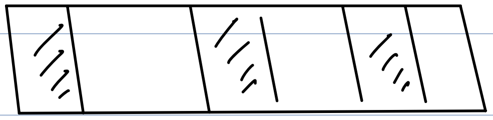
Issues with Array.

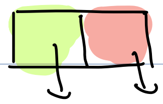


int → 4 Bytes
↓
array → 5
↓
20 Bytes

necessarily
LL → D.S → Not Stores contiguously.

LL

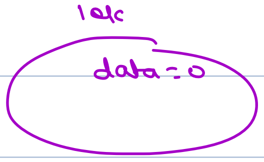
- A linear data structure that can utilize all the free memory
- We need not have continuous space to store nodes of a Linked List.



```
class Node {
    int data;
}
```

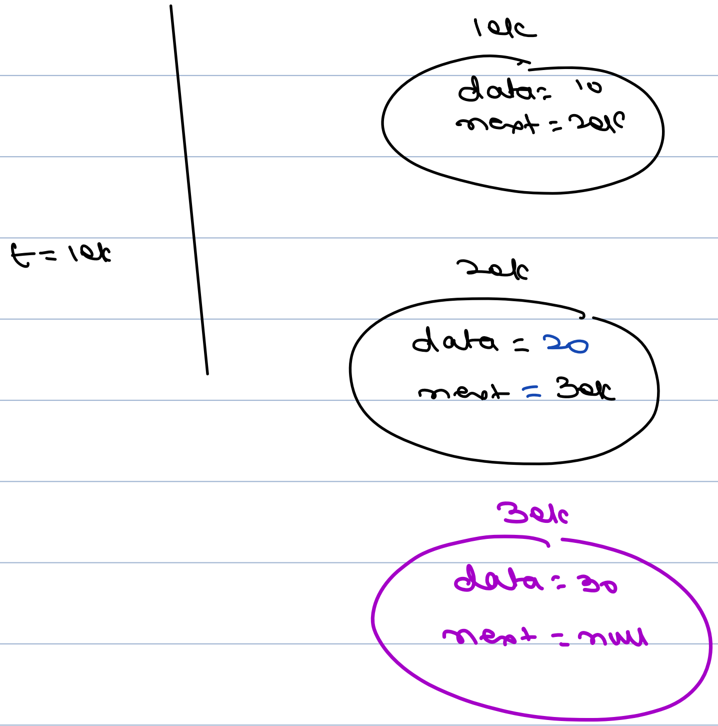
```
Node n = new Node ( );
Node temp = n;
```

n = 10k.
temp = 10k.

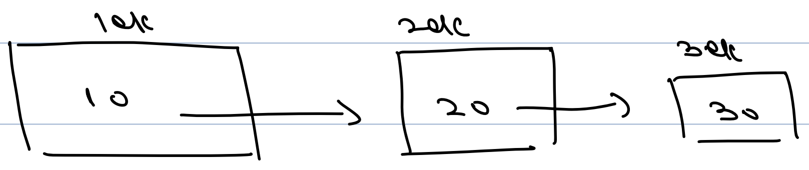


```
class Node {
        int data;
        Node next;
         Node( x) {
            data = x;
            next = null;
         }
}
```

```
Node t = new Node(10); ✓
t.next = new Node(20);
t.next.next = new Node(30);
```

