Asst 02: Spooky Searching Results

1 Introduction

1.1 Layout

The layout of this README:

- Introduction
- Results (Presented in Graphs)

1.2 Objective

Attempt to determine:

- A general trend of: SIZE(**n**) vs. Time to search for both processes and threads.
- Whether a reasonable tradeoff point exists between using processes and threads.
- Whether a reasonable tradeoff point exists between using neither or a processes/thread.

Note: Our sequential search was simply traversing the array, without any threads or processes. Therefore it serves as both a baseline for the tests.

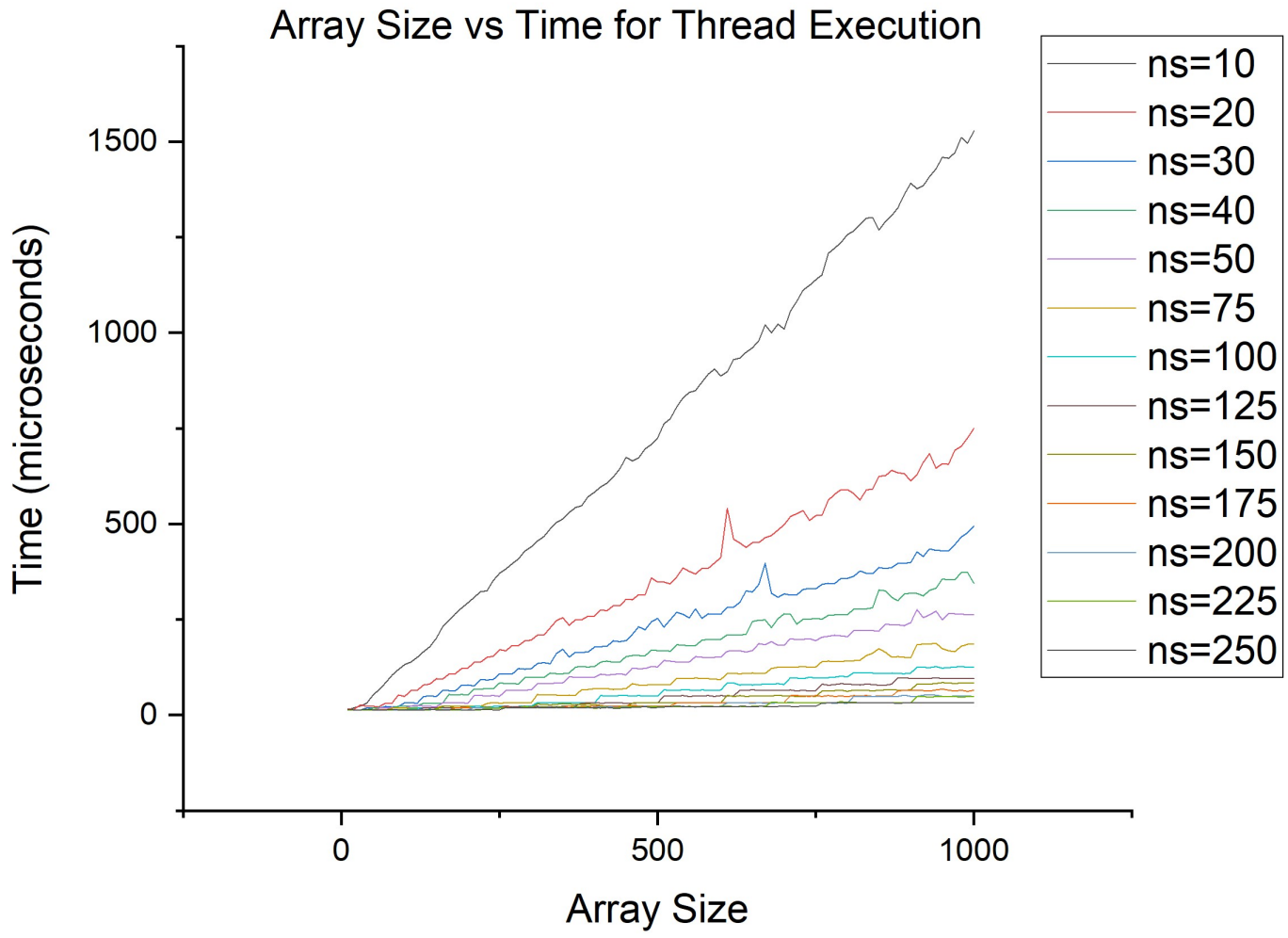
2 Results

2.1 SIZE(**n**) vs. Time

Sequential:

