DAY-20

Sorting Linked List Using Merge Sort.

Why Merge Sort for Linked List:-

-> linked list nodes may not be adjacent in memory.

-> merge operation of merge sort can be implemented without extra space for linked list.
['coz the interchange @(1) time and space)

-> we can not do random access in linked list

-> Merge sort accessed data sequentially and the need of random access is low.

* Merge sort is one of the most famous divide-and-conquer sorting algorithms.

-> can be used to sort values in any traversable DS.

Algorithm :-

1. If : the list containg one or fewer elements, return the same list
2. Else : Divide the list into two halves using the splitting func.
3. Sort : sort the two halved of the list.
4. Finally, merge the sorted list.

Code :-

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

def printList(head):
    ptr = head
    while ptr:
        print(ptr.data, end = " ")
        ptr = ptr.next
    print("None").
def SortedMerge(a, b):
    if a is None:
        return b
    elif b is None:
        return a
    if a.data <= b.data:  # for
        result = a        # recursion.
        result.next = SortedMerge
                      (a.next, b)
```

```
    else:
        result = b
        result.next = SortedMerge(a, b.next)
    return result
"here, we will use fast/slow pointer strategy"
def FrontBackSplit(source):
    if source is None or source.next is None:
        return source, None
    slow, fast = source, source.next
    while fast:
        fast = fast.next
        if fast:
            slow = slow.next
            fast = fast.next
    ret = soure, slow.next
    slow.next = None
    return ret
def MergeSort(head):
    if head is None or head.next is None:
        return head
    front, back = FrontBackSplit(head)
    front = MergeSort(front)
    back = MergeSort(back)
    return SortedMerge(front, back)
```

* slow/fast pointer strategy:
nothing but; "slow" pointer travels the linked list one node at a time whereas 'fast' pointer travels the linked list two nodes at a time.