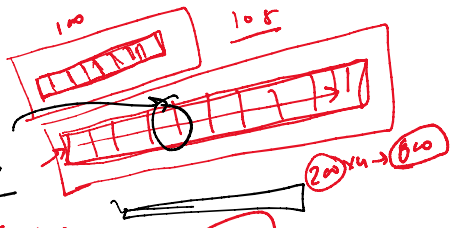
LinkedList
Thursday, August 5, 2021 8:25 PM

→ Limitation of an Array



① Fixed Size → Overflow

② Memory waste / Memory Issue

③ Heterogenous Data Not support

→ ④ Insertion and Delete operation from a specific location are time consuming

O(n)

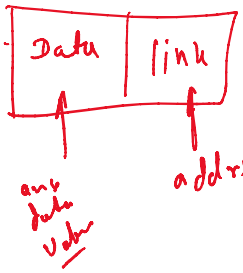
⑤

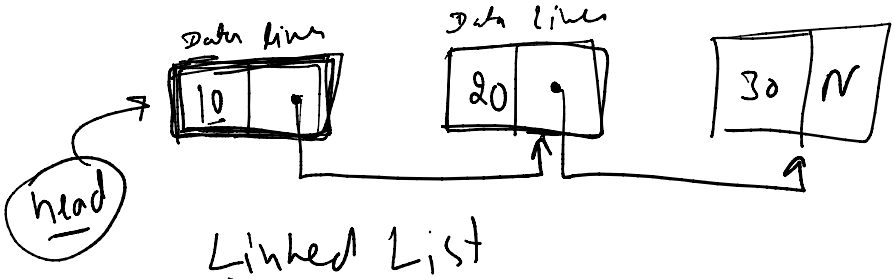
⇒ Linked List

Node → class / struct

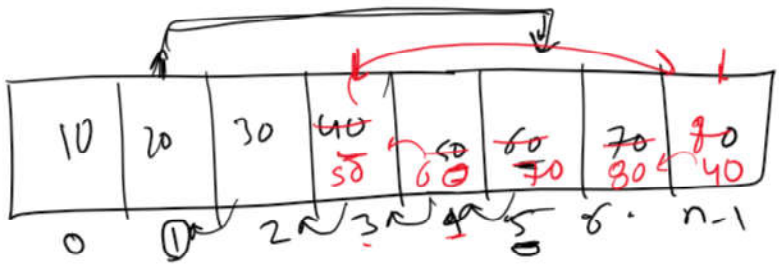
Holds data and reference (link)

Normal term





Linked List



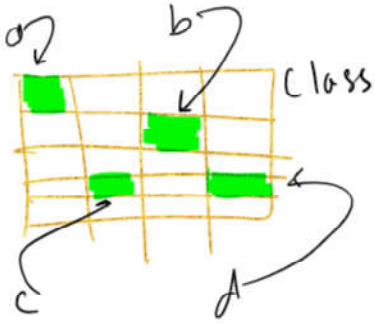
shift(old, new) →

```
item = arr[old]
while old < new:
    arr[old] = arr[old+1]
    old = old + 1
arr[old] = item
```

shift(3, 7)
old = 7
new = 7
item = 40

Linked List

① Node

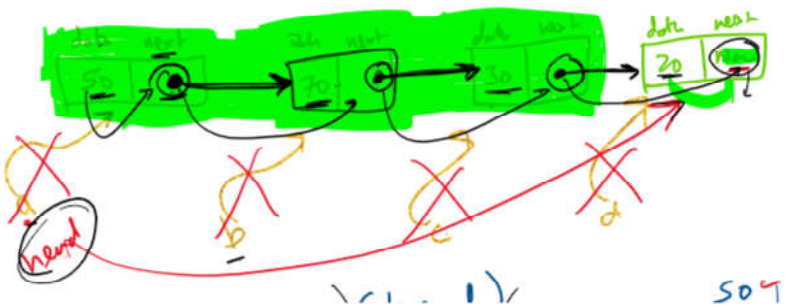



```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

    def __str__(self):
        return f"Node({self.data})"
```

```
a = Node(50)
b = Node(70)
c = Node(30)
d = Node(20)

head = a
head.next = b
```



