(1) Write the entire code by yourself again
- ✓ by reference
- ✓ by return pointer/void

(2) write a function to find the length of the LL ✓

(3) Insert at a given position (a) ✓

(4) find mid node : single iteration (d)

(5) Search a value in LL. & recursive (b) ✓

* If the LL has values in sorted order, what is the

© best searching algorithm?

— Complexity of Binary Search on LL.

Delete

Reverse a LL

head

Linked List

data

node

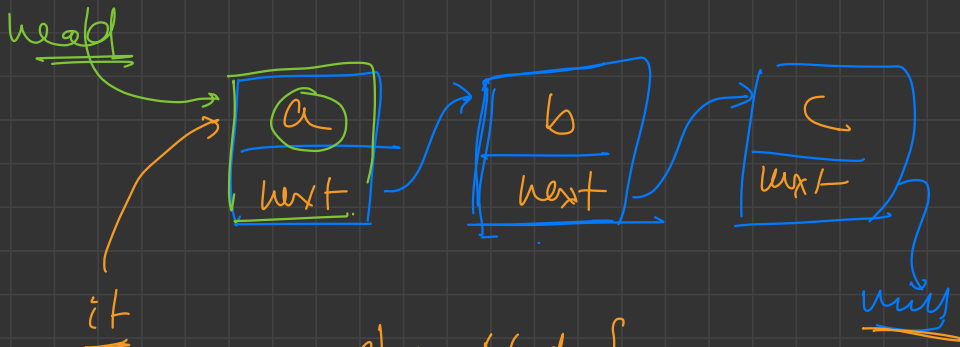
pointer to the next node

denote the end of LL

arr[i] can't iterate in a LL

Iterate int* ptr = &x;

Node* it = head



it → points to the entire object containing a

```
class Node{
    int val;
    Node* next;
};
```

(*it) → a

(*it).val

(*it).next : is the address of the block after the block which it is pointing to

Node * it = head;

after last block

while (it != NULL) {

cout << (*it).val << ' ';

it = it -> next;

}

address of next element

stop after the last block

last block

* it -> next != null

stop at the last block

(i < n) print

(i < n-1)

i = 0

while (i < n) {

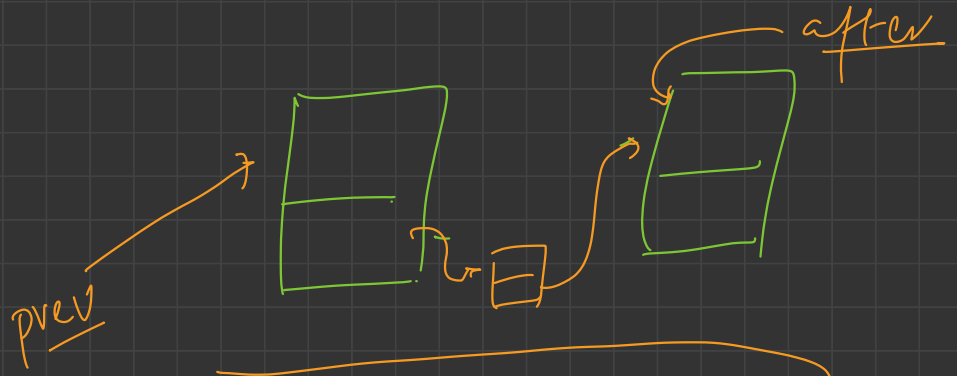
cout << a[i];

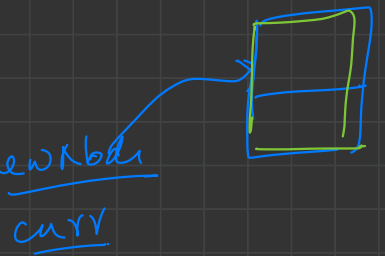
i++

}

i = (i+1);

denotes the index of next element.





① ② ③

* prev → next should be newNode ←

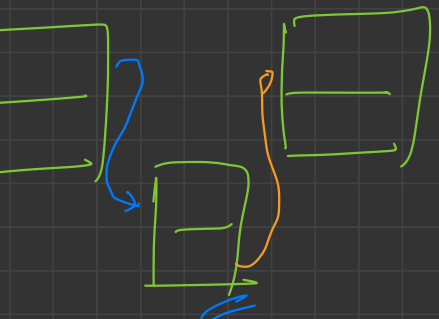
* newNode → next should be after ←

① ② ③ ④

```
after = prev->next;
prev->next = newNode;
newNode->next = after;
```