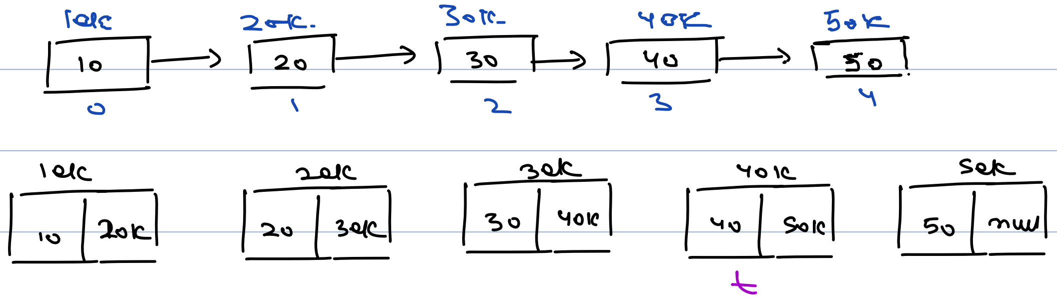


t = 10k ,



head of LL => (first node of LL) ,

Operations on LL :-



1) Access kth idx .

```
int   kth  ( Node  head , int  k ) {
        Node  temp =  head ;       -> 10k .
        for ( i=0 ; i<k ; i++ ) {
                temp = temp.next
        }
        return  temp.data ;
}
```

k = 3

temp = ~~10k~~ ~~20k~~ ~~30k~~ 40k .

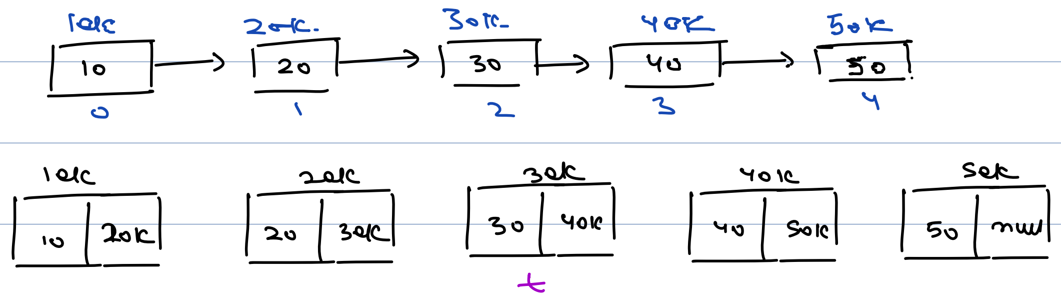
i = 0 ,

i = 1 ,

i = 2 ,

T.C -> O(k) (worst case O(n)).

2) Check for value x.



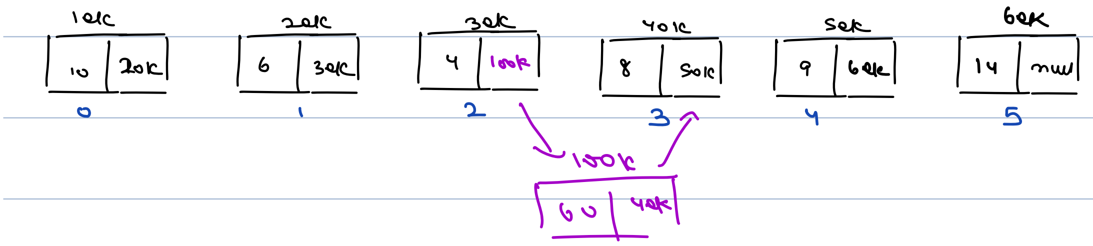
k = 30.

temp = ~~10k~~ ~~20k~~ 30k.

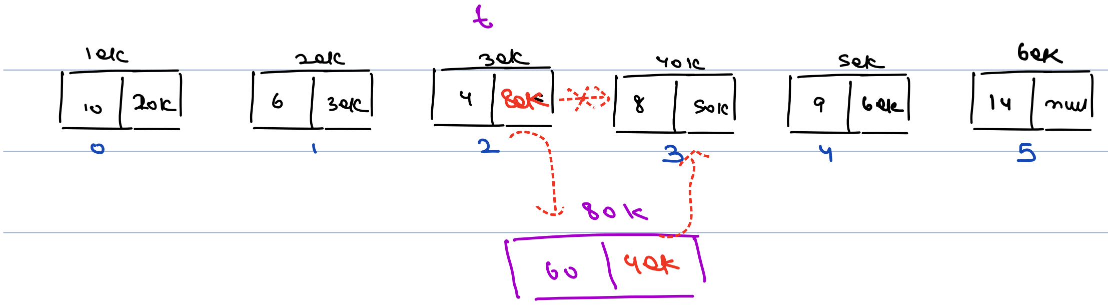
```
                                  -> 10k
bool check ( Node head, int x ) {
     Node temp= head;
     while ( temp != null ) {
         if (temp. data = = x) {
             return True;
         }
         temp= temp.next;
     }
     return false;
}
```

T.C -> O(n).