head = a
head.next = b
b.next = c
c.next = d

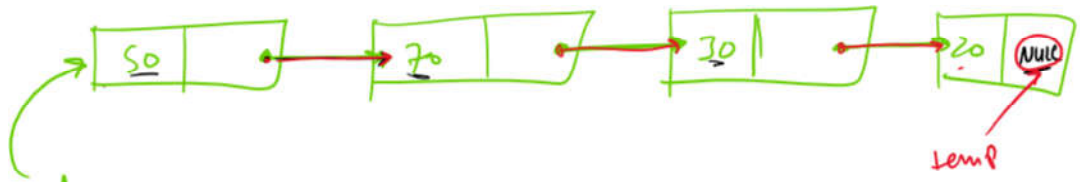
del a, b, c, d

2. temp = head

DISPLAY (head)

while head.next != NULL
do
print head.data
head = head.next
done
print head.data

50
70
30
20

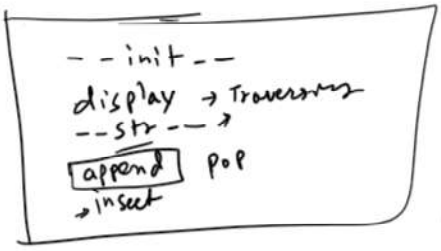


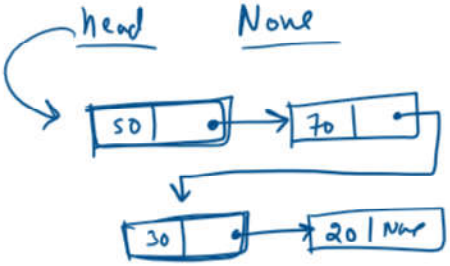
50
70
30
20

DISPLAY (head)

temp = head
WHILE temp != NULL
DO
print temp.data
temp = temp.next
DONE

Class LinkedList





mylist = LinkedList()

mylist.append(Node(50))
mylist.append(Node(70))
mylist.append(Node(30))
mylist.append(Node(20))

50
70

mylist.append(Node(70))

mylist.append(Node(20))

mylist.display()

70
30
20

(i) Linked List isEmpty

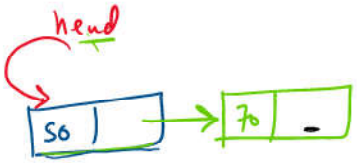
O(1)

APPEND (head, node)

head → NULL

if head == NULL:

head = node



else:

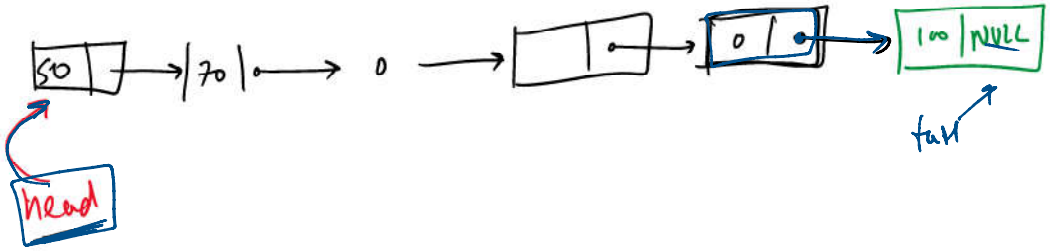
temp = head

while temp.next != NULL:

temp = temp.next

temp.next = node

O(n)



tail.next = node

tail = node

Append (tail, node)

head → NULL

tail →

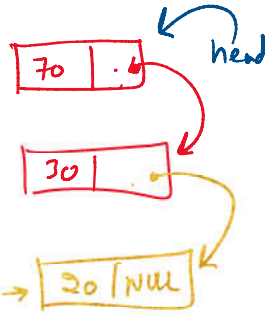
if tail == NULL:

head = tail = node

else:

tail.next = node

tail = node



→ pop

→ search - delete

→ insert → after - item
→ before - item

x = [10, 20, 30]

y = x

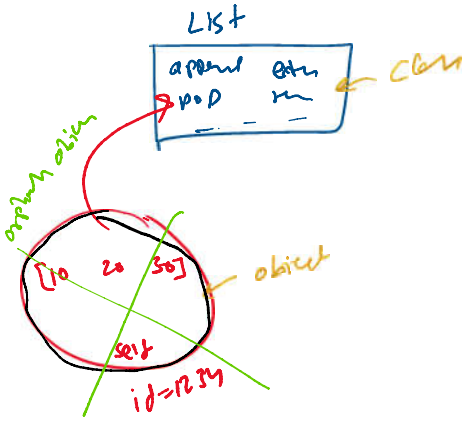
del x

del y

reference

↓

X



a = Node(30)

b = Node(50)