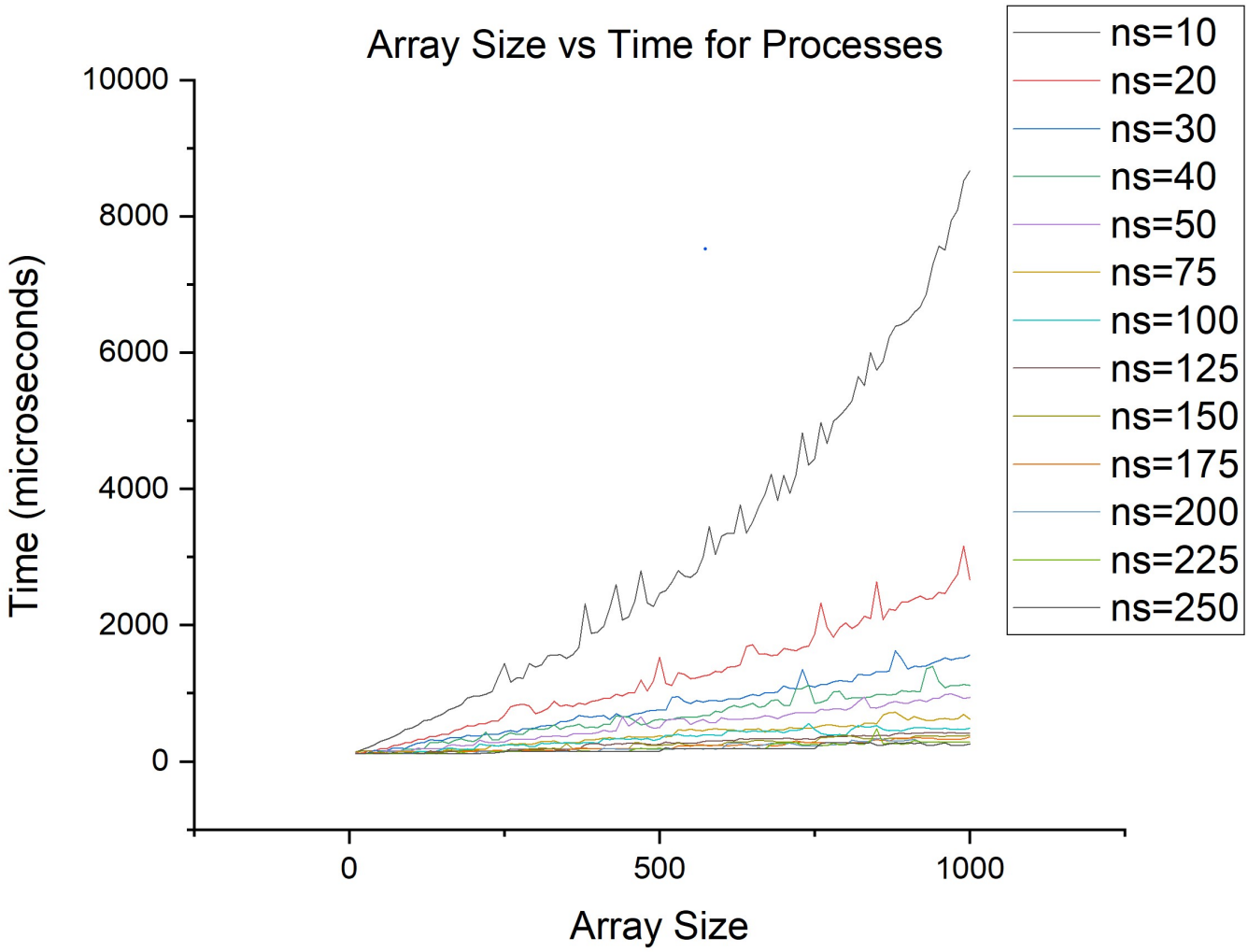
For *Sequential*, we discovered that it took a noticeable amount of time only when SIZE(**n**) increased to extremely large amounts - greater than 20,000. Therefore, the conclusion was made that it was unnecessary for us to create a graph for a *Sequential* graph.

