Data Structure Choices:

The data structures chosen for this problem are:

* A doubly linked list of events each containing a doubly linked list of fighters
* A doubly linked list of events each containing an AVL tree of fighters

The most obvious way to store events would have been an array. Events (UFC301, UFC315, UFC209) are created in incremental numeric order, meaning that there will never be a need to shuffle an array’s contents, given that events will only ever have to be pushed to the end. The issue however is that the number of events is always increasing, a fixed size data structure is simply not able to handle a constantly increasing size of events. Another obvious option for storing events would be some sort of sorted data structure, such as an AVL tree. However, the fact that events will only ever be pushed to the end of a data structure means that all of the complexity of an AVL tree’s functions would be largely wasted (EXPAND ON THIS). While a linked list is not a sorted data structure, its simple insertion at the front method makes it a perfect choice for this use case. Making the linked list a doubly linked list also avoids the issue of needing to traverse, for example, all the way to UFC1 if the eents were being stored in descending order. The worst-case time complexity for finding an event will always be n/2, with the additional constant timings of finding which end of the list to begin traversal from.

Storing fighters is a lot less clear-cut than storing events. Fighters in the dataset are stored in alphabetical order, meaning that they can keep their order depending on the data structure. However unlike the events, there can be insertion of a fighter that comes between two other fighters. This (and other discussed issues) makes an array pointless. A heap is also not suitable as the amount of fighters must be able to increase. A hash table also relies on an array, which is bad for insertion. Our final choices are linked lists and trees, which is why we will be comparing these two data structures directly.

In the dataset, both the events and fighters are already sorted. This means that inserting events and fighters into a doubly linked list will cause the data to be sorted. While a sorted data structure such as an AVL may have been an obvious choice for events, given that events are in increasing numerical order and that

The most significant operation for both data structure implementations is the many to many retrieval of which events a fighter has been in. Across both implementations, all events must be searched for a fighter. This is far more complex than the inverted operation of printing all fighters within an event, as only a single event needs to be scanned. If we consider ‘n’ to be fighters, which would be most logical as the amount of events grows far slower than the amount of fighters (in real world terms) then it is clear to see why the many to many operation is so complex.