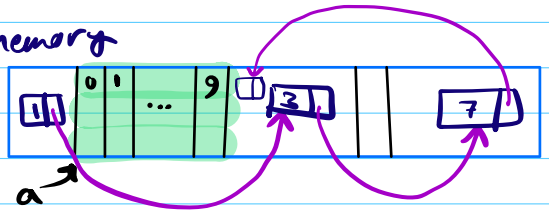
doubly linked list
& circles → next session

1- access kth element
2- check data == X
3- insert at kth element
4- delete data == X
↳ break?
5- reverse LL
6- deep copy

4 Bytes ← int a[10] → 10x4

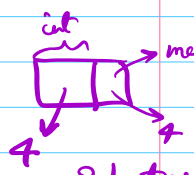
① memory



Arrays
vs.
Linked List

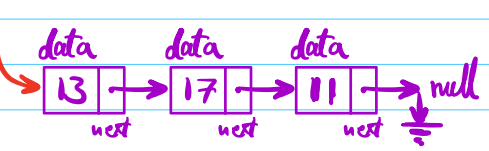
② print(a[7]) O(1) vs O(n)
7th element is in a LL
kth

③ int a[n] → n if var
Compile err



LL → size dynamic during runtime

(data + Pointer) size of each node



Linked List Code

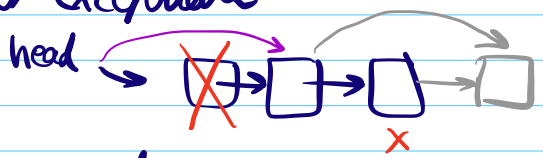
node

```
class node{
    int data
    node next
    node(val){
        data=val;
        next=null;
    }
}
```

Interview Code tips

examples coming

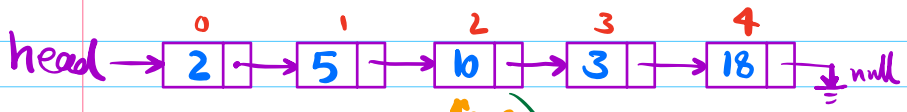
⚠
- NPE & null Pointer exception
- missing head



- missing a reference to node
- head or tail may need different logic

ret node

P1 Access kth element of LinkedList, return a ref to kth element



2 min

k=2 → ans → node f(Node h, K)

TC: O(k)
SC: O(1)

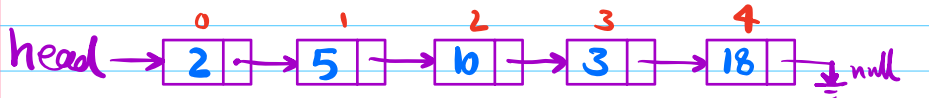
```
cur = head
for(int i=0; i<K; i++){
    if(cur==null) ret null
    cur = cur.next
}
ret cur
```

i = 0, 1, 2 k=2

in case k is wrong k>h this handles it

ret true/false

P2 Given a linked list, check if there is a node with data=X



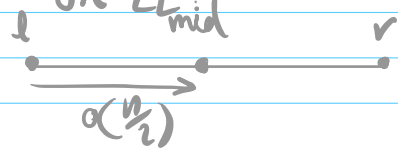
2 min

data=10 → true
data=20 → false

20

```
cur = head
while(cur != null)
    if(cur.data==X) ret true;
    cur = cur.next
}
ret false
```

if LL sorted?
⊗ not efficient to use bin Search on LL

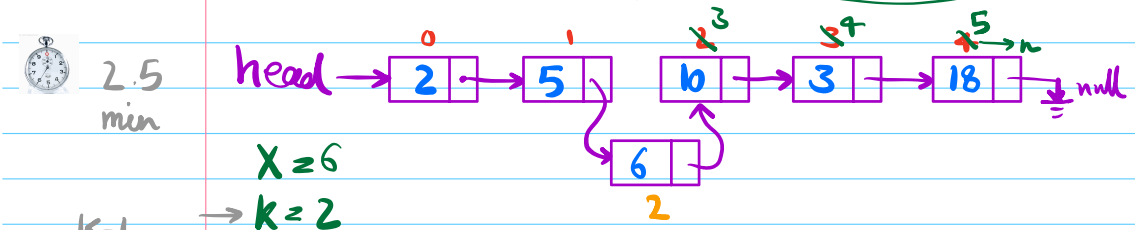


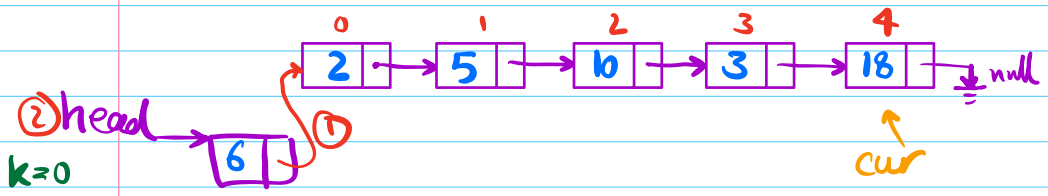
P3 Given a linked list, insert a node with data X at kth position in a linked list

