(1) Linked List Data Structure

↳ Linear Data Structure

* A linear dataStructure arranges the data in a sequential order, where elements are connected one after another.

Ex:- (i) Array
(ii) Linked List
(iii) Stack
(iv) Queue

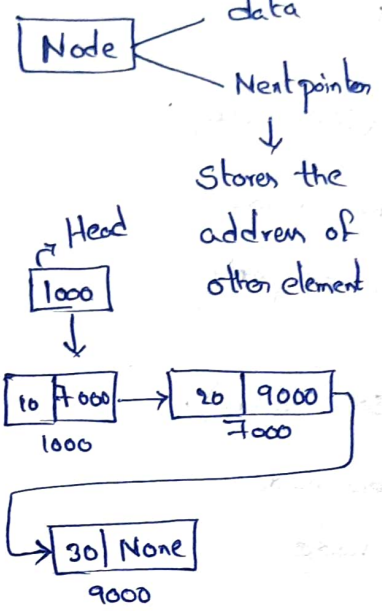
Non Linear Data Structure

* It organises data in a Hierarchical or interConnected way. One element can be connected to multiple elements.

Ex:-
(i) Tree
(ii) Graph
(iii) Heap
(iv) Trie

Linked List

(i) No Contiguous memory allocation



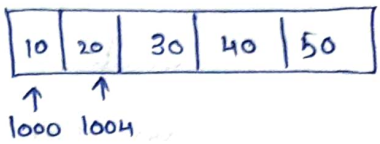
(ii) Linked List coill always be None

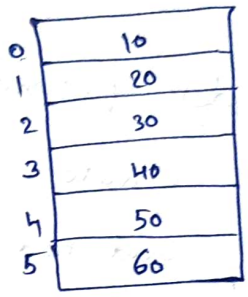
(iii) Random access is not possible

(iv) Insertion/Deletion ↓ frequent
Linked List

Dynamic Array

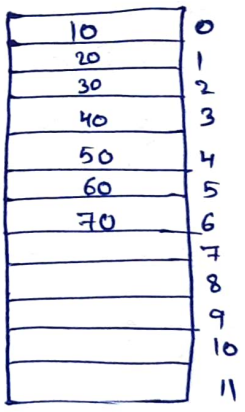
(i) Contiguous memory allocation





DA-1

If we want to add new element to the array then we double the array



(vi) Searching Operation ↓ frequent
Array
(Random Access)

class LinkedList :

① def append (self, value) :
    create a new node
    add node to end

② def prepend (self, value) :
    Create a new node
    add node to beginning

③ def insert (self, value) :
    create a new node
    and insert node

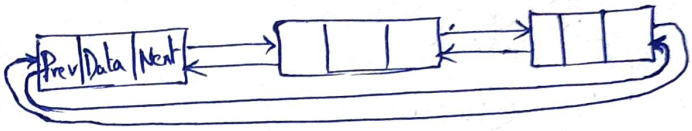
⋮

Types of Linked List

(i) Singly Linked List

(ii) Doubly Linked List

(iii) Circular Linked List

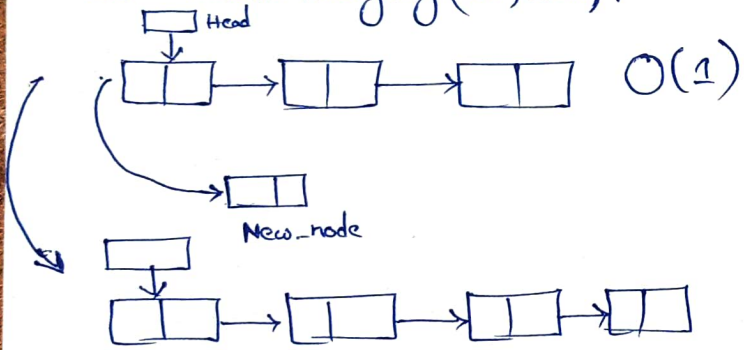


Linked List Code

```
Class Node :
    def __init__(self, data):
        self.data = data
        self.nextPtr = None

class LinkedList:
    def __init__(self):
        self.head = None
```

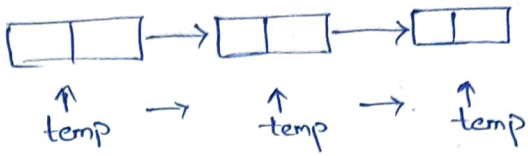
```
def insertAtBegining(self, data):
```




```
    # new node next pointer should point
      to the head
    # Head should point to the new
      node
    new_node = Node(data)
    new_node.nextPtr = self.head
    self.head = new_node
```