Ques Insert a new Node with Data.

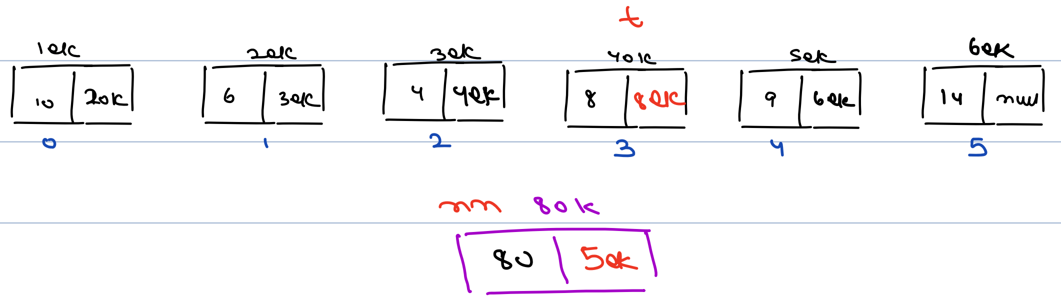


Ex1 V = 60 , P = 3



```
          → 80k
Node  nn = new Node(60);
   Node temp = head;
  for (i=0; i< p-1; i++) {
          temp = temp.next;
  }
   nn.next = temp.next;
   t.next = nn;
```

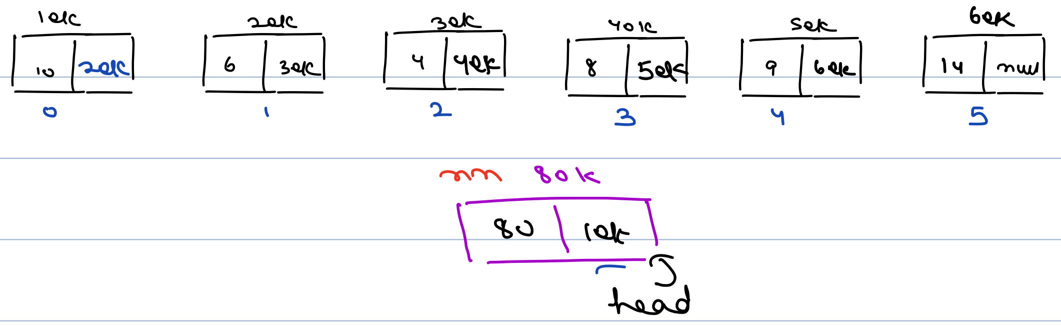
Ex1 V= 80 , P=4



```
                ⤻ 80k
Node  nn=  new  Node(80);
  Node temp= head;
 for (i=0; i< p-1; i++) {
          temp= temp.next;
 }
  nn.next= temp.next;
  t.next = nn;
```

Edge case :-

Ex1 V= 80 , P= 0



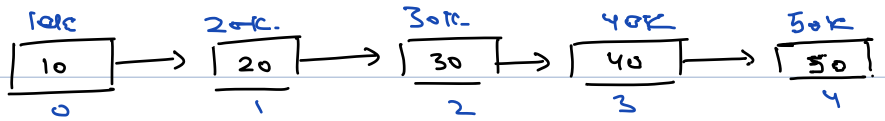
```
            → 80k
Node   nn=   new   Node(80);
 nn.next = head   ✓
  head = nn
```

```
                                        → 10k,
Node  insertAtK ( Node head, int v, int p){
      Node  nn=  new  Node(v);
      if ( p==0 ) {
            nn.next = head
            head = nn
            return head;
      }
      Node  temp= head;
      for (i=1; i<=p-1; i++) {
            temp= temp.next;
      }
      nn.next= temp.next;
      t.next = nn;
      return head;
}
```

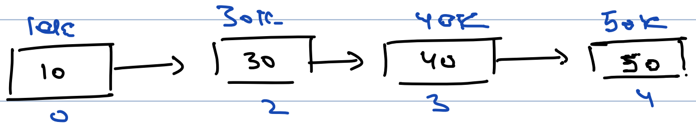
T.C: O(P) (worst O(n))

// Dry run insert at last.