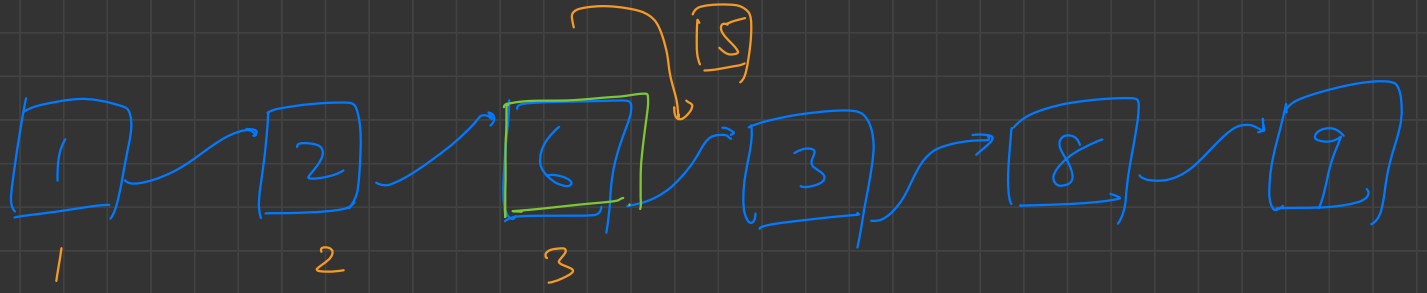
you can interchange

```
prev->next = newNode
newNod->next = ??
```

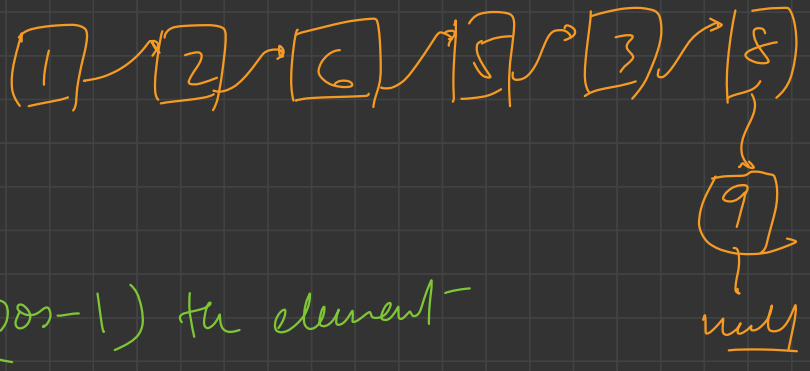



```
newNode->next = prev->next
prev->next = newNode
```

to insert newNode after prev

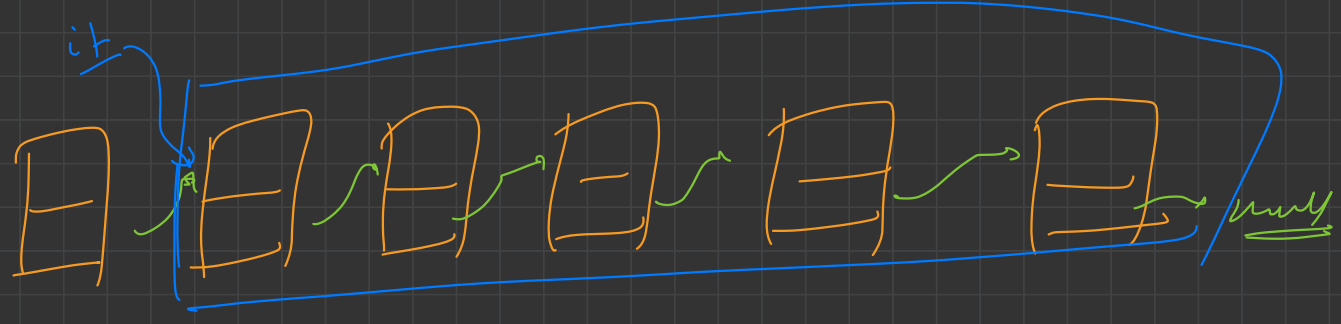


insert (position = 4, val = 5)



pointer to the (pos-1) th element

null



n

```
if( it == NULL) => false
else { it points to a valid node
    if( it -> val == x)
        return true;
return search( it -> next, x);
}
```

Best Searching methods in LL.