$0 <= k <= n+1$

k → index

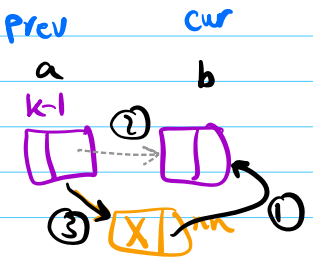
2.5 min






```
nn = new node(X)
if(k==0){ // insert before head
    nn.next = head
    head = nn
    ret head
}
cur = head                    k
for(int i=0; i<k-1; i++){
    if(cur==null) ret null;
    cur = cur.next;
}
nn.next = cur.next
cur.next = nn

ret head
```

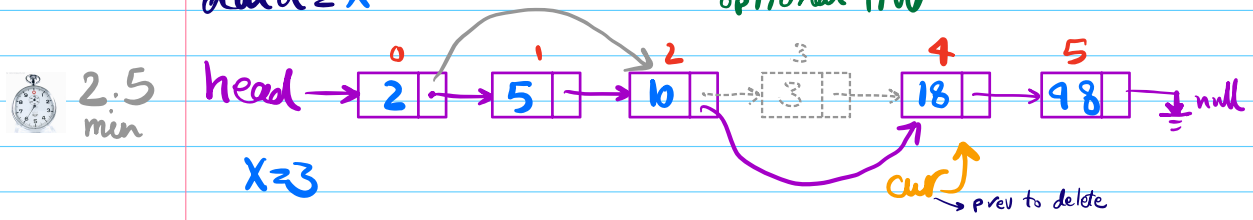


order of these two line matters, otherwise right side of the LL will be missed

new list head ← delNode(head, X)

P4 Given a linked list, delete the first occurance of node with data = X

all occurances optional HW




```
if(head==null) ret null
if(head.data==X){
    ① t=head C++
    ② head=head.next
    ③ delete_mem(t)
    ④ ret head
}
cur=head
while(cur.next!=null)
    if(cur.next.data==X){ // delete
        cur.next=cur.next.next
        ret head
    }
    cur=cur.next;
}
ret head
```

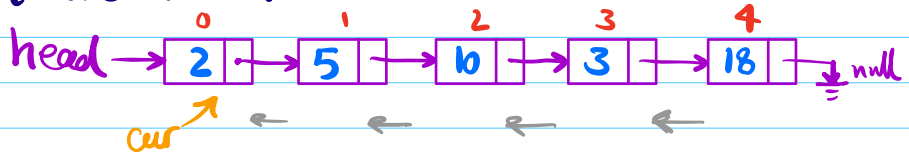
Case
1: 5
2: 99
3: 98
4: 10
5: 2

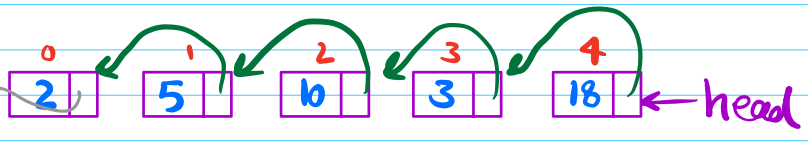
P5 reverse a given a linked list
ret the new head

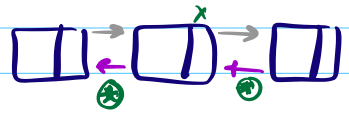
Common interview question

Constraints: - only update pointer not data
- SC: O(1)

2.5 min



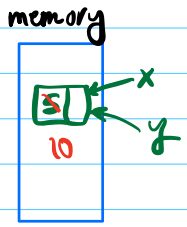




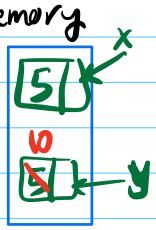

```
next = null
prev = null
cur = head
while(cur != null){
    next = cur.next
    cur.next = prev
    prev = cur
    cur = next
}
head = prev
ret head
```

shallow copy vs. deep copy

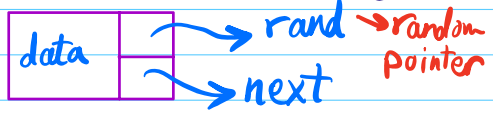
```
X = new Node(5)
Y = X  //copy
Y.data = 10
print(X.data) 10
print(Y.data) 10
```

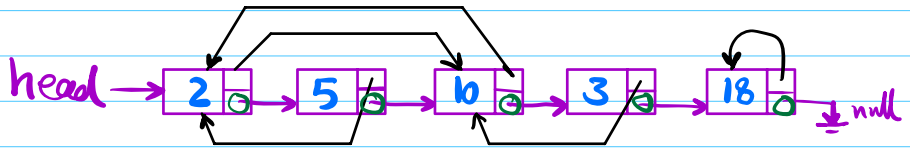



```
X = new Node(5)
Y = new Node(X.data)
Y.data = 10
print(X.data) 5
print(Y.data) 10
```



