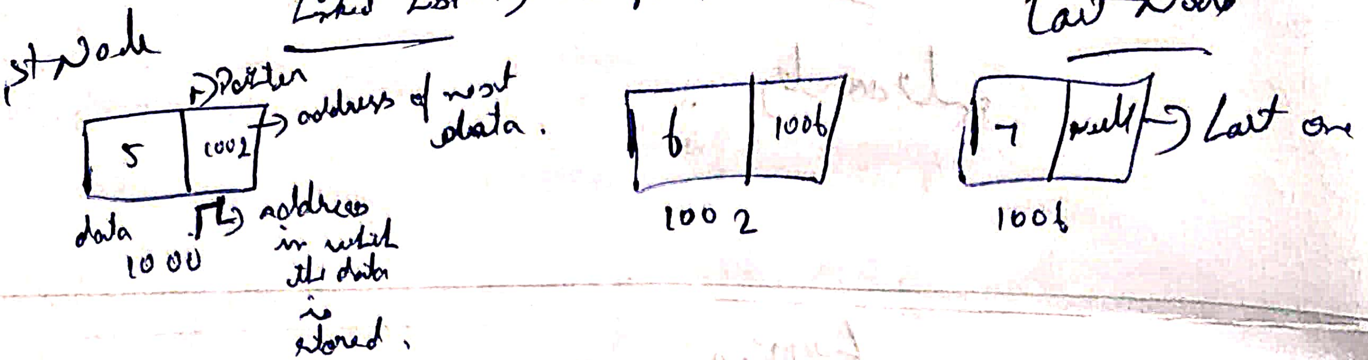
→ Property ⇒ Opposite to ~~list~~ array.

Linked List → Multiple list linked



Normal list

num = [5, 7, 8, 9].

print (id (num[0])) → whenever runs, it changes frequently as it is getting memory from RAM.

next index element id ⇒ difference ⇒ 32
⇒ 4 bytes.

But in linked list.

different id's but all are linked based on the address stored.

Node

First one ⇒ Head

Last ⇒ Tail.

Operation:
* Insertion
* Deletion.

~~Ops~~ diff. datatypes included.

	Start	Middle	End
Insert	O(1)	O(n)	O(n)
Delete	O(1)	O(n	O(n)
Search	O(n)		

Linked list can be effectively used in OOPS.

Time Complexity & Space Complexity

Big O notation:

i) O(1) → Constant

ii) O(n) → Linear

iii) O(log n) → Logarithmic ⇒ Two pointer technique → Binary search

iv) O(N log N) → Linear logarithmic → Merge Sorting.

v) O(n^2) → Quadratic.

i) print("Hi") → constant O(1) → No. of times.

(or) a=1, b=2 print(a+b)

ii) l = [1,2,3,4,5] → O(n)

```
for i in l
    print(i)
```

v) O(n^2).

```
l = [1,2,3,4,5]
for i in l
    for j in l
        print(i,j)
```

$n^2 = 5^2 = 25$

↳ 25 Combinations comes

Space Complexity.

↳ Space occupied by program.

* O(1) ⇒ Constant space. (Note: Same datatype, same variable)
  a=10. ⇒ Even a=100 occupies same size.

* O(n) Linear.
  l=[1,2,3,4,5]
  ↳ based on values in list it occupies space.

Both Constant + Linear.
a=10
b=20
c=[1,2,3]
↳ Highest space occupied ⇒ Take it. O(n)

* O(log N) ⇒ Recursion ⇒ Stack.

* O(n^2) ⇒ Matrix form of array
  rows + Column
  l = [[1,2,3]
  [1,2,3]
  [1,2,3]]

Auxiliary space

l=10 → give O(1)
K=[ ] ← but we will take O(n)
for i in range(1, l+1): This is auxiliary space.
K.append(i).
Max. is O(n).

O(1) → Efficient
O(log n)
O(n)
O(n log n)
O(n^2)

Arrays - K = [1, 2, 3, 4]. Based on this -> Time complexity + Space complexity

* Access

O(1)

K[2] -> based on index

* Insert

when added to one position, the elements in that position shift to next position

1st, middle, O(n - i)

Last index O(1)

* Delete

After deleting 1 value, all the values shift to 1 position before

1st & middle.

O(n - i)

Last O(1)

* Search.

O(n).

Worst case always consider

[1, 2, 3, 4, 5]

check 5 in list.

-> worst case.

-> last position.

best case [5, 1, 2, 3, 4]

Searching algorithms.

* Linear Search

```
l = [1, 2, 3, 4].
x = 3
for i in l:
    if i == x
        print i.
```

normal

If we want index?

* Binary Search.

[1, 2, 3, 4, 5]

Search for the value in list, if it is present in left, middle (or) right..
then again divide & see where it is
eliminate other values.

Recursion -> base condition should be there -> to stop iteration.

factorial.

n = 5

```
def fact(n)
    if n == 1:
        return 1
    return n * fact(n-1).
```

have to give this condition, then only it will stop

5 × 4!

5 × 4 × 3!

5 × 4 × 3 × 2!

5 × 4 × 3 × 2 × 1!

n × (n-1)!

It will be stored in Stack.

1
2
3
4
5

1 <- condition. 1! = 1.
2 × 1!
3 × 2!
4 × 3!
5 × 4!

-> then 1 × 2 × 3 × 4 × 5 = 120 [120]