- ~~O(nlogn)~~
- linear: O(n)
- binary: ~~O(nlogn)~~
- O(logn) → binary search

arrays

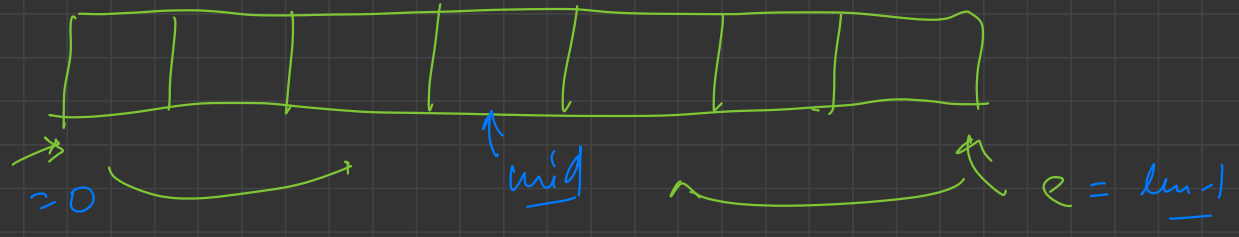
linear: O(n)

binary: O(log n)

O(n) <= Ok logn)
>=

LL → linear search: O(n)

↳ possibility: Binary Search: O(logn) works !!!



while(s <= e) {

mid = (s+e)/2 ;

if(a[mid] - - -)

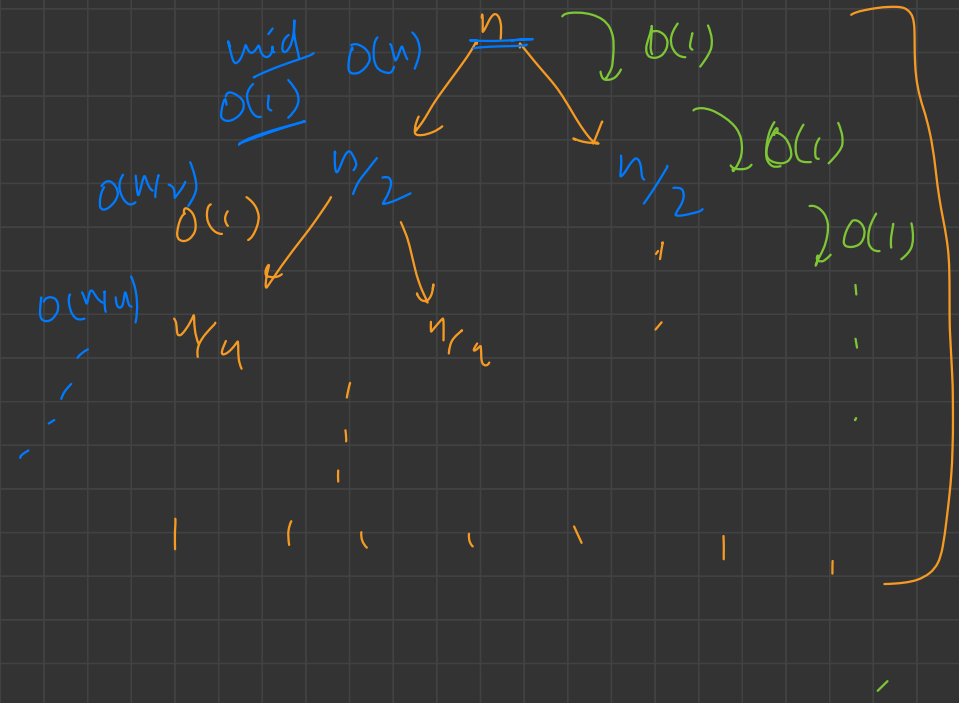
O(1)

in case of LL

O(mid)

O(n/2) O(n/4) - - -

?



in array

O(1)

x

log n

||

O(log n)

in LL

$$O(n/2) + O(n/4) + O(n/8) + \cdots + O(1) >= O(n/2)$$

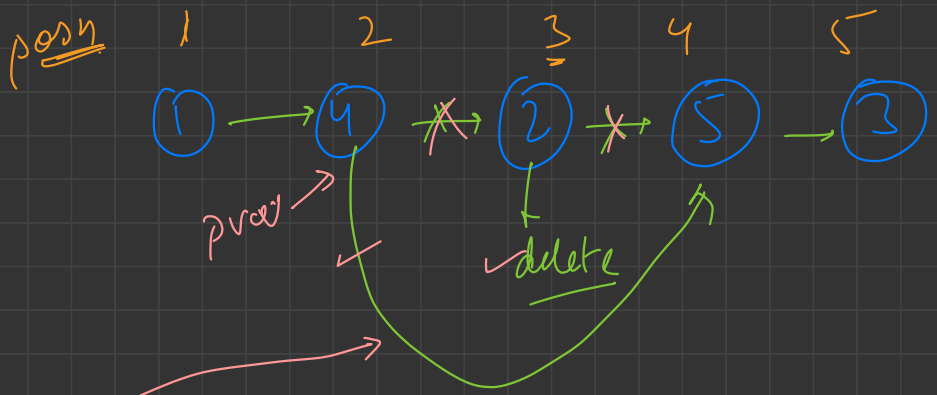
$\Rightarrow$ Binary Search in LL is at least $O(n/2)$