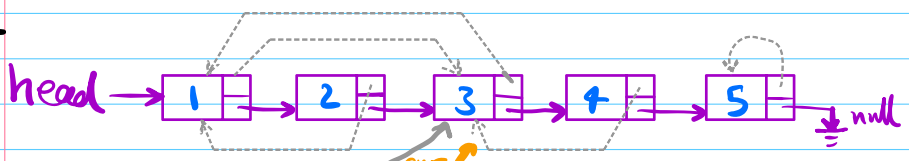
P6 Given linkedList where every node has a random Pointer along with next pointers, create a deep copy of the list.





Solution 1

SC: O(n)




```
while(cur != null){
  curC = new Node(cur.data);
  prevC.next = curC
  HM.add(cur, curC)
  cur = cur.next
}
```

prevC curC

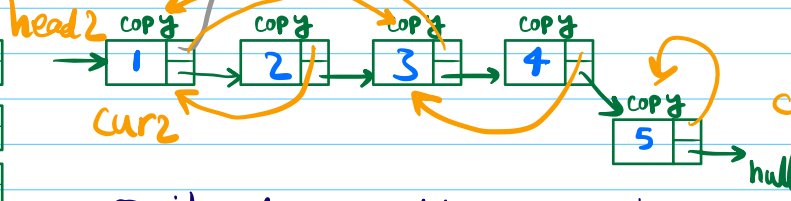
key value

HM<node, node>

curC ← HM[cur]

HM ↓

1	1
2	2
3	3
4	4
5	5



① iterate over LL → copy LL node by node, create the HM — TC: O(n)

② iterate once more — TC: O(n)

TC: O(n)

SC: O(n)

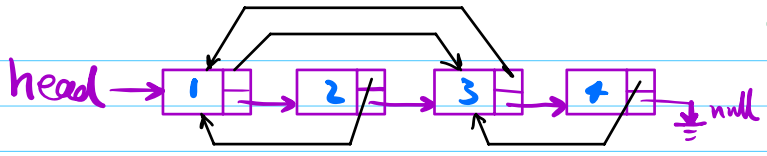
Solution2

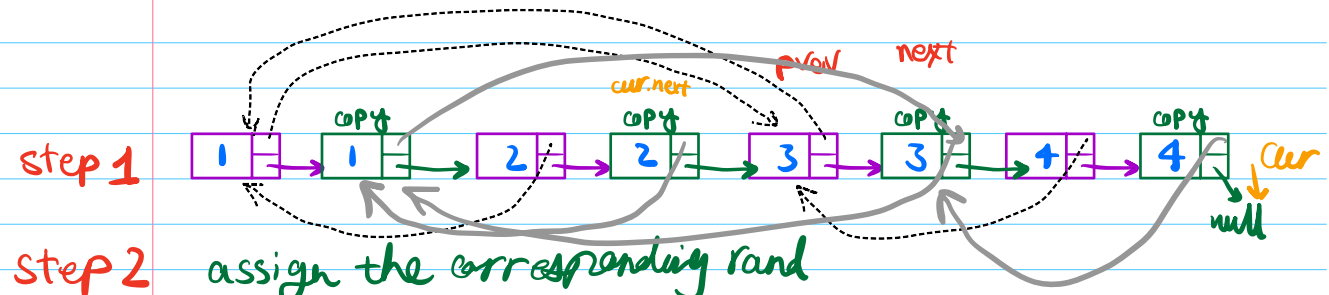
SC: O(1) ??

orig input

O(1) ←

copy output





step 2 assign the corresponding rand

step 3 clean up the mess i.e. split ▭ from ▭

y is the deep copy of X

```
while (X != null)
    y = new node(x.data)
    y.next = X.next
    x.next = y
    X = y.next
```

step 1

TC: O(n)

SC: O(1)

```
cur = head
while (cur != null && cur.next != null){
    cur.next.rand = cur.rand.next
    cur = cur.next.next   go to next ▭
}
```

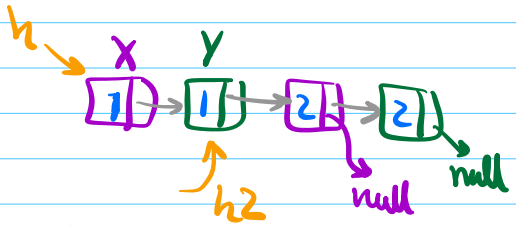
step 2

TC: O(n)

SC: O(1)

step 3:

y is the deep copy of x




```
x = head
y = head.next
head2 = y
while( x != null && y != null ){
    x.next = x.next.next
    if(y.next != null) y.next = y.next.next
    x = x.next
    y = y.next
}
ret head2
```

split from

not the last y

move forward one pair

TC: O(n)
SC: O(1)