**Optimization of Software Implementation**

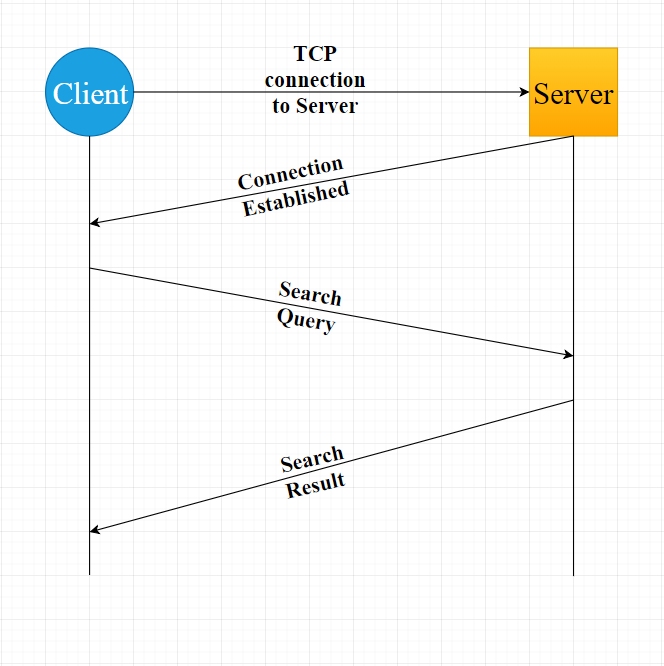
Our project involves the use of data information obtained from public sources, of which these data are organized in two particular data structures; dictionaries and list, and are by no means ordered in any specific sequence. As no information about the order of data wasn’t provided by our data source providers, our group made a presupposition that the obtained data were in random order to begin with. For data processing purposes, we aimed to make our data easy to work with and manipulate by applying concepts learned from various courses taken here at McMaster University. Concepts learned from 2nd year courses through 4th year were introduced in order to accomplish a fast and optimized implementation our software program. These techniques include

1. Sort Algorithm (Comp Sci 2C03)
2. Search Algorithm (Comp Sci 2C03)
3. Socket Programming (Comp Sci 4C03)
4. Multithreading/Multiprocessing (Comp Sci 4F03)

**Quicksort Algorithm and Search Algorithm**

Since our data is presumed to be non-ordered, we applied the quicksort algorithm to order it before any processing is done on the data. Sorting the data first made the process of searching items much faster based on the search algorithm implemented. Initially, a linear search algorithm was implemented, however the runtime of the search algorithm improved from O (N) to O (Log N) based on the binary search algorithm implemented. This improvement is accomplished because the binary search algorithm required a sorted/order data which was obtained by the initial application of the quicksort algorithm.

**Socket Programming (Client to Server)**

Our project involved the implementation of a back-end (i.e. server) and a front-end (for clients). This dynamic was accomplished by applying the knowledge of P2P programming learned from Comp Sci 4C03. The first peer in this case is the client and the second peer the server. Based on the nature of this client-server interaction, the server is always listening and ready to take request from the client. A TCP connection is applied here to ensure security during the interaction between the clients (User) and sever (Back-end). Since our software will likely be utilized by multiple people at once, the TCP connection was a suitable protocol to use because it allowed the server side to talk to multiple clients at the same time. Every time a client accesses our website, it connects to the server by creating a TCP socket accompanied with an IP address and a source port number. Since there are multiple clients likely accessing our server at the same time, the source port numbers are used for distinguishing between the individual clients.

**Multithreading/Multiprocessing**

The data used for this project were obtained from three different ecommerce website e.g. Amazon, Walmart and eBay. The initial implementation of our software was initially executed in a sequential fashion. However, multithreading was introduced to split the execution into different chunks for faster execution. Since we had three websites to made API calls to, we optimized our program to fork three additional threads whereby each thread was responsible to making an API call to a respective website i.e. Walmart, Amazon, and eBay. Since these API calls are independent of each other and uses no shared memory location, the threads were executed in parallel which reduced the timing by a large portion compared to the sequential execution implemented initially. Once the individual threads were done with their respective jobs, they all return and terminate (i.e. join together) to form a single thread execution. A graphical representation of the forking and joining of threads is shown below;

