Linked List 2: Sorting and Detecting Loop

Quiz 1 :- what is the time Complexity needed to delete a node from a linked list?

↳ O(N)

Quiz 2 :- what is the time Complexity needed to insert a node as the head of a linked list?

↳ O(1)

Quiz 3 :- what is the time Complexity needed to delete the last node from a linked list?

↳ O(N)

Quiz 4 :- Can we do binary Search in a sorted Linked List?

↳

No

Question :- Given a Linked List (LL), Find the middle element.

Ex :- Head → $a_1 \rightarrow a_2 \rightarrow a_3 \rightarrow a_4 \rightarrow a_5$ (mid = a_3)

Ex :- Head → $a_1 \rightarrow a_2 \rightarrow a_3 \rightarrow a_4 \rightarrow a_5 \rightarrow a_6$ (mid = a_3)

Brute force

→ Find Size of LL

↳ Travel to half of length & return that

TC	$O(N)$
SC	$O(1)$

Constraint :- Do it in 1 iteration

Ex Head → $a_1 \rightarrow a_2 \rightarrow a_3 \rightarrow a_4 \rightarrow a_5 \rightarrow a_6 \rightarrow a_7$

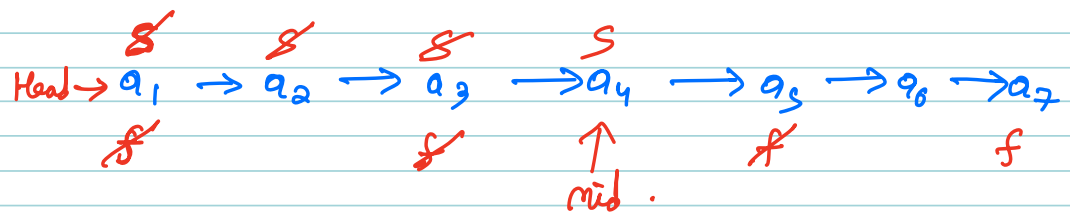
IDEA :-

X =	100 km/hr	After 1 hr →	Distance Travelled 100
Y =	200 km/hr	After 1 hr →	200

↳ From this can we conclude that in 1 hr 'y' will cover 200 km & 'x' will cover 100 km i.e. half of y

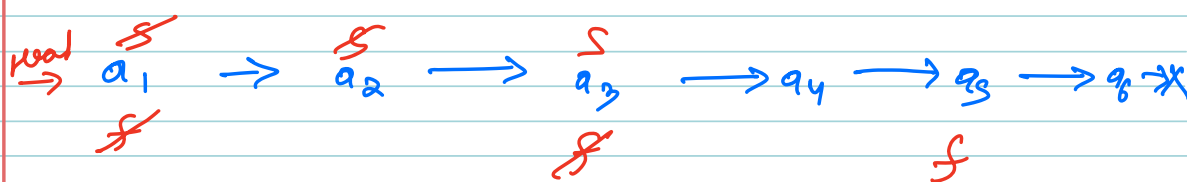
So, Something similar can be used for LL. Use slow & fast pointer.

Eg 1.:- **ODD LENGTH**



Condition to Stop :- f.next == null

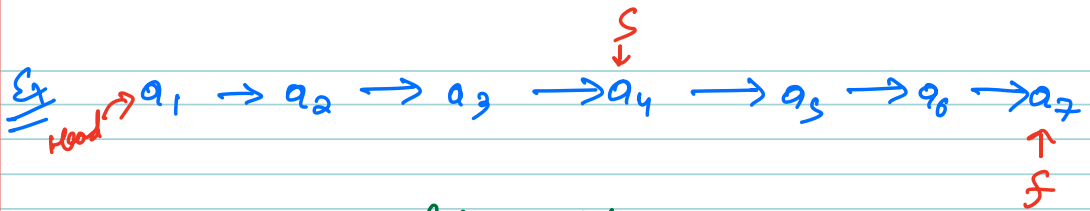
Eg 2 :- **EVEN LENGTH**



condition to stop :- f.next.next == null.