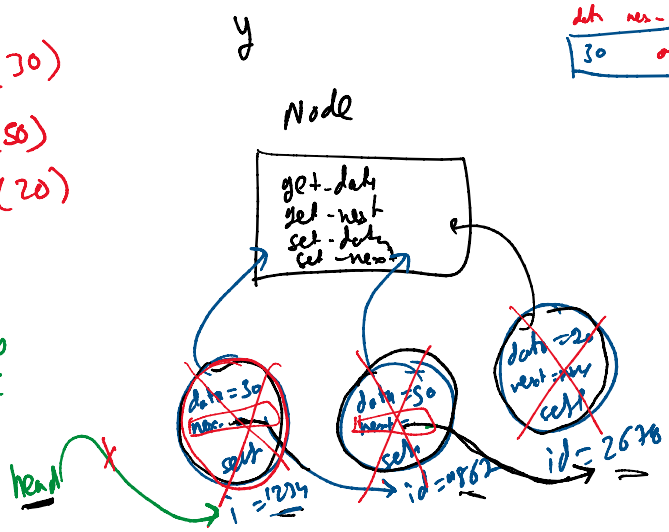
c = Node(20)

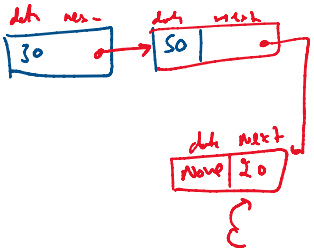
head = a

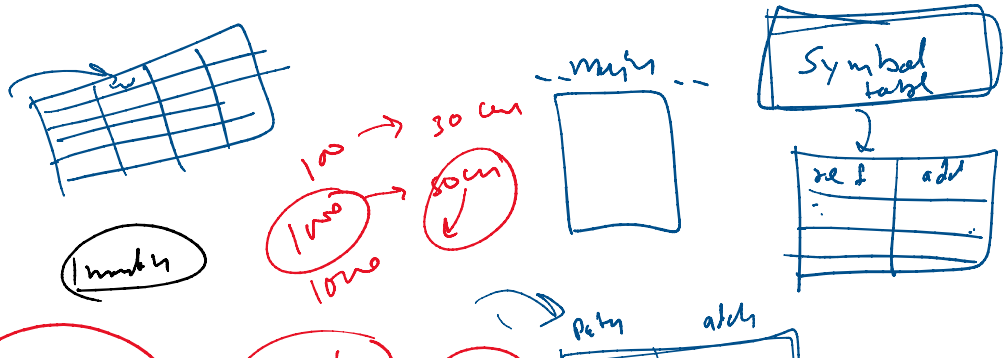
a.next = b

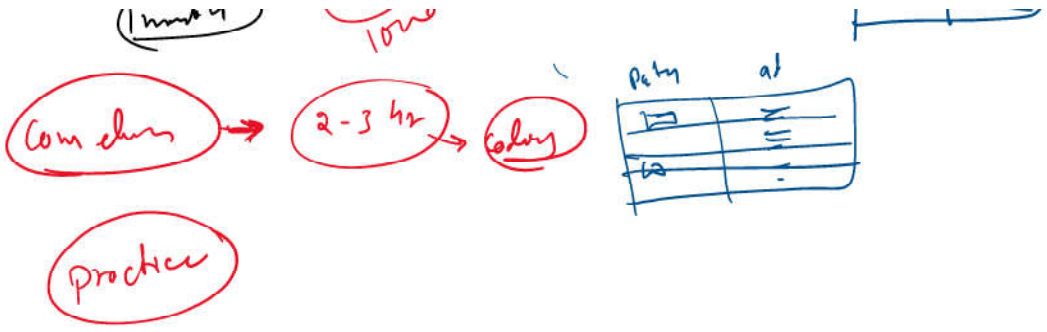
b.next = c

del a, b, c





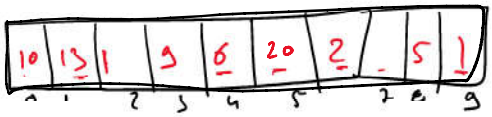




Constraint →

Even num == odd Num

i/p



W-P to rearrange array items such that

all even numbers should be on even indices

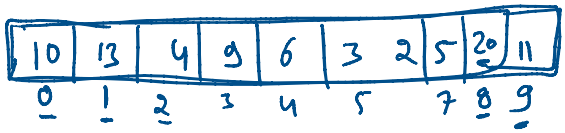
and all odd numbers should be on odd indices

Time so√.