Thread:



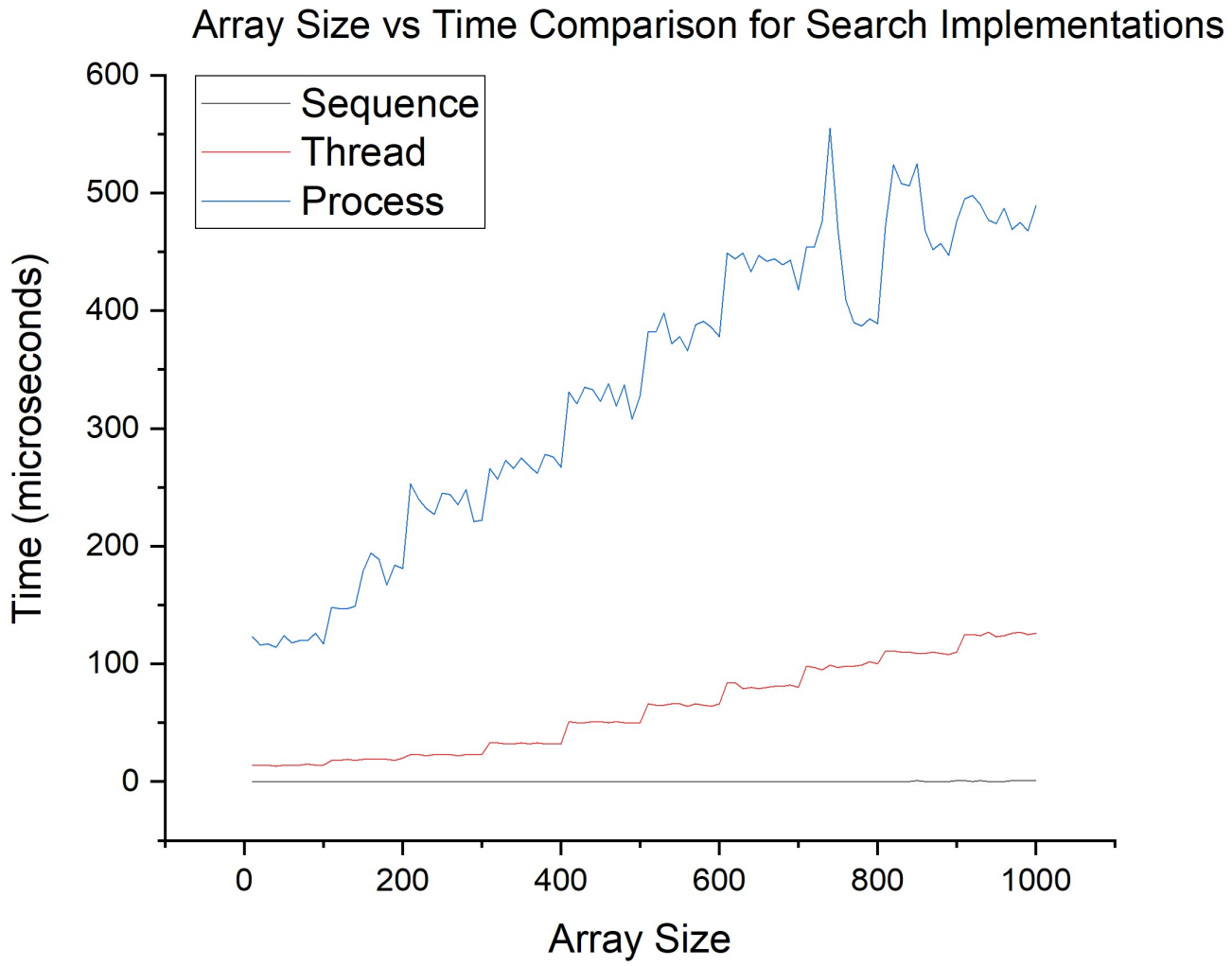
For *Threads*, each line represents one case in which the split size(**ns**) was kept constant. It is quite noticeable that as the array size(**n**) increased, the time increased linearly. This allows us to conclude, with reasonable confidence that, while keeping split size constant, there exists a correlation between array size and time. Through our graph, we also can observe a negative correlation between the split size(**ns**) and the time taken for each execution, but we will delve into that later, as we have data and other graphs available to more clearly illustrate the relationship.

Process:



For *Processes*, the graph is very similar to that of *Threads*. Each line still represents one case in which the split size(**ns**) was kept constant. Similarily to that of *Threads*, the array size(**n**) does seem to have an influence on the time. However, interestingly, while the split size(**ns**) is large(i.e. when split size is 250), it is perceived that the graphs are linear. As the split size begins to decrease, we view that the graph is more exponential than linear.
This seems to be different than what we observed with *Threads*.

