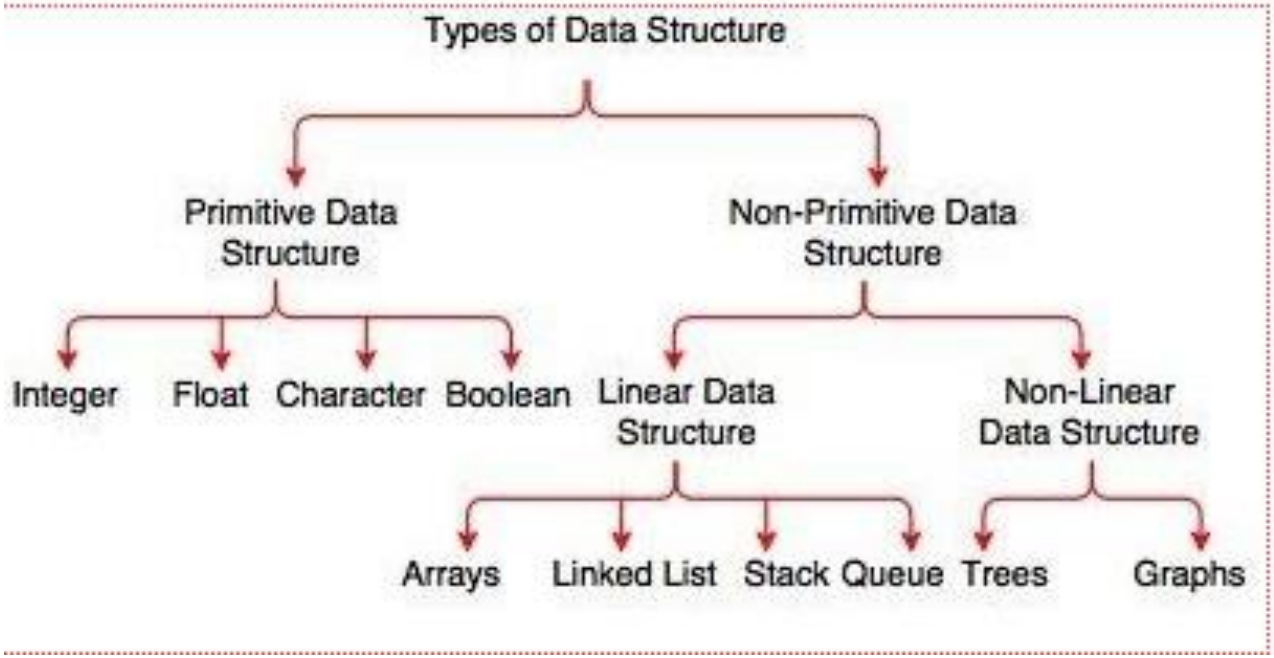
# DS-1

# Data Structures

Data Structure is a way of collecting and organizing data in such a way that we can perform operations on these data in an effective way. Data Structures is about rendering data elements in terms of some relationship, for better organization and storage. For example, we have some data which has, player's **name** "Virat" and **age** 31. Here "Virat" is of **String** data type and 26 is of **integer** data type.

We can organize this data as a record like **Player** record, which will have both player's name and age in it. Now we can collect and store player's records in a file or database as a data structure. **For example**: “Shakib" 32, “Stokes" 28, “Williamson" 29

# Types of data structures



# Continued…

Data structures can also be classified on the basis of the following characteristics

| **Characterstic** | **Description** |
|---|---|
| Linear | In Linear data structures,the data items are arranged in a linear sequence. Example: **Array** |
| Non-Linear | In Non-Linear data structures,the data items are not in sequence. Example: **Tree**, **Graph** |
| Homogeneous | In homogeneous data structures,all the elements are of same type. Example: **Array** |
| Non-Homogeneous | In Non-Homogeneous data structure, the elements may or may not be of the same type. Example: **Structures** |
| Static | Static data structures are those whose sizes and structures associated memory locations are fixed, at compile time. Example: **Array** |
| Dynamic | Dynamic structures are those which expands or shrinks depending upon the program need and its execution. Also, their associated memory locations changes. Example: **Linked List created using pointers** |