O(2) ✓

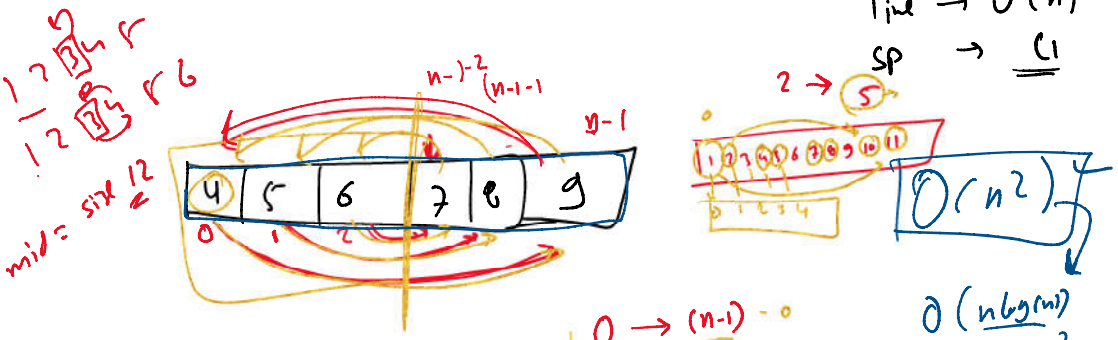
O(n)

o/p



Time → O(n)

SP → C1



(arr, size)

O(nlog(n))

0 → (n-1) - 0

1 → (n-1) - 1

2 → (n-1) - 2

3 → (n-1) - 3

n/2

Maths

n(n+1)/2

O(1)

s = 0

for i in arr:

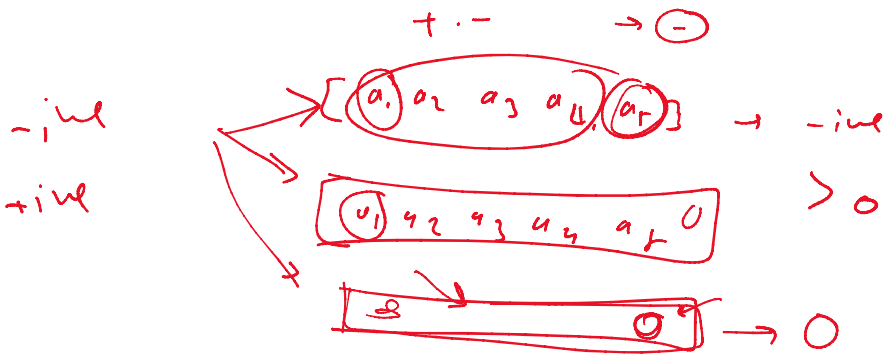
s += i

O(n)

print(s)

$S_1 = [a_1, a_2, a_3, a_4] \rightarrow a_1 \cdot a_2 \cdot a_3 \cdot a_4 < 0$

+ · − → −



$-\infty < a < \infty$

n

3

(-1) (2) (0)

