```
Inserting end time Vector: 7805 (Micro-sec) 100%
Inserting end time Set: 118676 (Micro-sec) 1520.51%
Inserting end Time List: 24035 (Micro-sec) 307.944%
Inserting end Time Unordered Set: 69915 (Micro-sec) 895.772%
------------------------------------------------------------------
Inserting begin Time Vector: 5.7927e+06 (Micro-sec) 100%
Inserting begin Time Set: 108009 (Micro-sec) 1.86457%
Inserting begin Time List: 21859 (Micro-sec) 0.377355%
Inserting begin Time Unordered Set: 23586 (Micro-sec) 0.407168%
------------------------------------------------------------------
Find Time durations for Vector: 4.70992e+07 (Micro-sec)100%
Find Time durations for List: 9.85486e+07 (Micro-sec)209.236%
Find Time durations for Set: 55509 (Micro-sec)0.117856%
Find Time durations for Unordered Set: 22704 (Micro-sec)0.0482047%

RUN SUCCESSFUL (total time: 2m 32s)
```

By the data received from our program. we can see that the order of which is fastest to slowest for all four for **insertion at the end and finding** is as follows (in comparison to the vector):
Insertion from end:

1) Vector O(1)
2) List O(1)
3) Unordered Set O(1)
4) Set O(logn)

Finding:

1) Unordered Set O(1) or O(n)
2) Set O(logn)
3) Vector O(n)
4) List O(n)

The data shows that for **insertion at the front**, from fastest to slowest is as follows:

1) List O(1)
2) Unordered Set O(1)
3) Set O(log n)
4) Vector O(n)

List:

- It usually has an inserting time complexity of O(1) for both front and end, however, since this is a double ended list, it works slower than the vector when inserting from the end, which has an insert time complexity of O(1) when inserting at the end.

- When inserting from the beginning it's the fastest however because it is a double linked list.
- For finding - it's linear - goes one by one and takes longer than vector because of the reason mentioned above, it's a double ended linked list.

Unordered Set:

- Normally has an insertion time complexity (regardless if it's from the beginning or end) of O(1) but can go up to linear time O(n) in the worst case, depending on the hash functions. Based on our data, we have a time complexity of O(n) because of the number of key collisions.
- For finding, has a time complexity of O(1), making it the fastest one. Does not have to be in order.

Vector:

- Fastest when inserting from the end because it adds a new element at the end of the vector by default. It also effectively increases the container by size one and causes an automatic reallocation of the allocated storage.
- Slowest when inserting from the beginning because it has to traverse all the way back (shifting) - adds a new element at the end of the vector by default.
- For finding - it's linear - goes one by one.

Set:

- Insertion for set, regardless whether it's from the front or the end will be O (logn), therefore, relative to the other containers when inserting from the end it ends up last and when inserting to the containers from the front, ends up third.
- For finding, the time complexity is O(logn), compared to the time complexity of an unordered set which is O(1), set is slower because it has to take the time to order them first