Linked List

→ Linear Based Data Structure

Skip List

↳ Advance Data Structure

Linked List	Arrays
1) No contiguous memory allocation	1) contiguous memory allocation
2) Insertion & Deletion is easy in Linked list	2) Searching is quite easy & more frequent operation
3) No Random Accell	3) Random accell
4) Memory Requirement is high	4) Lesser memory requirement

Node

- Actual Data
- Pointer
  - ↳ store the address of next node

Array

size of array is fixed

Static Array

Array

Runtime
↳ change the size of an array

Dynamic Array

ArrayList

Operations

Detect the cycle

Reversal

Recursive

Iterative

Insertion

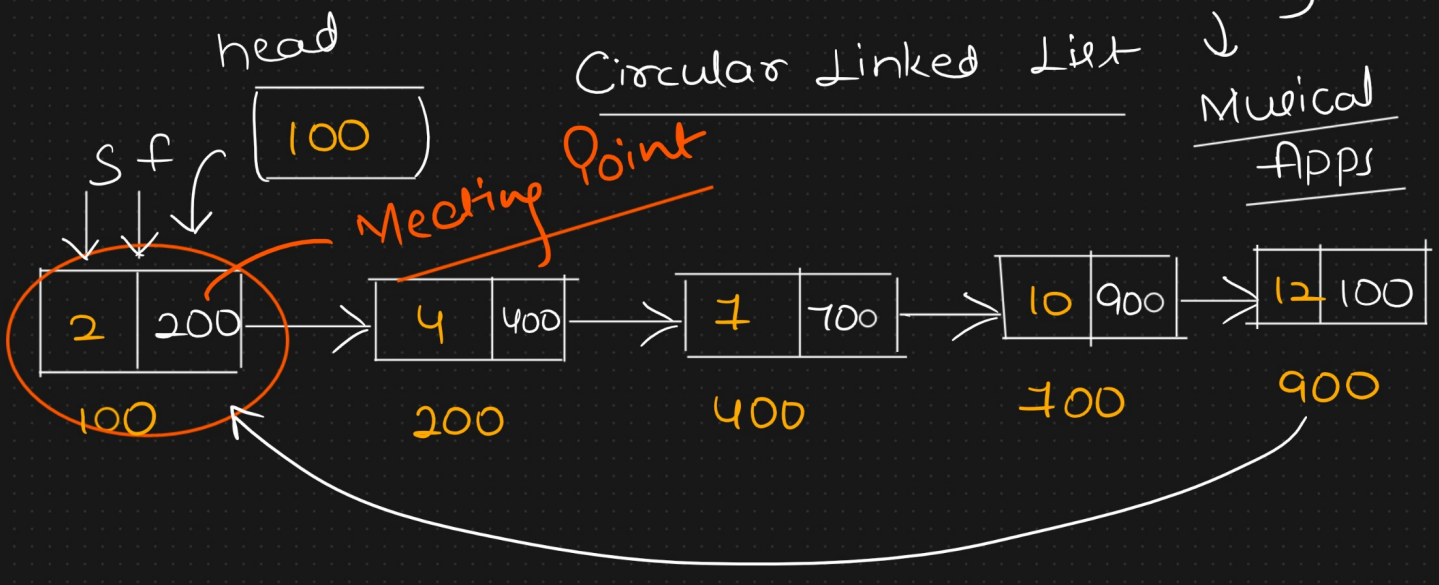
Deletion

Middle Node in a Linked List