n

1	-1
1	2
1	0

S_1 [-1] → 1 → n_1

S_2 [2] → 1 → n_2

S_3 [0] → 1 → n_3

4

-1 -2 -3 0

1	-1
2	-2 -3
1	0

$S_1 = [\ -1 \quad] < 0$

$S_2 = [\ -2 \ -3\ ] > 0$

$S_3 = [\ 0 \quad] = 0$

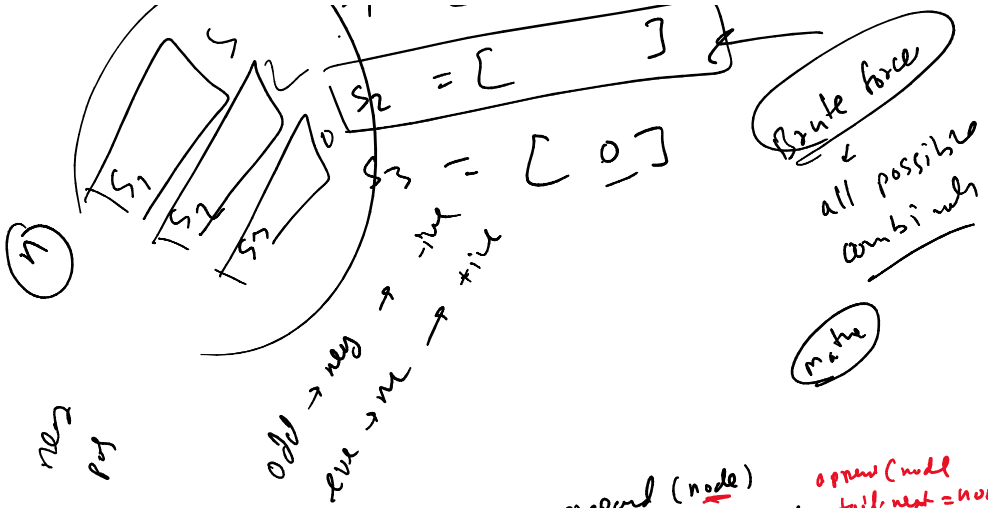
0

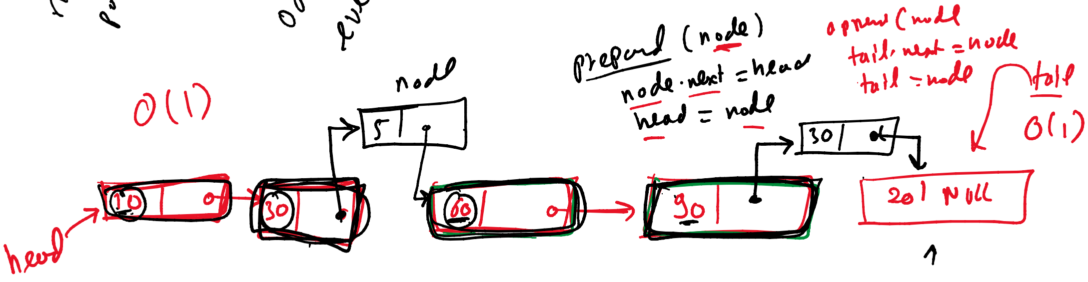
$S_1 = [-1, -2, -3]$

$n_1, n_2, n_3 > 0$

$S_2 = [\quad]$

force





Insert_before (head, data, node)   Insert_after (head, data, node)

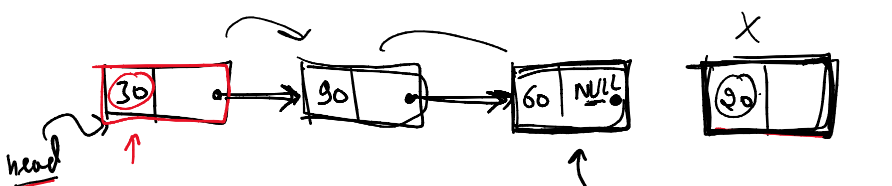
Singly Link list

HackerRank Interview

Questions → discuss
array
link list

→ delete last Node

O(1)

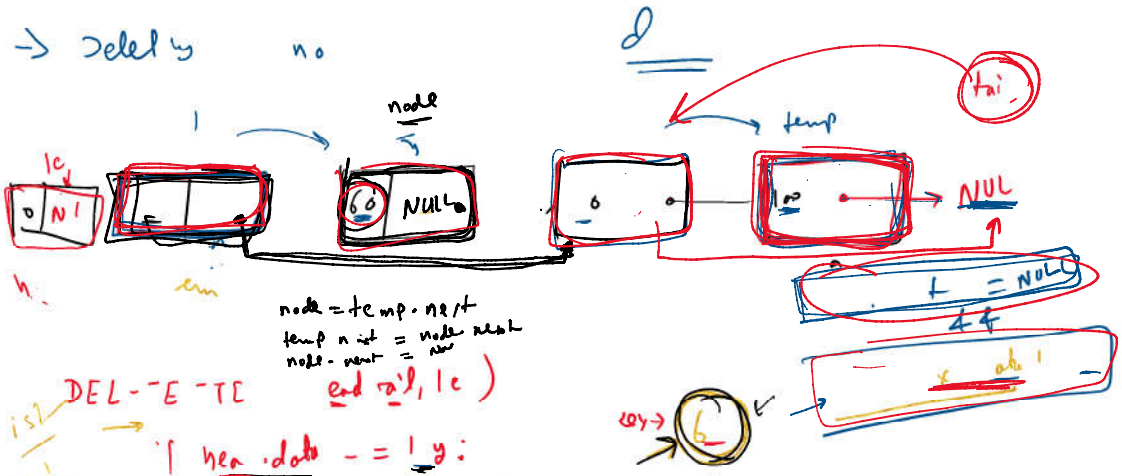


pop (

Sets

undo

→ Delete by no



DELETE (head, tail, key)

IF is

if head.data == key:

temp = head

head = head.next

if head == NULL:
tail = NULL

else:

temp = head

WHILE temp.next != NULL and temp.next.data != key

DO

temp = temp.next

DONE

if temp == tail:

return -1 (item not found)