# Data Structures Operations

Data are processed by means of certain operations which appearing in the data structure. Some of the most frequently used operations are:

(**1) *Traversing:*** Accessing each records exactly once so that certain items in the record may be processed.
(**2) *Searching:*** Finding the location of a particular record with a given key value, or finding the location of all records which satisfy one or more conditions.
**(3) *Inserting:*** Adding a new record to the structure.
**(4) *Deleting:*** Removing the record from the structure.
**(5) *Sorting:*** Managing the data or record in some logical order(Ascending or descending order).
**(6) *Merging:*** Combining the record in two different sorted files into a single sorted file.

# Mathematical Notations and analysis

**Asymptotic analysis** refers to computing the running time of any operation in mathematical units of computation. For example, the running time of one operation is computed as $f(n)$ and may be for another operation it is computed as $g(n^2)$. This means the first operation running time will increase linearly with the increase in **n** and the running time of the second operation will increase exponentially when **n** increases. Similarly, the running time of both operations will be nearly the same if **n** is significantly small.

Usually, the time required by an algorithm falls under three types −

- **Best Case** − Minimum time required for program execution.
- **Average Case** − Average time required for program execution.
- **Worst Case** − Maximum time required for program execution.

# Continued…

**Asymptotic Notations**

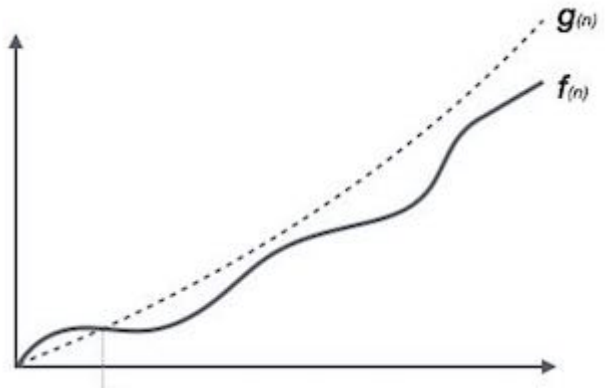
Following are the commonly used asymptotic notations to calculate the running time complexity of an algorithm.

- O Notation
- Ω Notation
- θ Notation

**Big Oh Notation, O**

The notation O(n) is the formal way to express the upper bound of an algorithm's running time. It measures the worst case time complexity or the longest amount of time an algorithm can possibly take to complete.

# Continued…



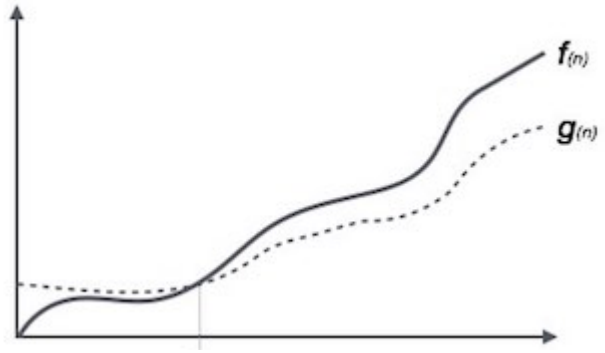

For example, for a function ***f*(n)**

O(*f*(n)) = { *g*(n) : there exists c > 0 and $n_0$ such that *f*(n) ≤ c.*g*(n)}

# Continued…

**Omega Notation, Ω**

The notation Ω(n) is the formal way to express the lower bound of an algorithm's running time. It measures the best case time complexity or the best amount of time an algorithm can possibly take to complete.

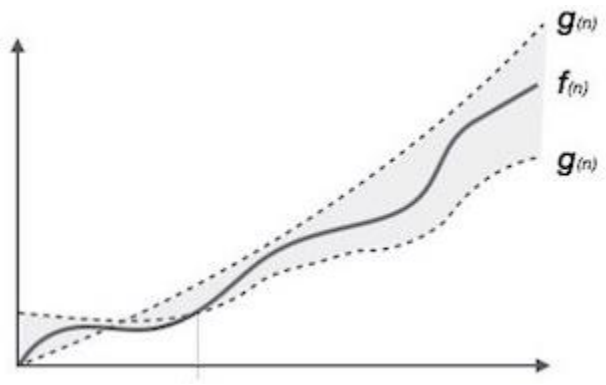



- For example, for a function ***f*(n)**
- $\Omega(f(n)) \geq \{ g(n)$ : there exists $c > 0$ and $n_0$ such that $g(n) \leq c.f(n)\}$

