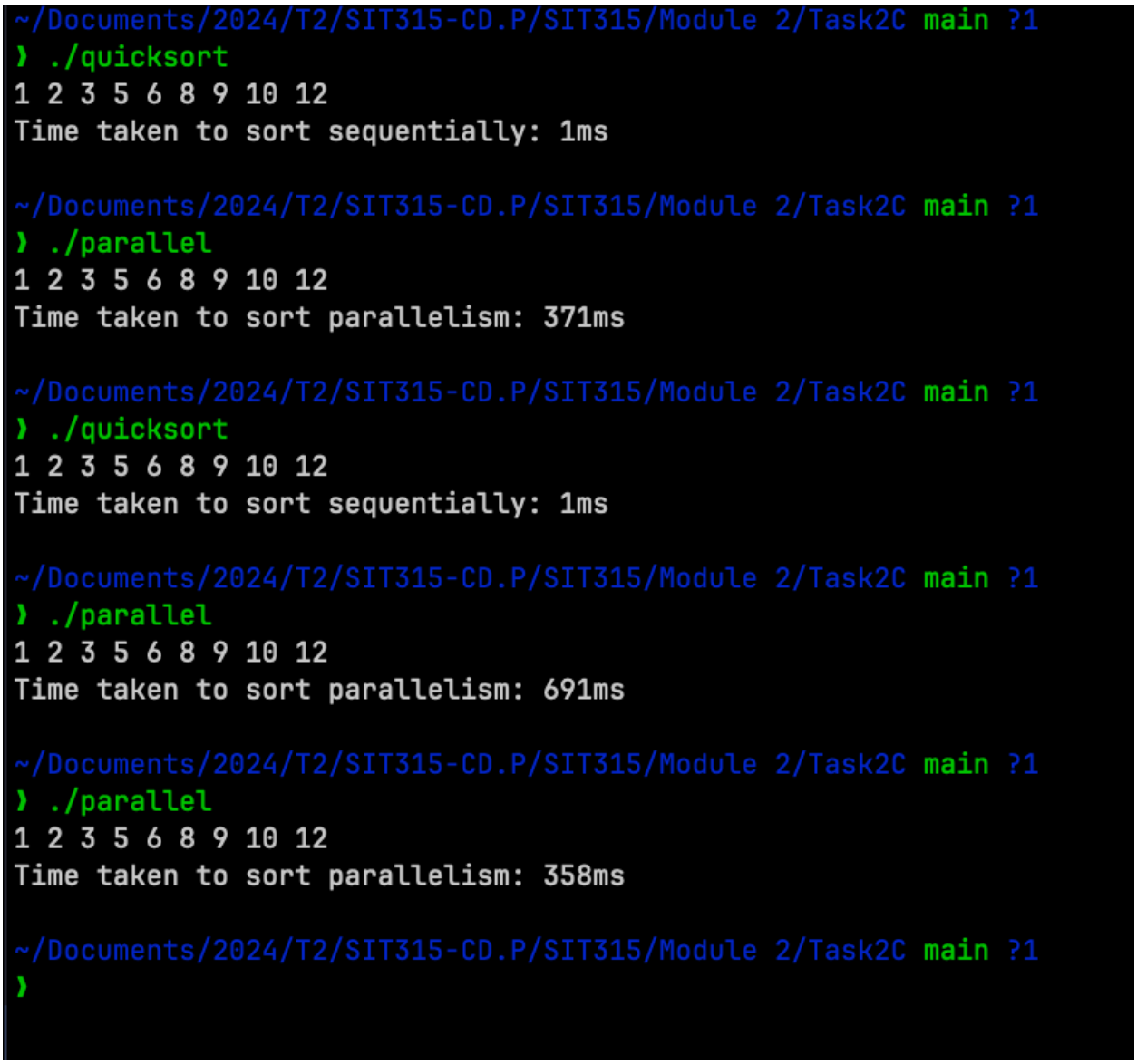
1. Speed comparison: on different input



```
~/Doc/2024/T2/SIT315-CD.P/SIT315/Module 2/Task2C main ?1       12:32
❯ ./quicksort
1 2 3 4 5 8 9 21 24 24 32 43 45 54 54 58 66 67 75 78 86 88 97 99
Time taken to sort sequentially: 4ms

~/Doc/2024/T2/SIT315-CD.P/SIT315/Module 2/Task2C main ?1       12:32
❯ ./parallel
1 2 3 4 5 8 9 21 24 24 32 43 45 54 54 58 66 67 75 78 86 88 97 99
Time taken to sort parallelism: 860ms
```

Analsis:

The sequential version of QuickSort algorithm is faster than the parallel version using pthread.h library due to thread creation overhead, granularity of parallelism, CPU context switching and threads synchronization overhead.

1. Thread creation overhead: creating and managing threads incurs overhead. For small tasks or small arrays, the cost of creating threads can exceed the benefits of parallelism. In phtread.h, multithreading implementation every recursive call potentially creates a new thread and this can lead to excessive thread creation, especially for small subarrays which significantly increases overhead.

2. Granularity: as the quick sort progresses, the size of the subarrays decreases, for small subarrays, the sorting operation is quick, and the overhead of creating threads to handle these small tasks can be disproportionate which leads to slower performance.

3. Context switching with multiple threads running concurrently, the CPU may spend additional time switching between threads which introduce delays, especially if there are more threads than CPU cores.

4. Synchronization overhead: Thread joining introduces synchronization overhead because the threads are waiting to finish and if the tasks are small, this waiting time can contribute significantly to the total execution time.