Linked List:

- **Terminologies:**
  - Linked list is a linear data structure that contains multiple records linked to each other.
  - Each item in the list is called as “Node”.
  - Each node contains data and pointer (address) to the next node.
  - Typically linked lists are implemented as self-referential structure/class. The structure/class contains pointer of the same type to hold address of next node.
- **Difference in array and linked list**

	Array	**Linked List**
1.	Array cannot grow or shrink dynamically (realloc() is not efficient).	Linked list can grow/shrink dynamically.
2.	Array has contiguous memory. Hence random access is possible.	Nodes are not in contiguous memory. Hence only sequential access is allowed.
3.	Arrays do not have memory overheads.	Linked lists have memory overheads e.g. each node contains address of 'next' node.
4.	Insert/Delete at middle position need to shift remaining elements. Hence will be slower.	Insert/Delete at middle position need to modify links (next/prev pointers). Hence will be faster.
5.	To deal to fixed number of elements and frequent random access, arrays are better.	To deal to dynamic number of elements and frequent insertion/deletion operators, linked lists are better.

- **Types of Linked Lists:**
  - **Singly Linear Linked List**
    - **A. Singly Linked List – only head pointer:**
      1. **Add First:O(1)**
      2. **Add Last: O(n)**
      3. **Insert at position: O(n)**
      4. **Delete First: O(1)**
      5. **Delete At Position: O(n)**
      6. **Delete All: O(n)**

    - B. **Singly Linked List –head and tail pointer:**
      1. **Add First: O(1)**
      2. **Add Last: O(1)**
      3. **Delete First: O(1)**

- **Singly Circular Linked List:**

  Last node’s next pointer keeps address of first (head) node.
  1. **Add First: O(n)**
  2. **Add Last: O(n)**
  3. **Insert at position: O(n)**
  4. **Delete First: O(n)**
  5. **Delete At Position: O(n)**

- **Doubly Linear Linked List**

  Each node contains data, address of next node and address of previous node.
  1. **Add First: O(1)**
  2. **Add Last: O(n)**
  3. **Insert at position: O(n)**
  4. **Delete First: O(1)**
  5. **Delete At Position: O(n)**
  6. **Delete All: O(n)**

- **Doubly Circular Linked List**

  Each node contains data, address of next node and address of previous node. Last node’s next contains address of first node, while first node’s prev contains address of last node.
  1. **Add First: O(1)**
  2. **Add Last: O(1)**

	Singly Linked List with Head	**Singly Linked List with Head & Tail**
1.	Add last operation is slower i.e. O(n)	Add last operation is faster i.e. O(1)
2.	No overload.	Extra overhead i.e. tail pointer.
3.	Simplified list manipulation operations.	List manipulation need to consider tail pointer wherever appropriate.

	Singly Linked List	**Doubly Linked List**
1.	Unidirectional access.	Bi-directional access.
2.	For search and delete operation two pointers are required (Need to maintain prev pointer) during traversing.	For search and delete operation only one pointer is required (No need to maintain prev pointer) during traversing.
3.	Simplified list manipulation operations.	List manipulation need to consider prev pointer of each node in all operations.

	Doubly Linked List	**Doubly Circular Linked List**
1.	Traversing till last node is slower (add last operation) i.e. O(n)	Traversing till last node is faster (add last operations) i.e. O(1)