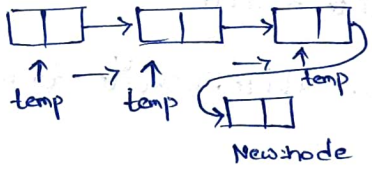
```
def printLinkedList(self):
```

→ O(n)




```
    # initialize temp to head
    # Print temp.data
    # move temp pointer → temp = temp.nextPtr
    # Repeat this till temp is not None
    temp = self.head
    while temp:
        print(temp.data)
        temp = temp.nextPtr
    return
```

```
def insertionAtEnd(self, data):
```

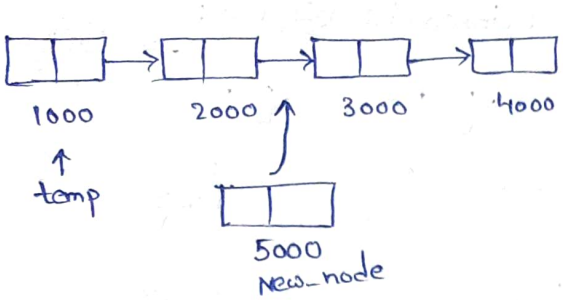


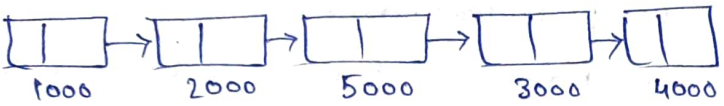
→ O(n)

```
    # Initialize temp to head
    # Run loop till temp.nextPtr is not
      none.
    # When temp reaches final node, update the
      temp.nextPtr to new node & update temp
      to new node.
    temp = self.head
    while temp.nextPtr is not None:
        temp = temp.nextPtr
    temp.nextPtr = new_node
    temp = new_node.
```

def insertAfterNode (self, Address, data) :

↳ O(n)





initialize temp to head

move temp till ~~next~~ temp.nextPtr is not equal to given address

If it is equal to given address, exit the loop

update new_node nextPtr to temp.nextPtr.nextPtr

And temp.nextPtr.nextPtr to new_node

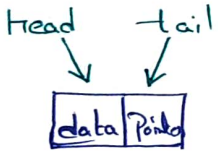
```
temp = self.head
while temp.nextPtr ≠ Address :
    temp = temp.nextPtr
new_node.nextPtr = temp.nextPtr.nextPtr
temp.nextPtr.nextPtr = new_node
```

Linked List - Code

↳ With Tail pointer

```
class Node :
    def __init__(self, value):
        self.data = value
        self.nextPtr = None
```

```
class LinkedList :
    def __init__(self, value):
        self new_node = Node(value)
        self.head = new_node
        self.tail = new_node
        self.length = 0
```



(i) Append a node

```
def append(self, value):
    self new_node = value Node(value)
    else: self.tail.nextPtr = new_node
          self.tail = new_node
          self.length += 1
    if self.head is None:
        self.head = new_node
        self.tail = new_node  self.length = 1
```

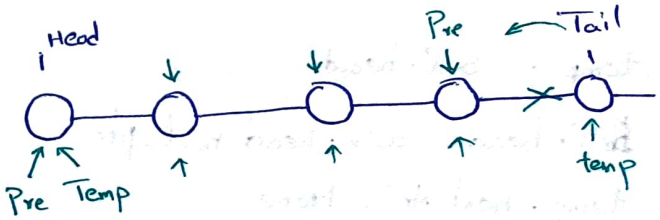
(ii) Pop a node from end

Edge Cases

(i) Empty Linked List

(ii) Only Single Element

```
def pop(self):
    if self.length == 0:        # self.head is None
        return None             # → No Elements
    temp = self.head
    pre = self.head
    while (temp.next):          # → (or) temp.next is not None:
        self.pre = temp
        temp = temp.next_Ptr
    self.tail = pre
    self.tail.next_Ptr = None
    self.length -= 1
    if self.length == 0:        # After popping Last element
        self.head = None
        self.tail = None
    return temp
```



(iii) Prepending

```
def prepend (self, value):
    self.new_node = Node(value)
    if self.length == 0:
        self.head = new_node
        self.tail = new_node
    else:
        new_node.next = self.head
        self.head = new_node
    self.length += 1
    return True
```

(iv) Pop First

Edge case

(i) Only one node

(ii) No nodes

```
def pop_first (self):
    if self.length == 0:
        return None
    temp = self.head
    self.head = self.head.next_ptr
    temp.next_ptr = None
    self.length -= 1
    if self.length == 0:
        self.tail = None
    return temp
```