Comparison between *Sequential*, *Threads*, *Process* vs. Time



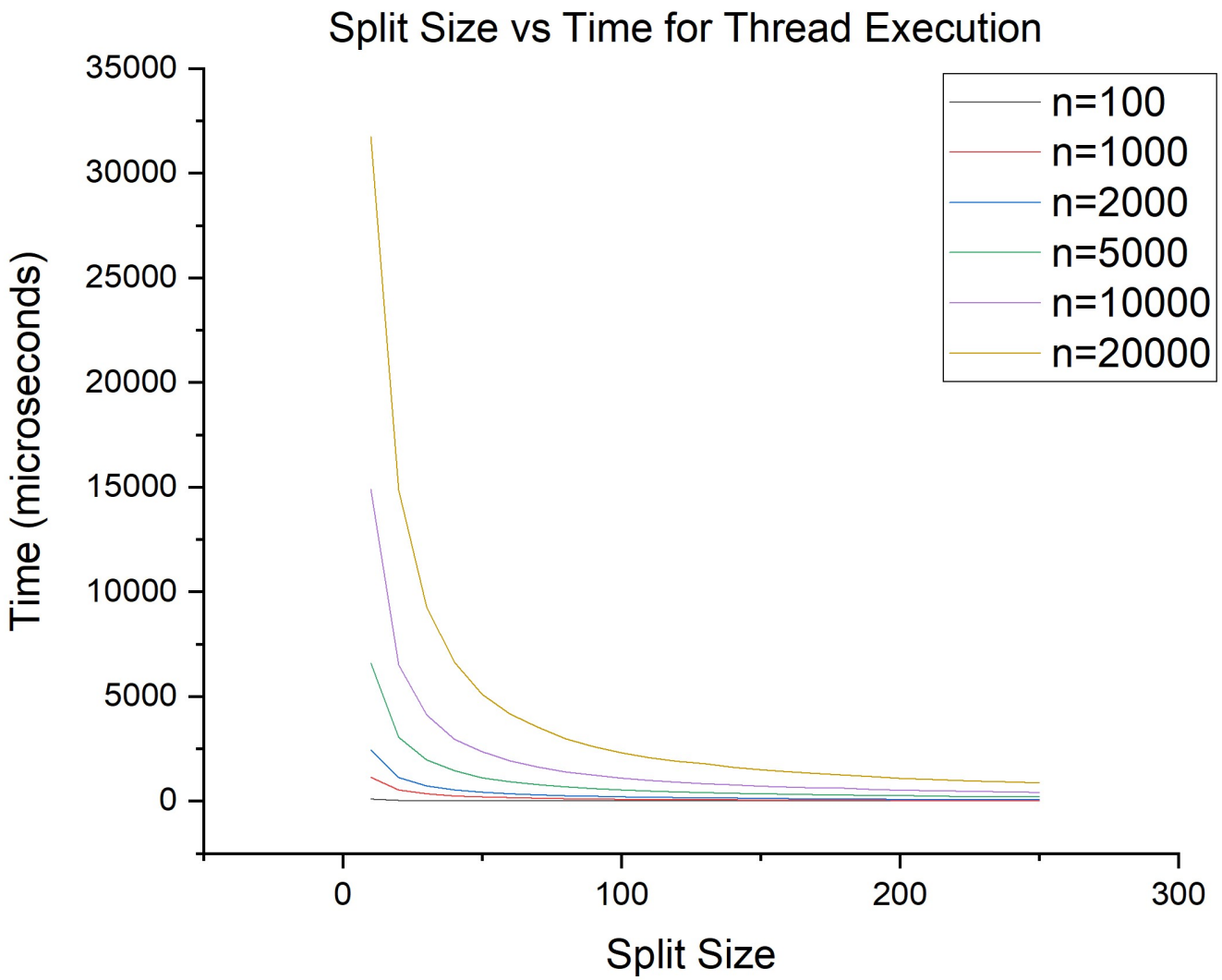
This graph keeps the split size(**ns**) constant at a value of 100. Clearly, *Sequential* seems to be almost constant, with the only perturbation occurring as the array size(**n**) increases to 10,000. *Threads* seem to be fairly linear, increasing only a small amount in time per each increase in array size(**n**). *Processes* seem to be fairly linear as well, but increase time at a much steeper rate than *Threads*.
Another important value that we can conclude from this graph is the turning point between using *Threads* and *Processes*. As shown on the graph, it seems that at 10 threads(*This occurs when array size is 1000 on the graph*) and 1 proccess(*This occurs when the array size is 0-100 on the graph*), the time required for both seem to be similar.