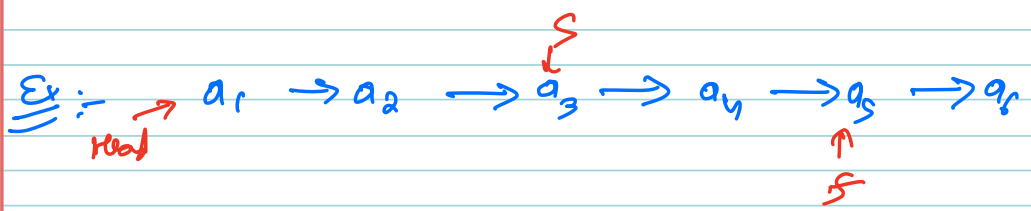
PSEUDO CODE

```
Node mid (Node h) {
    if (h == null) { return null; }
    Node s = h;
    Node f = h;
    while (f.next.next != null && f.next != null) {
        s = s.next;
        f = f.next.next;
    }
    return s;
}
```



Correct while condition

while (f.next != null && f.next.next != null)



TC	$\to O(N)$
SC	$\to O(1)$

Question :- Given two sorted LL, merge them into a single sorted LL.

Eg :- LL1 ⇒ 2 → 4 → 6 → 8 → 10 ← x

LL2 ⇒ 1 → 3 → 5 → 7 → 9 ← y

output :- 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9 → 10

Interesting point :- In merge two sorted array we need to create a resultant array, while in the case of LL we dont need to create.

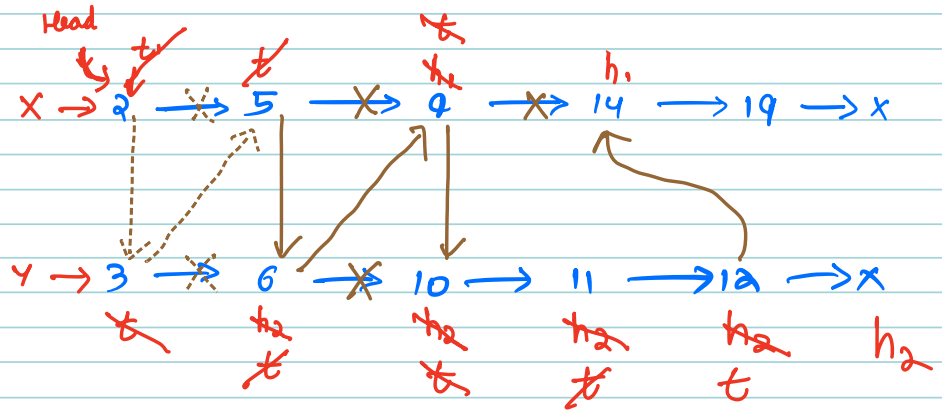
Ques 5 :- Given two sorted Linked Lists, Merge them into a single sorted linked list.

$x \rightarrow [2 \rightarrow 10 \rightarrow 11]$

$y \rightarrow [1 \rightarrow 5 \rightarrow 12 \rightarrow 15]$

$\hookrightarrow 1 \rightarrow 2 \rightarrow 5 \rightarrow 10 \rightarrow 11 \rightarrow 12 \rightarrow 15$

Ex



Consume h_1

$t.next = h_1$

$t = h_1$

$h_1 = h_1.next$

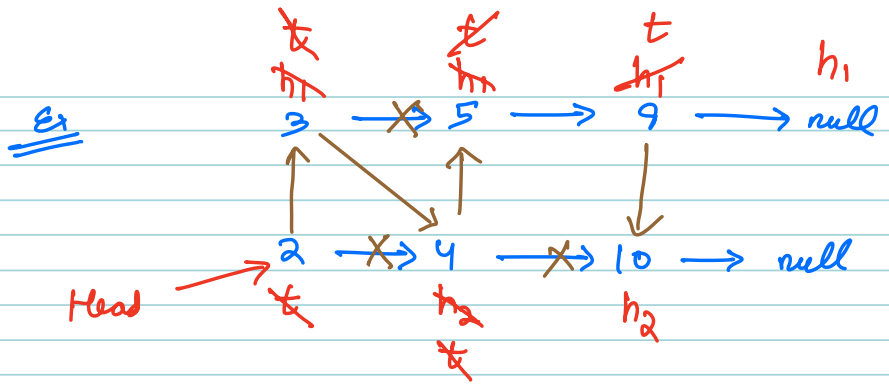
Consume h_2

$t.next = h_2$

$t = h_2$

$h_2 = h_2.next$

* Interesting point :- At the end we just need to make one connection. No loop is required.