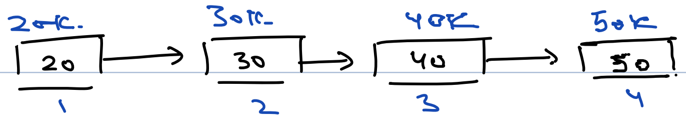
Ques Deletion in a LL.



e.g 1) x = 20

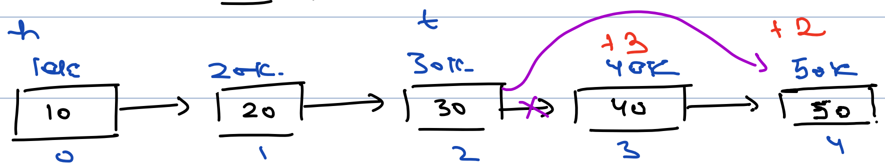


e.g 2) x = 10



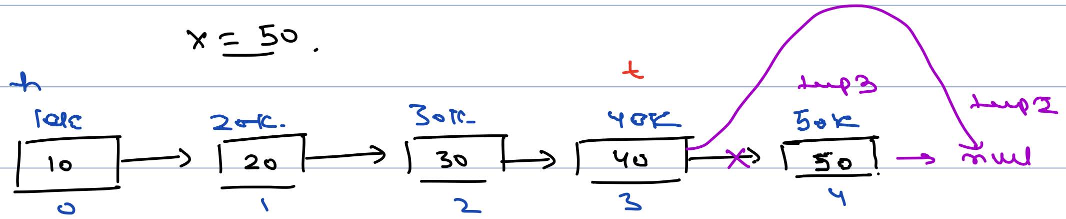
Soln

x = 40.



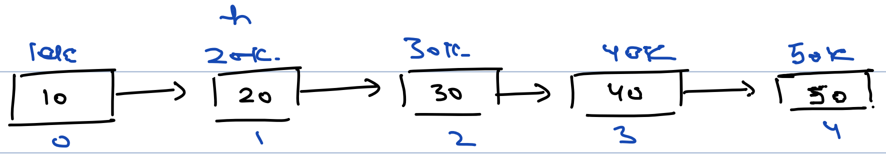
```
Node temp = head
while (temp.next != null) {
        if (temp.next.data == x) {
            temp3 = temp.next;
          temp2 = temp.next.next;
           temp.next = temp2;
            free(temp3)   // c++;
            return head;
        }
      temp = temp.next;
}
```

x = 50.



```
Node temp= head
while (temp.next != null) {
    if (temp.next.data = =x) { ✓
        temp3 = temp.next; ✓
        temp2 = temp.next.next; ✓
        temp.next = temp2; ✓
        free(temp3) //c++;
        return head;
    }
    temp= temp.next;
}
```

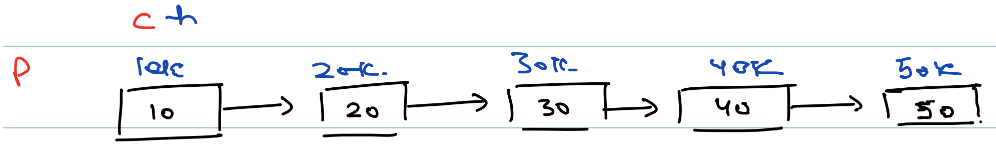
x = 10.