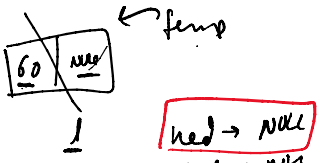
if temp.next == tail:

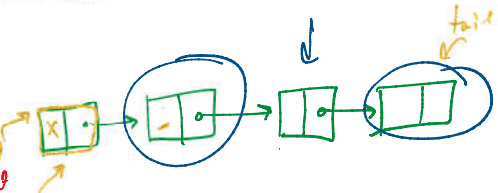
tail = temp

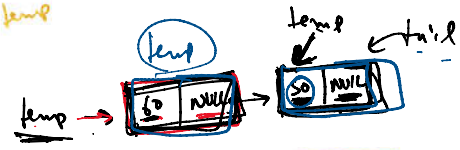
node = temp.next

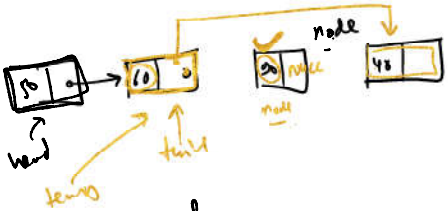
temp.next = node.next

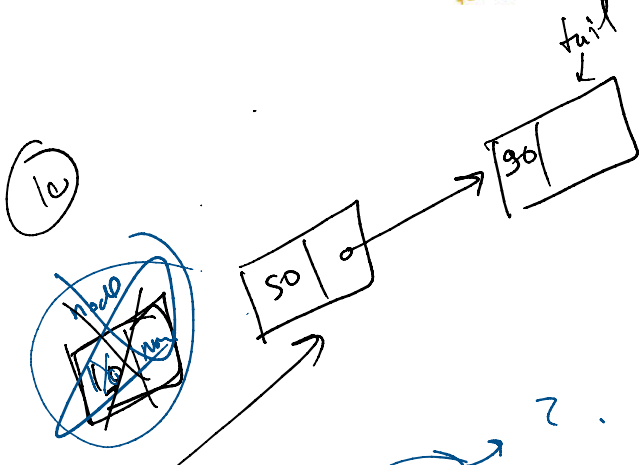
node.next = NULL

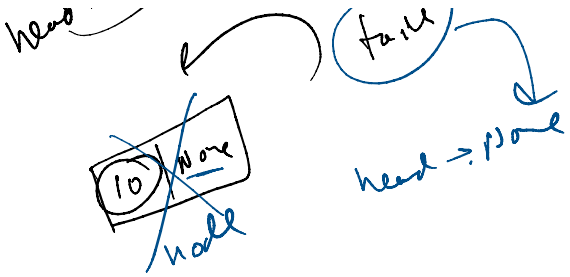


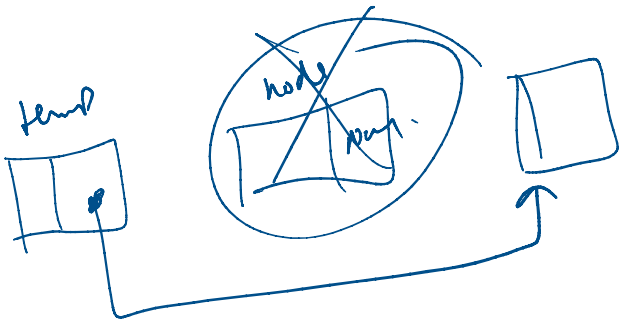