Consume h_1

$t.next = h_1$

$t = h_1$

$h_1 = h_1.next$

Consume h_2

$t.next = h_2$

$t = h_2$

$h_2 = h_2.next$

PSEUDO CODE

```
void merge ( Node h1 , Node h2){
  if ( h1 == null) { return h2;}
  if ( h2 == null) { return h1;}

  Node head , tail;
  if ( h1.data < h2.data ){
        head = h1
        tail = h1
        h1 = h1.next;
  } else {
        head = h2
        tail = h2
        h2 = h2.next;
  }
```

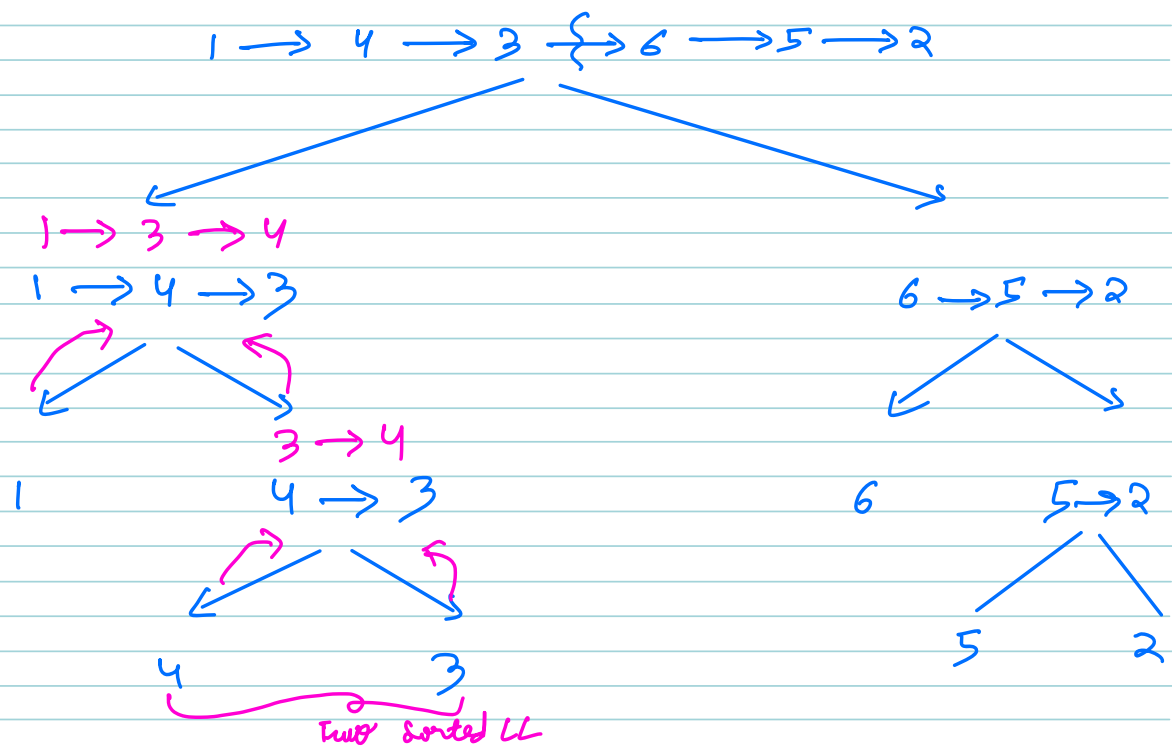
a b
5 5

```
while ( h1 != null && h2 != null){
    if ( h1.data < h2.data){
        t.next = h1
        t = h1
        h1 = h1.next
    } else{
        t.next = h2
        t = h2
        h2 = h2.next
    }
}
if ( h1 != null) {
    t.next = h1
}
if ( h2 != null) {
    t.next = h2
}
return head;
}
```

TC → O(N+M)

SC → O(1)

Question :- A Linked List is given, sort the Linked list using merge sort.