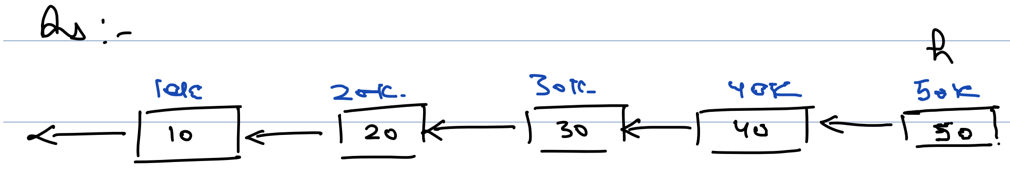
```
Node delete ( Node head, int x ) {
    if (head.data==x ) {
        head= head.next;
        return head;
    }
    Node temp= head
    while (temp.next != null) {
        if (temp.next.data==x){ ✓
            temp3= temp.next; ✓
            temp2= temp.next.next; ✓
            temp.next= temp2; ✓
            free(temp3)  // C++;
            return head;
        }
        temp= temp.next;
    }
}
```

T.C → O(n)

Reverse a LL

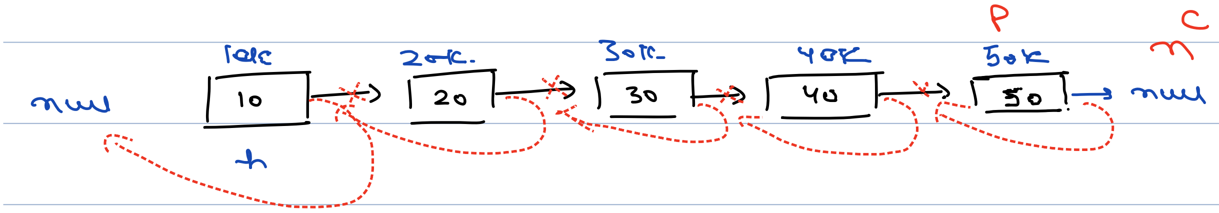


Ans :-



Soln :-

curr
prev



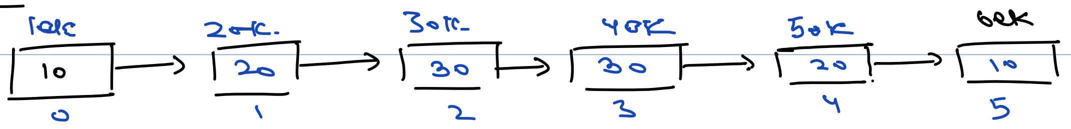
```
Node reverse ( Node head) {
        Node c = head;
        Node p = null;

        while( c != null) {
            Node next = c.next
            c. next = p;
            p = c;
            c = next;
        }
        head = p;   //skip & return p directly.
        return h;
}
```

$T.C \rightarrow O(n)$

$S.C \rightarrow O(1)$

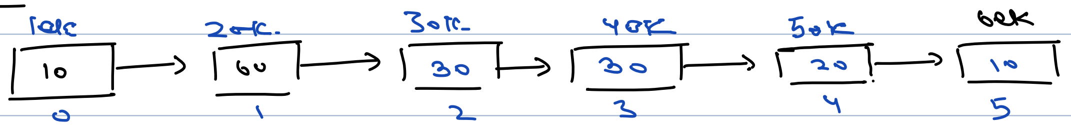
Ques Given a LL, check if it is palindrome.

e.g1



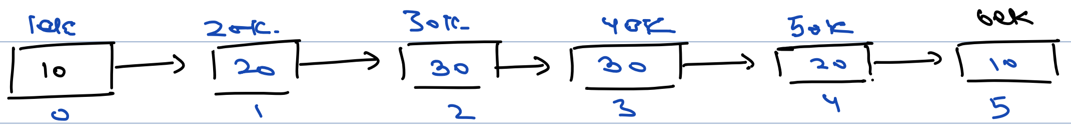
Ans True

e.g 2



Ans false.