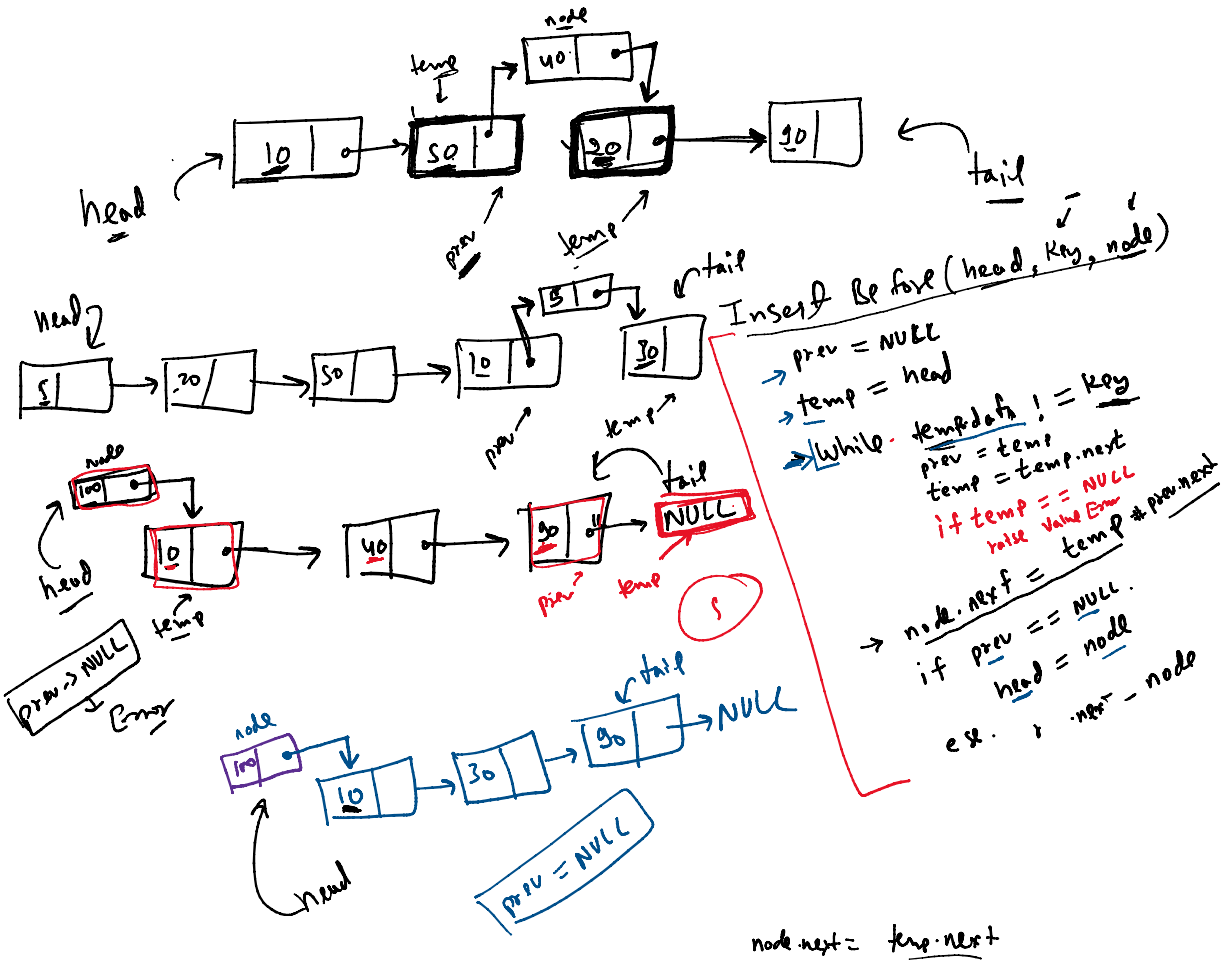








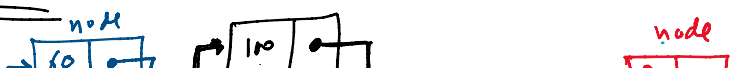
→ Insertion Before Element



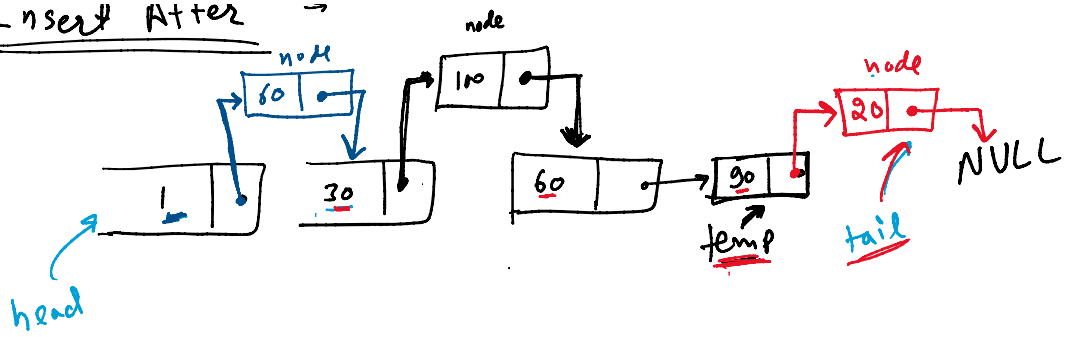

```
InsertBefore(head, Key, node)
→ prev = NULL
→ temp = head
→ while temp.data != key
      prev = temp
      temp = temp.next
      if temp == NULL
          raise ValueError
→ node.next = temp
→ if prev == NULL:
      head = node
  else:
      prev.next = node
```