2.2 SPLIT SIZE(**ns**) vs. Time

Sequential:

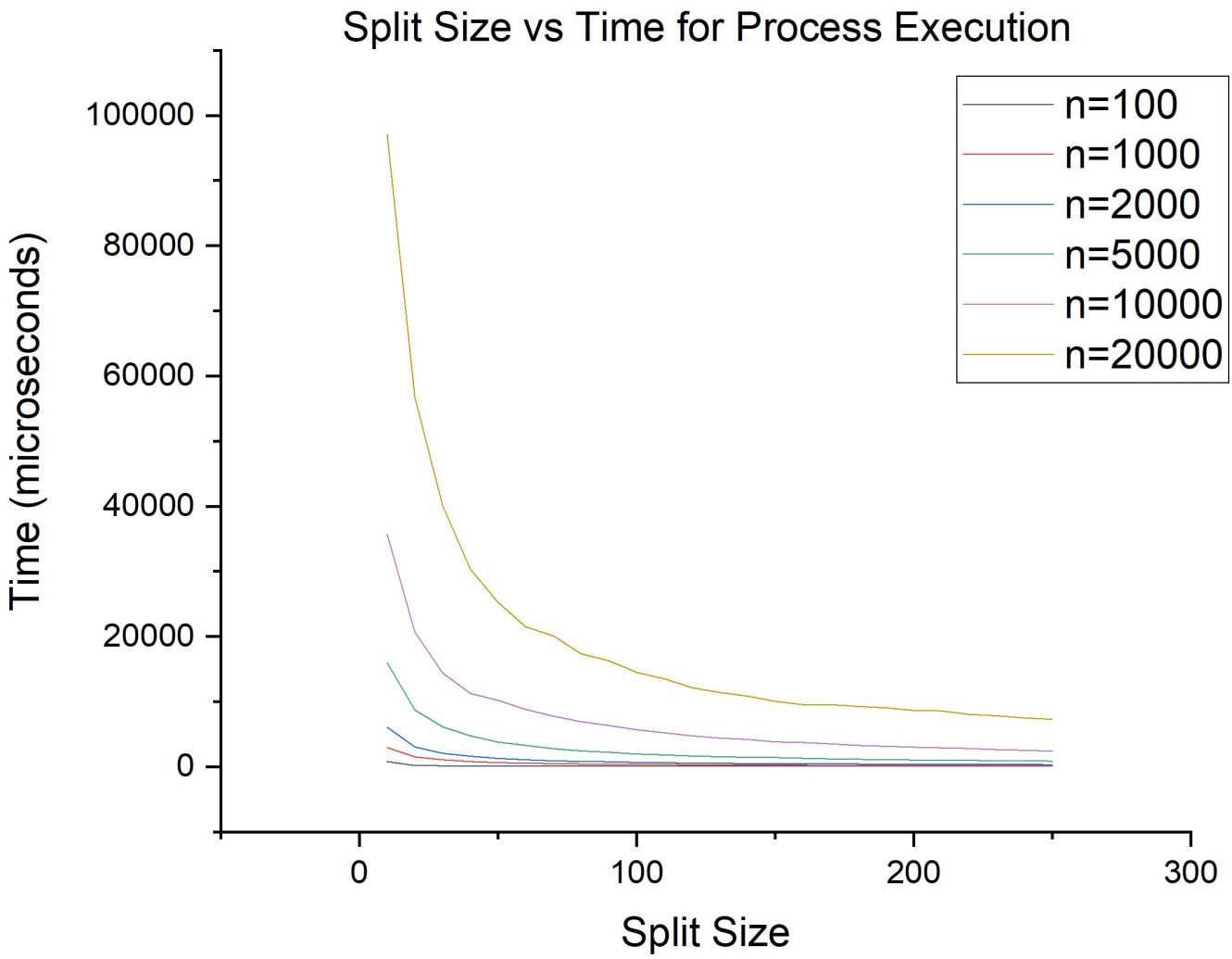
For *Sequential*, there is no split size. Therefore, we decided to not include a graph once again, as it would be the same.