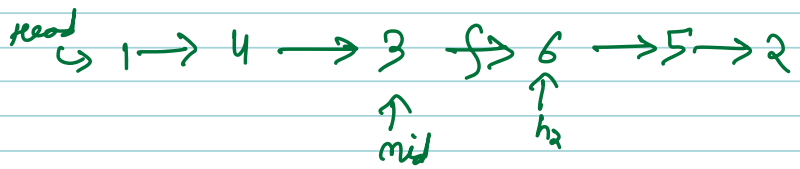
Steps :-

1. Divide into two halves
2. Keep Dividing till you get single element as single element is always sorted.
3. Start Merging two Sorted Lists.

output :- 1 → 2 → 3 → 4 → 5 → 6

PSEUDO CODE

Step 1 :- Find Middle & break




```
mid = findMid(head)
h2 = mid.next;
mid.next = null
```

Step 2 :- Call part Recursion on both Left & Right

```
head = mergeSort(head);
h2 = mergeSort(h2);
```

Step 3 :- merge head & h2

```
Ans = merge(head, h2);
```

Complete Code

```
Node mergeSort ( Node h ) {
    if ( h == null || h.next == null ) {
        return h;
    }
    Node mid = middle ( h );          -> n
    Node h2 = mid.next;
    mid.next = null;
    h = mergeSort ( h );              -> T(n/2)
    h2 = mergeSort ( h2 );            -> T(n/2)
    Node ans = merge ( h, h2 );       -> n
    return ans;
}
```

Step 1 — Step 2 — Step 3

TC → O(N logN)

Recurrence Relation

↳ $T(N) = 2T\left(\frac{N}{2}\right) + N$

SC → O(logN)

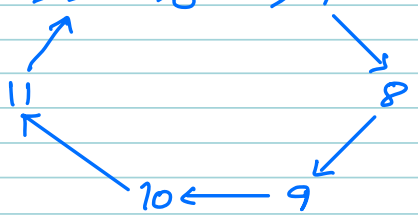
↑ Recursion Stack

Resume :- 10:43

Floyd Cycle Detection Algo

Question :- Given a Linked List. Find whether it contains a cycle.

Ex :- 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9 → 10 → 11 → 5



Ans = yes

Ex2 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9

Ans = No.

IDEA :-
1. Put a temp at the head.
2. Keep doing ⇒ temp = temp.next.
3. if temp reaches null then no cycle otherwise there is a cycle.

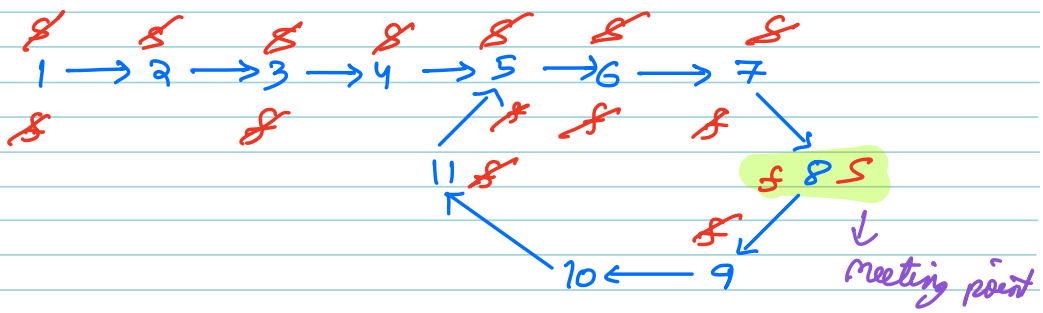
Problem :- If there is a cycle, then it will lead to infinite loop.

IDEA2 HashSet<Node> will store address of Node
- ① Iterate on LL, keep on storing nodes
- ② If you reach Null, then no cycle
- ③ If address start repeating, there is a cycle.

TC	O(N)
SC	O(N)