node.next = temp.next
temp.next = node

→ Insert After →



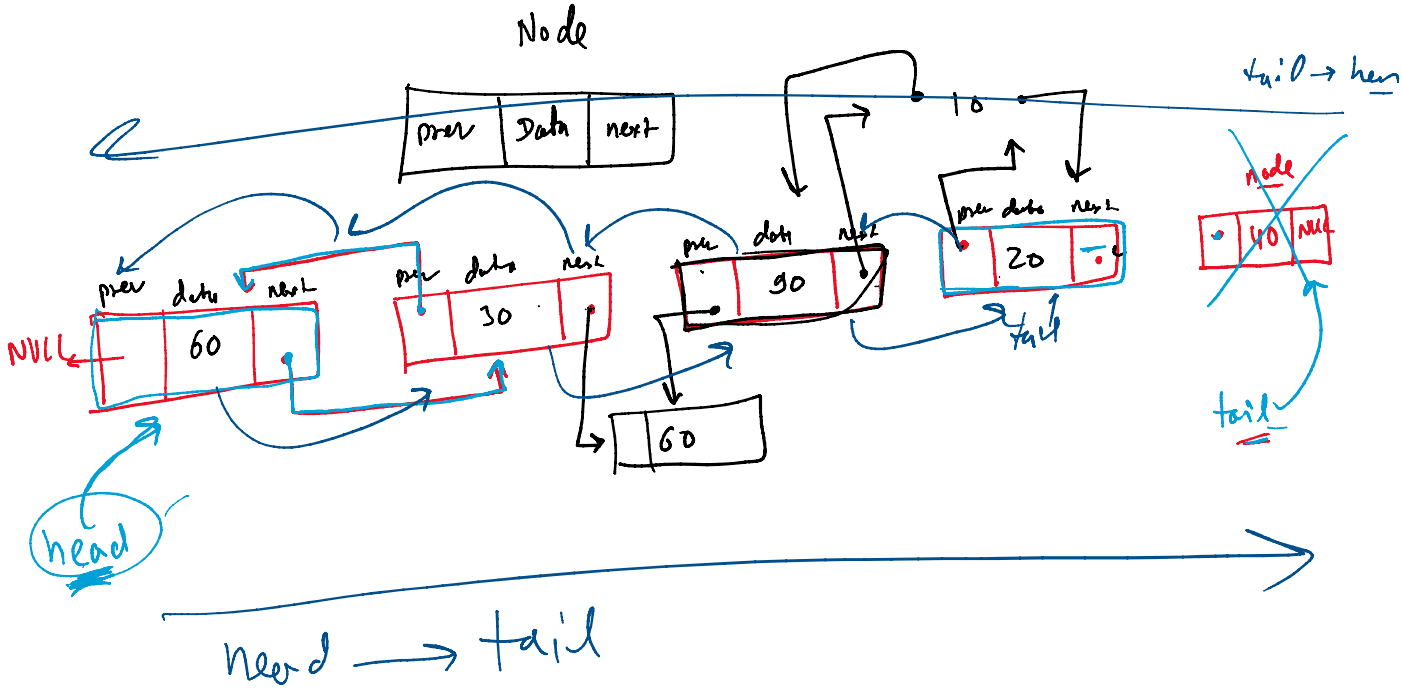
→ Insert After →

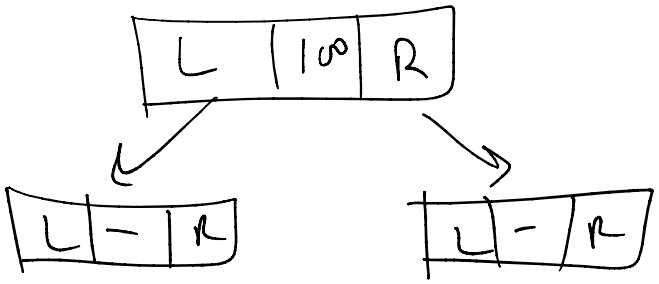


Insert_After (head, tail, key, node)

```
temp = head
while temp.data != key
        temp = temp.next
        if temp == NULL
            raise ValueError
→ node.next = temp.next
→ temp.next = node
    if temp == tail
        tail = node
```

Doubly LinkList





→ Infty → Array
Linklist

→ Hackerlen → Array
Linked list 2

→ other