Thread:



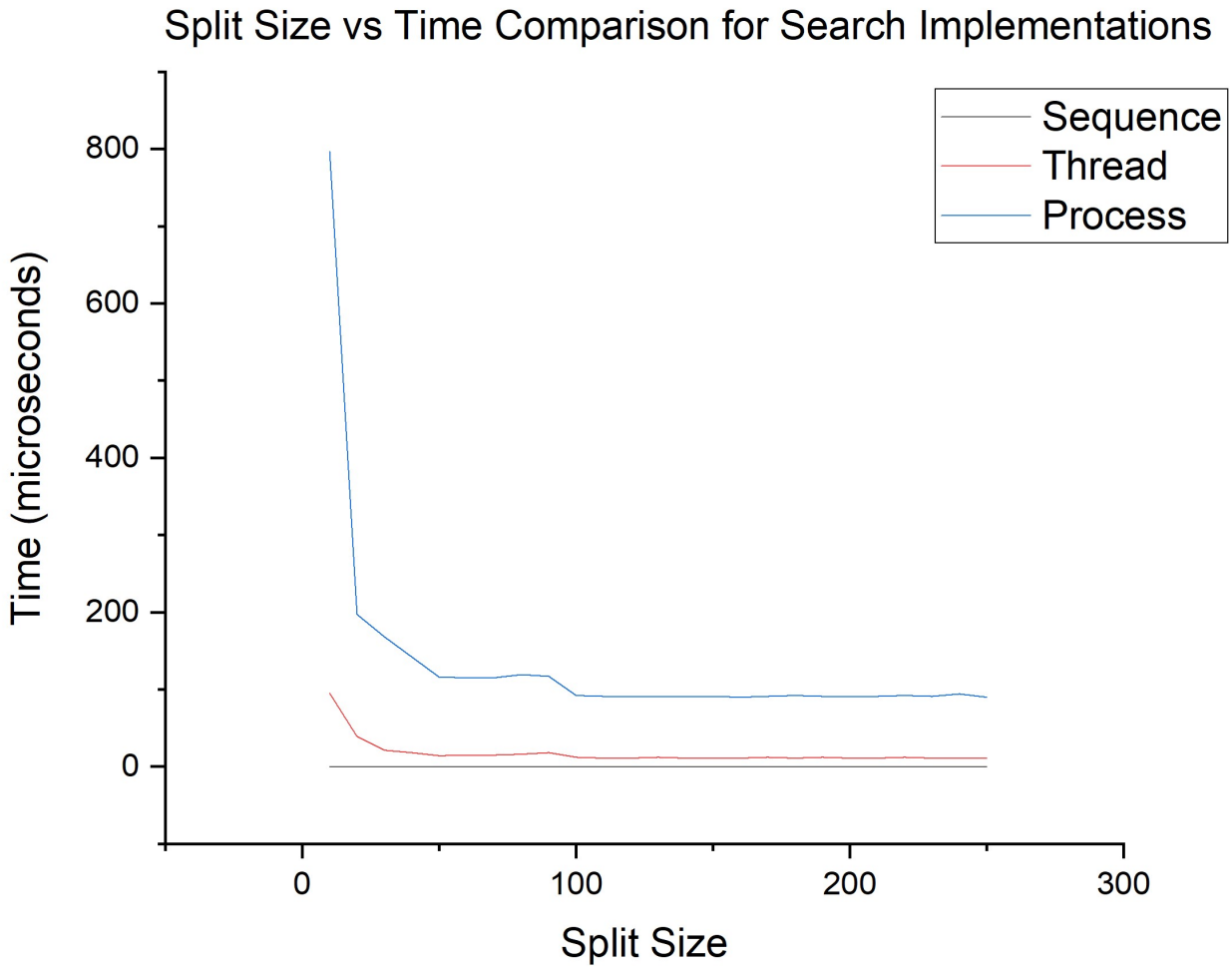
For *Threads* there seems to be e^{-x} relationship between the split size(**ns**) and the time. This is in agreement with our conclusions observed previously. We can also see that the array size has little impact compared to split size. As split size increases, the amount of time decreases drastically.

Process:



Similarily to *Processes* there seems to be a e^{-x} relationship between the split size and the time taken to complete an execution. However, the size of the array seems to have a larger factor on the time, as there is a sizeable margin between different array sizes (represented by the different graph colors).

Comparison between *Sequential*, *Threads*, *Process* vs. Time



Here the size of the array was kept constant at a value of 100. Similarily when we compared the array Size vs. Time, the *Processes* took the longest, with *Threads* noticeably faster, and *Sequential* essentially constant. **We could not seem to find a reasonable turning point for using processes over threads as threads were always faster. Also we could not find a reasonable turning point for using workers versus sequential searching.**