# Continued…

**Theta Notation, θ**

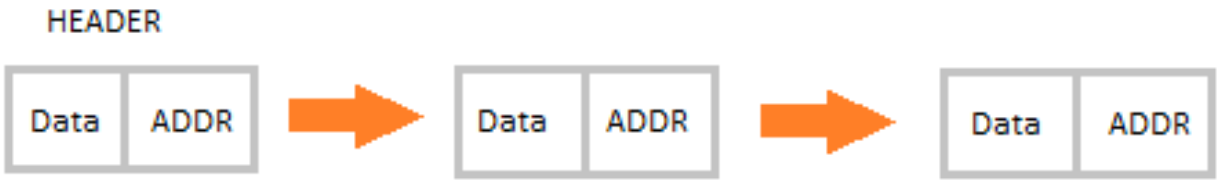
The notation θ(n) is the formal way to express both the lower bound and the upper bound of an algorithm's running time. It is represented as follows −




θ(*f*(n)) = { *g*(n) if and only if *g*(n) = O(*f*(n)) and *g*(n) = Ω(*f*(n)) }

# Linear DS: Linked Lists

- Linked List is a very commonly used linear data structure which consists of group of **nodes** in a sequence.
- Each node holds its own **data** and the **address of the next node** hence forming a chain like structure.
- Linked Lists are used to create trees and graphs.



# Continued…

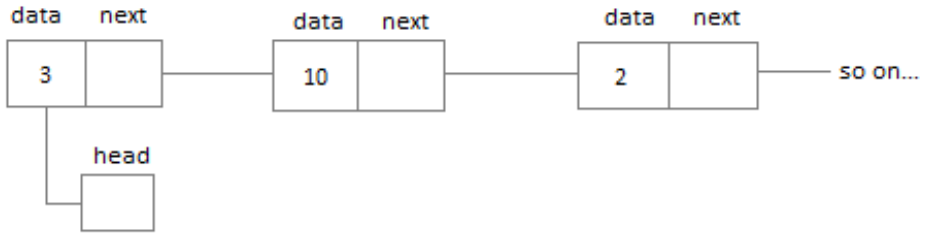
**Types of Linked Lists**

There are 3 different implementations of Linked List available, they are:

- Singly Linked List
- Doubly Linked List
- Circular Linked List

**1. Singly Linked List**

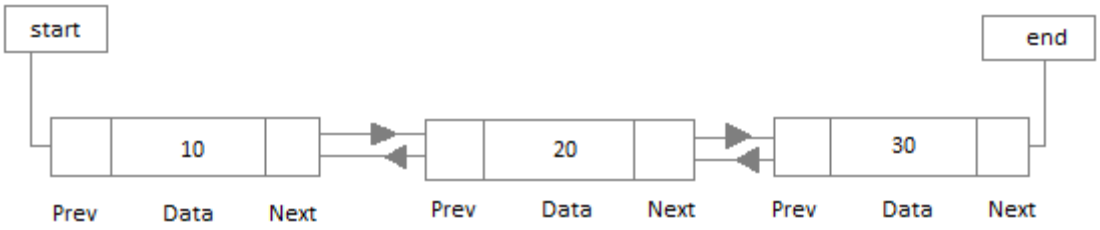
Singly linked lists contain nodes which have a **data** part as well as an **address part** i.e. next, which points to the next node in the sequence of nodes.