Soln :-



Soln 1 :-

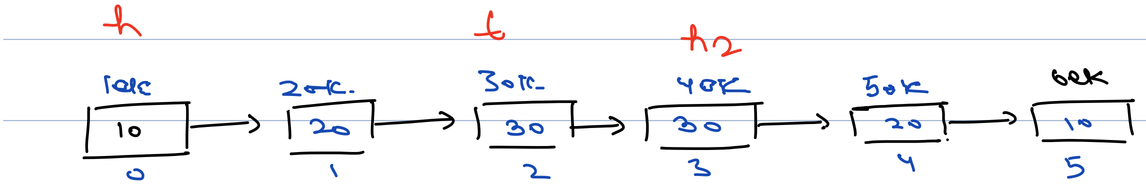
1) Create a copy of LL.

2) Reverse it

3) Compare

$T.C \rightarrow O(n)$

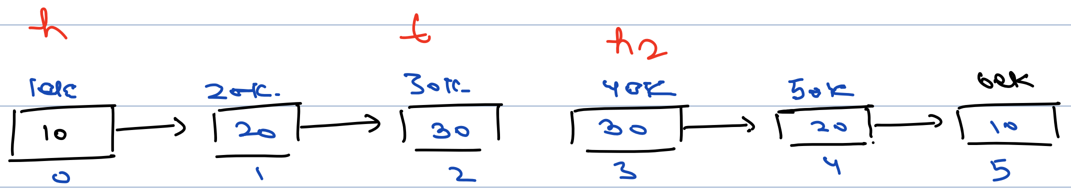
$S.C \rightarrow O(n)$



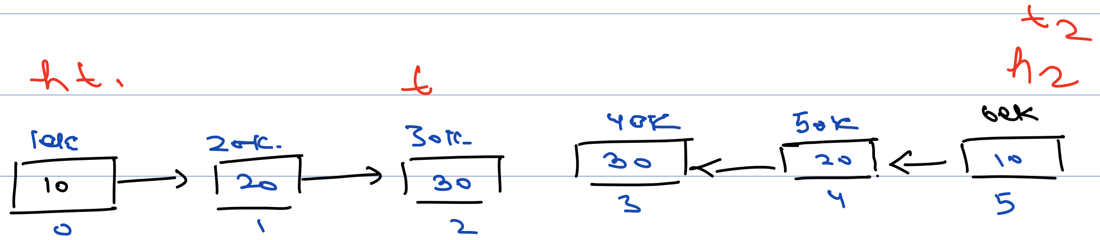
Step-1 :-

find len of LL.

```
n=0;
temp= head;
while (temp!=null) {
    n++;
    temp= temp.next;
}

// n= 6

int halflen = n/2;

Node temp = head;
for (i=0; i<halflen-1; i++) {
    temp = temp.next;
}

Node head2= temp.next;
temp.next= null;
```



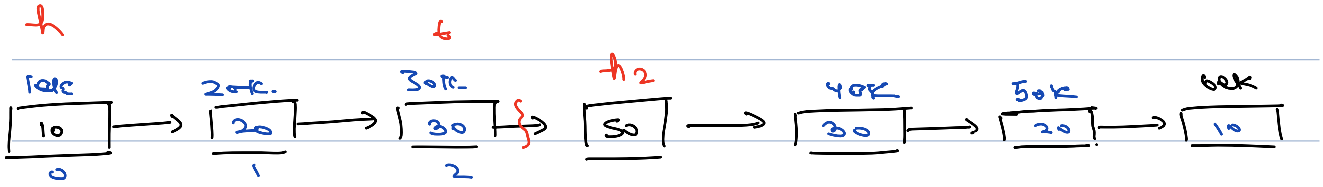
head2 = reverse (head2);



```
Node t1 = h; Node t2 = h2
while (t1 != null && t2 != null) {
        if (t1.data != t2.data) {
            return false;
        }
        t1 = t1.next;
        t2 = t2.next;
}
return True;
```

T.C → O(n)

S.C → O(1).



① n = 7

② half len = 3

Note :- while trying Palindrome Problem, change LL back to how it was originally.