$\approx O(n)$

(1) Delete a node in LL
- head
- tail
- given position

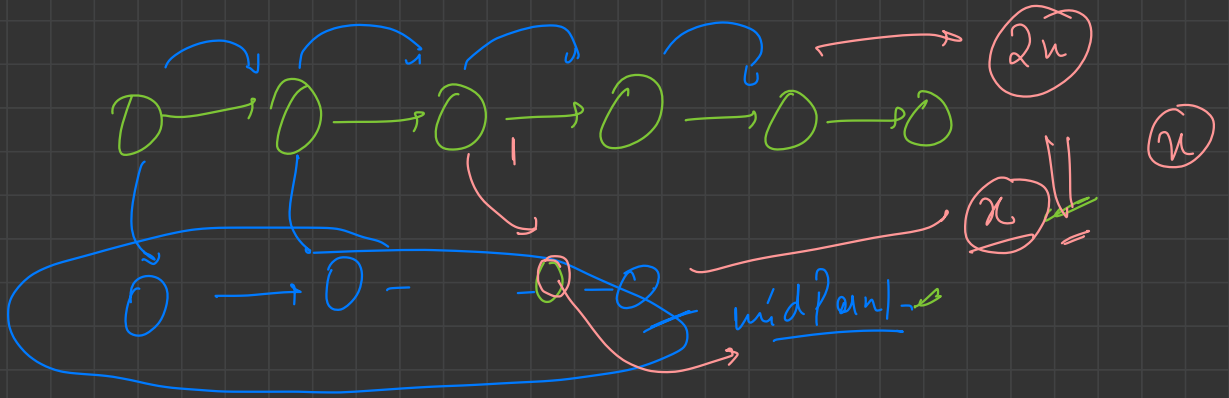
(2) mid point in single iteration

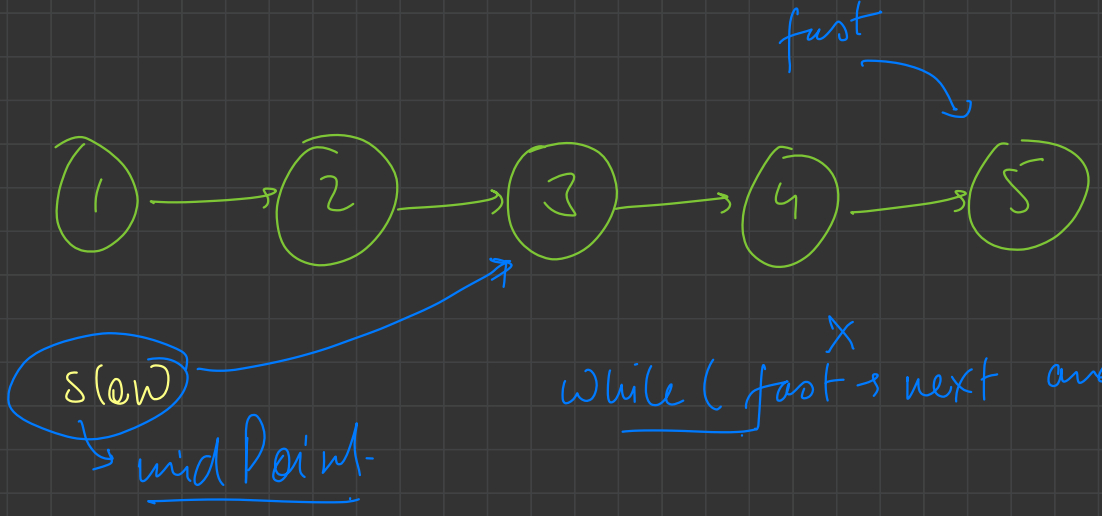
(3) reverse LL $\rightarrow$ iterative / recursive