# Continued…

**2. Doubly Linked List**

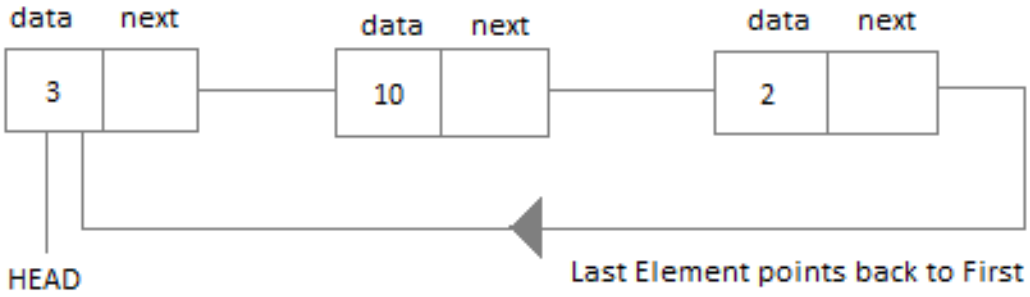
n a doubly linked list, each node contains a **data** part and two addresses, one for the **previous** node and one for the **next** node.




**3. Circular Linked List**

In circular linked list the last node of the list holds the address of the first node hence forming a circular chain.

# Continued…

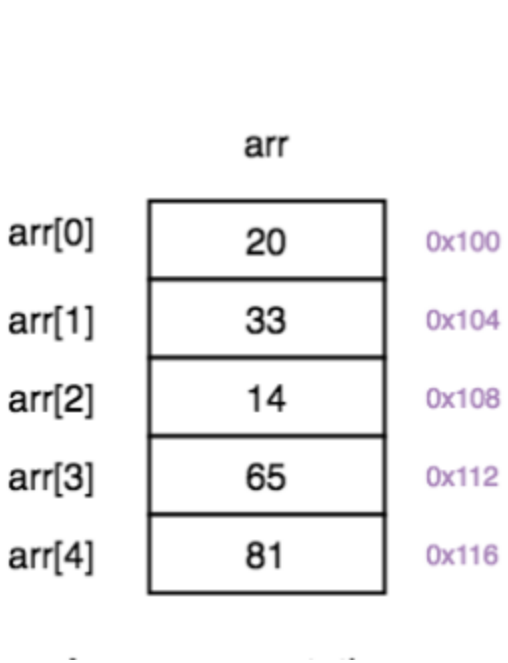

data
next
3
data
next
10
data
next
2
HEAD
Last Element points back to First

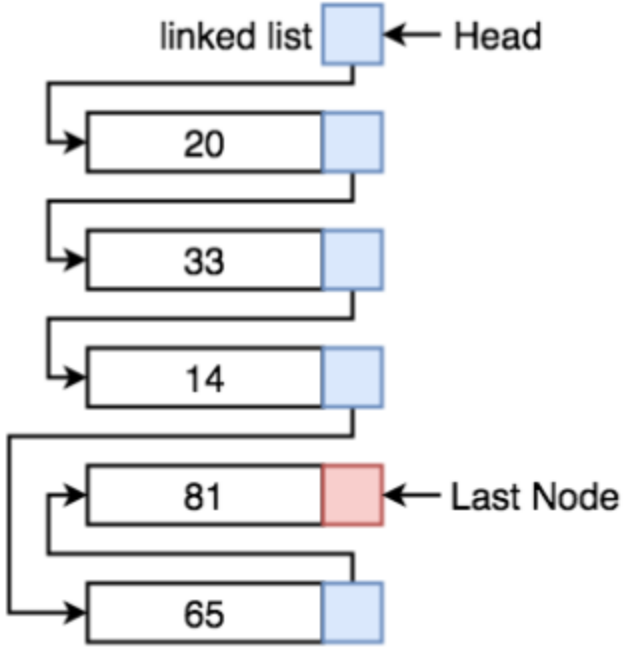
# Array VS Linked lists

In case of array, memory is allocated in contiguous manner, hence array elements get stored in consecutive memory locations. So when you have to access any array element, all we have to do is use the array index, for example arr[4] will directly access the 5th memory location, returning the data stored there.

But in case of linked list, data elements are allocated memory at runtime, hence the memory location can be anywhere. Therefore to be able to access every node of the linked list, address of every node is stored in the previous node, hence forming a link between every node.

# Continued…


arr
arr[0]
20
0x100
arr[1]
33
0x104
arr[2]
14
0x108
arr[3]
65
0x112
arr[4]
81
0x116


linked list
Head
20
33
14
81
Last Node
65