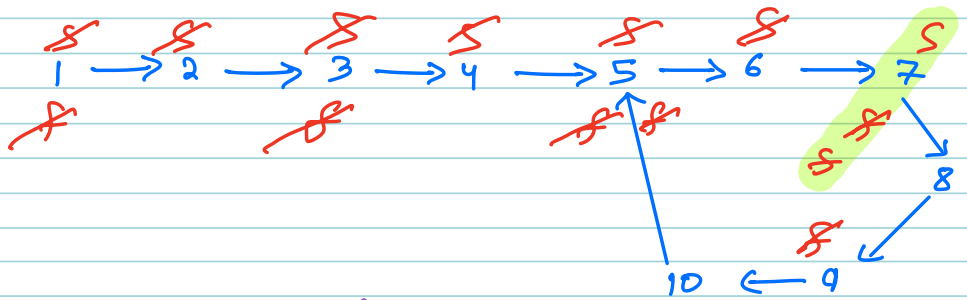
IDEA 3

Ex



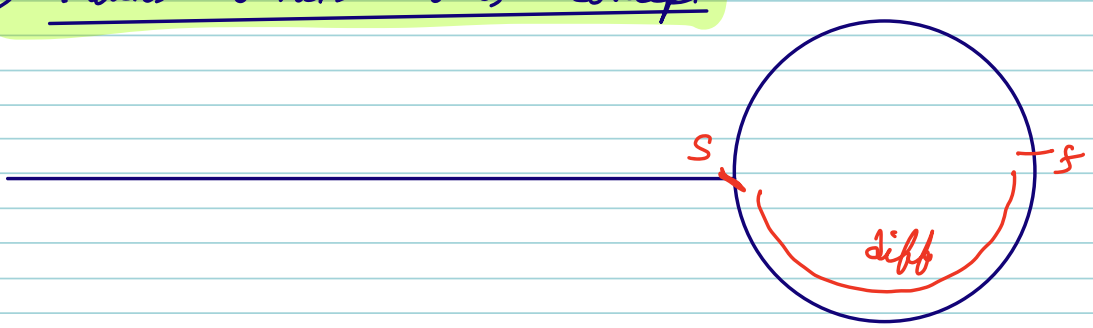
↳ Slow will meet fast if this a cycle otherwise fast will reach end.

Ex



=> Working.

=> **Maths behind this Concept**



→ fast will enter the loop first. Agree?
↳ yes.

→ Then At some time s will be at the starting point of loop & f will be in the loop. There will be some Diff b/w them.

→ Now After this, with each movement the Diff b/w them will reduce by 1 because slow will move 1 node while fast will move over two node.

TC → O(N)

↳ Observe slow will only complete one loop in worst case.
In worst case when start stands on loop point, the fast will be at max distance away i.e. (loop length).
They will be able to meet in just iteration over loop.

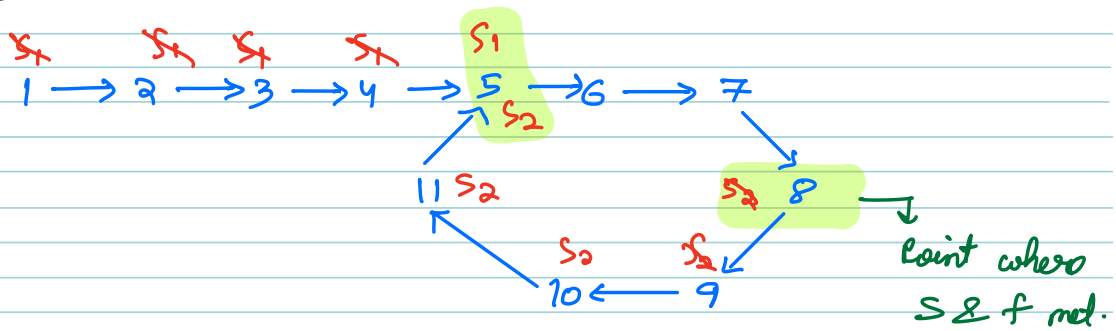
PSEUDO CODE

```
bool detectLoop(Node h) {
    Node s = h;
    Node f = h;
    bool isCycle = False;
    while (f != null && f.next != null){
        s = s.next;
        f = f.next.next;
        if (s == f){
            isCycle = True;
            break;
        }
    }
    return isCycle;
```

3

Question :- Detect the starting point of cycle in a given Linked List.