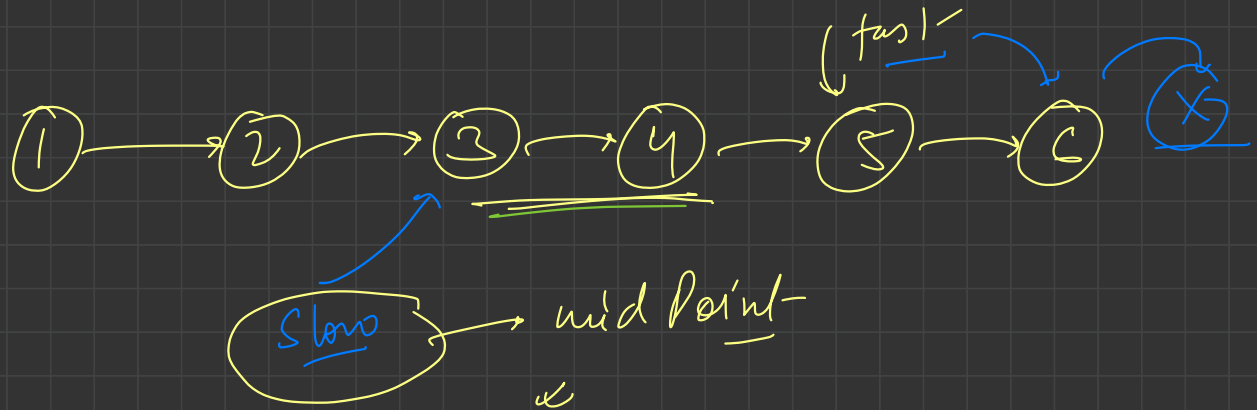


(pos-1) th element pointer ←

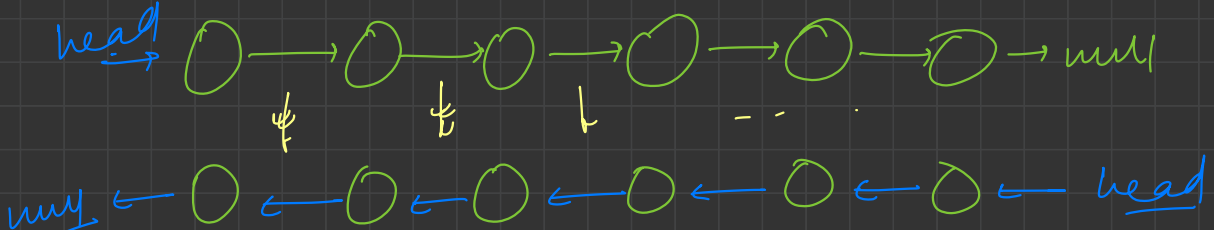
```
Node* temp = prev->next;
prev->next = prev->next->next;
delete temp
```

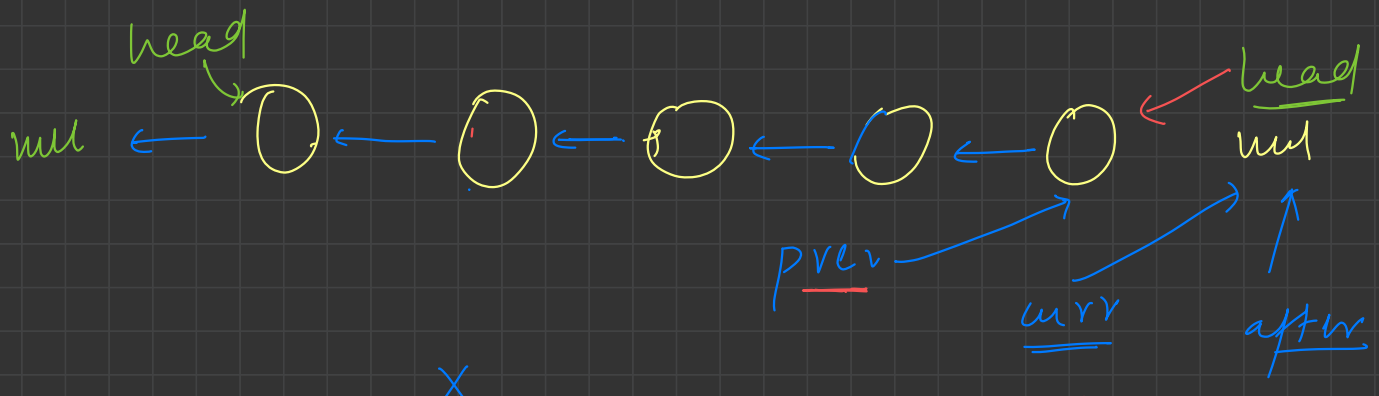

2n
n
n
midPoint


fast
1
2
3
4
5
slow
midPoint
while (fast → next and fast → next → next)



Reverse a linked list






```
while (curr != null) {
    after = curr -> next;
    curr -> next = prev;
    prev = curr;
    curr = after;
}
head = prev;
```

implement recursively

OOPS!