Data structure general format

Node declaration- (think hard about the node. So visualise without lines)

```
Struct [dataType]Node {
      //data stored in Node eg- Object/value
      //pointer (node link another node same node type???)
};
```

NOTE: (ask yourself)

- Does node link to another node(s)? Often yes
- Does node store data? Always yes

Data structure declaration- (think hard about how to add new node in instantiate structure)

[Contains method to LINK up node such as AddAnotherNodeTo]

```
Class [dataType] {
public:
      …
      AddAnotherTo[DataType] //Instantiated data struct add node
private:
      //Root node pointer (root node link to another same node type)
};
```

NOTE: (ask yourself)

- How to add another node? ANSWER is dependent on data structure property
    - BST → Sorting process. So go left go right etc
    - Stack → Push

Arrow notation:

```
[Pointer] -> [Data member] OR (*Pointer).DataMember
```

So

[Pointer] -> [Data member] = [Pointer2] -> [Data member]

Think of it as referring to [data member] and assigning [pointer2] data member

Understanding Big-O cheat sheet:

Data Structure	Time Complexity							
	Average				Worst			
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)

Access: (related to searching) How does to access an *element @ a location in the data structure?

Search: How to loop through a data structure? Similar to access but not ever structure can access at point

Insertion: How to add an *element to a data structure

Deletion: How to add an *element to data structure

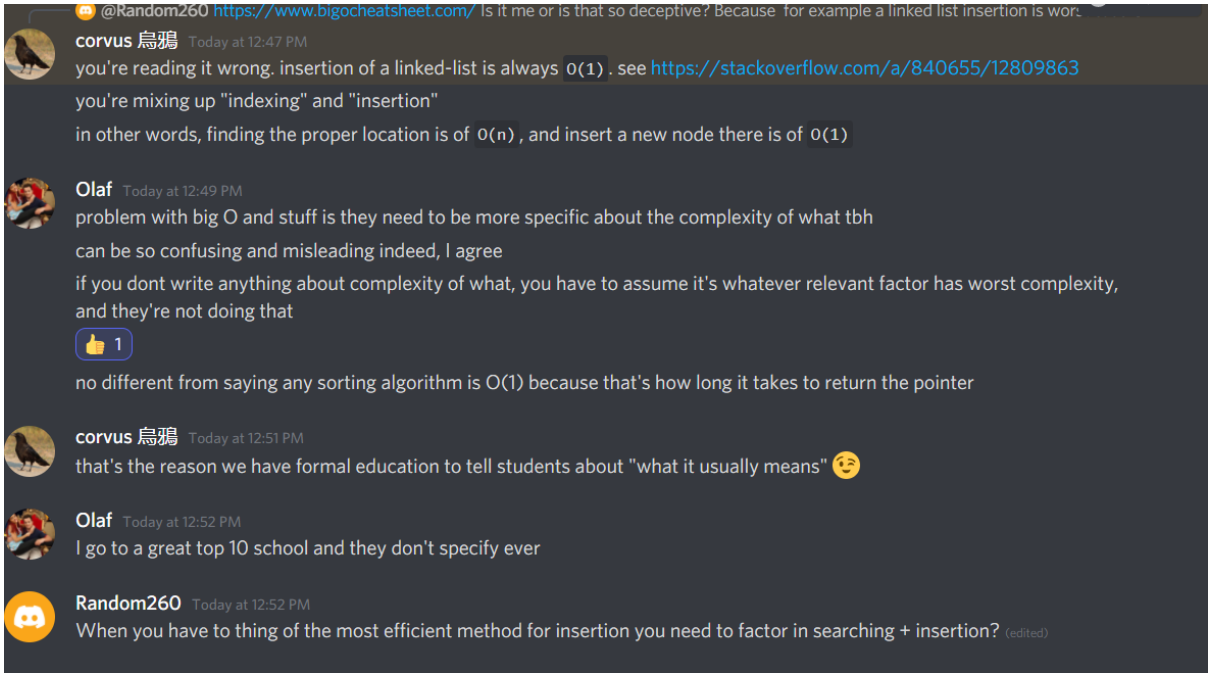
Worst means: Assume element to access/search is at the end of the data structure

So in an array, obviously accessing an element of an array is quickest O(1)

But searching is slow because we need to iterate through each element one by one

*Element/Node

Why Big(O) deceptive:




```
The insertion operation itself is O(1), but the traversal to get to the point where you want to insert is going to be O(n) worst case
```

```
Important: Consider in combination insertion + searching
```

Just realised it is. So for most cases if you want to inserts lets say in middle you need to actually "add" insertion + searching. If only there was simple maths to compare

STL algorithms vs Raw loops?

```
Algorithm looks nicer

Find() is for single thing

Loop for iterating through all elements
```