IDEA

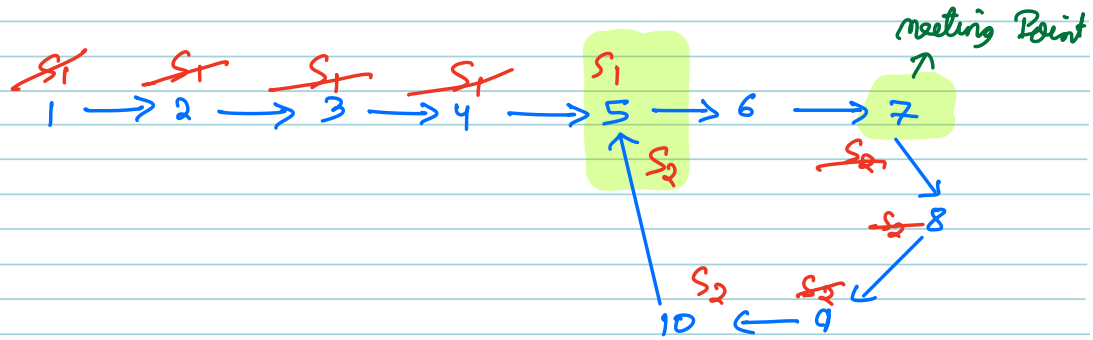


Steps 1.) Place S_1 on head & S_2 on meeting Point.

2.) start iterating with 1 node hop at a time.

3.) will observe they meet at cycle starting point.

Ex



PSEUDO CODE

```
detectCycleStart ( Node h ){
    Node s = h;
    Node f = h;
    bool isCycle = False;
    while ( f != null && f.next != null){
        s = s.next;
        f = f.next.next;
        if ( s == f){
            isCycle = True;
            break;
        }
    }
    if ( isCycle == False ){
        return False;
    }
    S1 = h , S2 = f;
    while ( S1 != S2 ){
        S1 = S1.next;
        S2 = S2.next;
    }
    return S1;
}
```

⇒ Follow up :- Remove the Cycle

Steps :- ① Make temp stand on starting point of cycle.

② Move temp ahead till

temp.next != S;

③ Break temp.next connection after loop

i.e. temp.next = null

PSEUDO CODE

```
remove loop( Node h){
    Node s = h;
    Node f = h;
    bool isCycle = False;
    while( f != null && f.next != null){
        s = s.next;
        f = f.next.next;
        if( s == f){
            isCycle = True;
            break;
        }
    }
    if( isCycle == False){
        return False;
    }
```

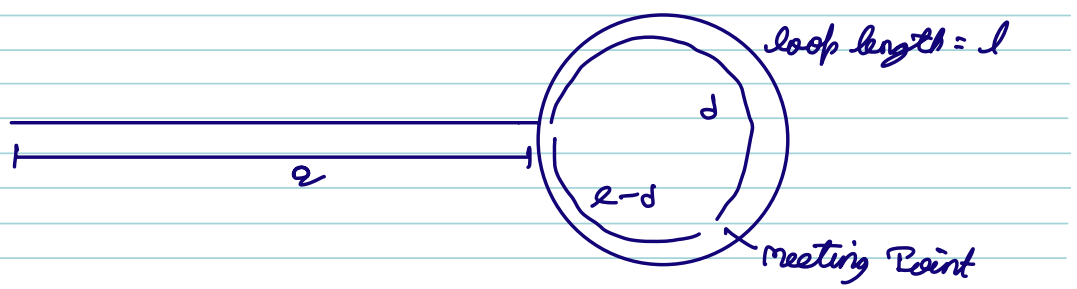
```
}

s1 = h , s2 = f;
while ( s1 != s2 ){
    s1 = s1.next;
    s2 = s2.next;
}
temp = s1;
while ( temp.next != s1 ){
    temp = temp.next;
}
temp.next = null
return h;
}
```

TC → O(N)
SC → O(1)

★★★

Q :- Why they met at loop starting Point ??



↳ Distance travelled by slow before reaching meeting point = a + d

↳ Distance travelled by fast before reaching meeting point = a + (c × l) + d

(c × l) : no. of loop iteration

↳ Iterations of loop by f

Now, Distance travelled by fast = 2 × distance travelled by slow.

$$\Rightarrow a + (c \times l) + d = 2(a + d)$$

$$\Rightarrow a + (c \times l) + d = 2a + 2d$$

$$\Rightarrow (c \times l) = a + d$$

$$\Rightarrow a = cl - d$$

$$\Rightarrow a = cl - d + \underbrace{l - l}_{\text{add extra}}$$

$$\Rightarrow a = l(c-1) + (l-d)$$

$(c-1)$: Constant

$$\Rightarrow a = [l \times \text{Constant}] + [l - d]$$

$[l \times \text{Constant}]$: Some iterations of loop