↳ Palindrome in Linked List

Circular Linked List

Spotify
↓
Musical Apps



head.next = 200

↳ Pointer

head.next.next = 400

Floyd's cycle Detection

Tortoise — slow, Hare — fast

slow = head

fast = head

slow | 1Node/iteration

fast | 2Node/iteration

fast = fast.next.next

slow = slow.next

Cycle exists

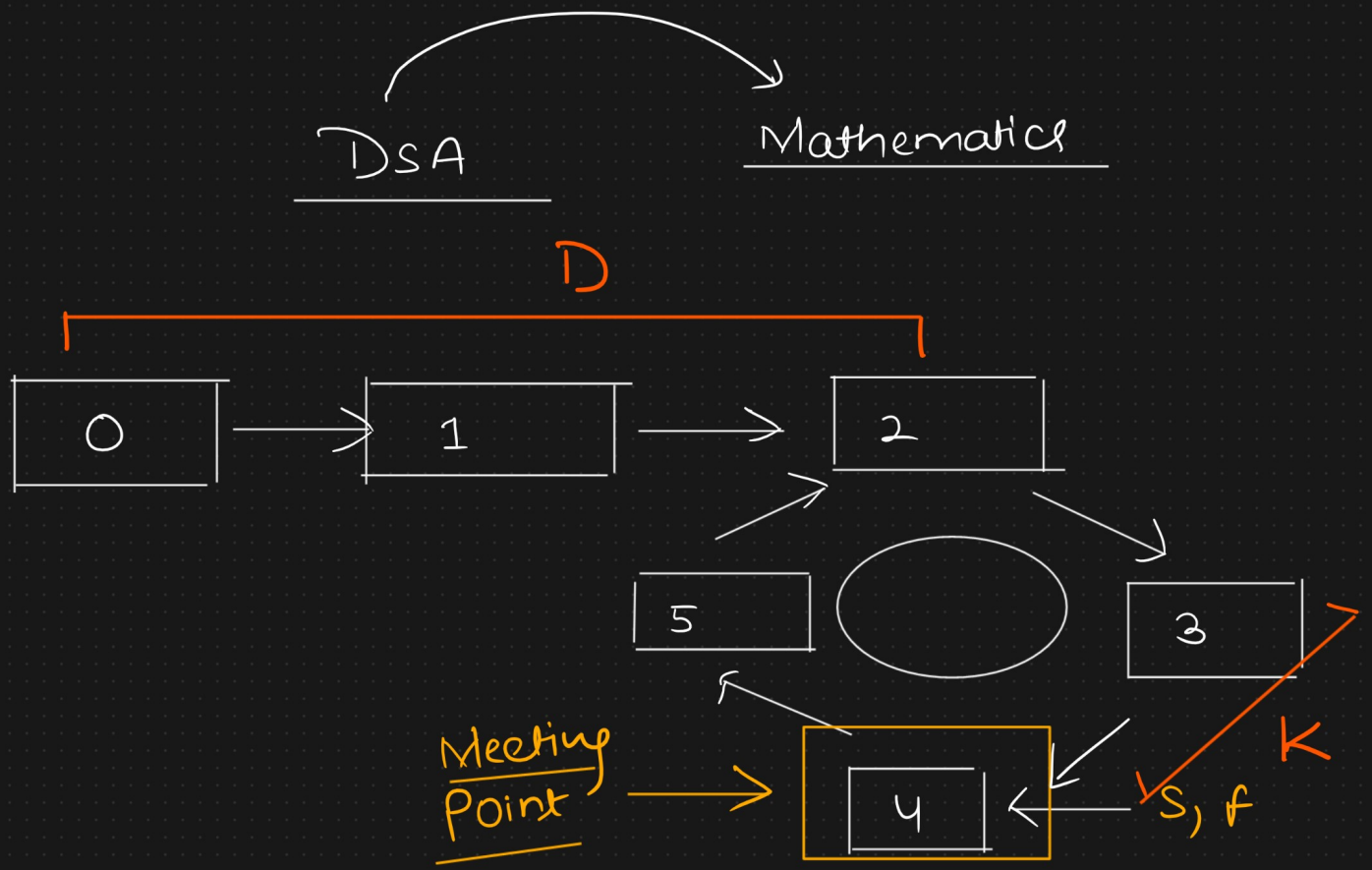
↳ (slow == fast) — Meet

No cycle

↳ (slow != fast)

flag=0

```
while( slow != null && fast != null &&
                        fast.next != null)
{
    slow = slow.next;
    fast = fast.next.next;
    if(slow == fast){
        flag = 1;
        break;
    }
}
```



① —— N = D + K + C(i)

② —— 2N = D + K + C(j)

⟹ N = C(j-i)

Meeting Point

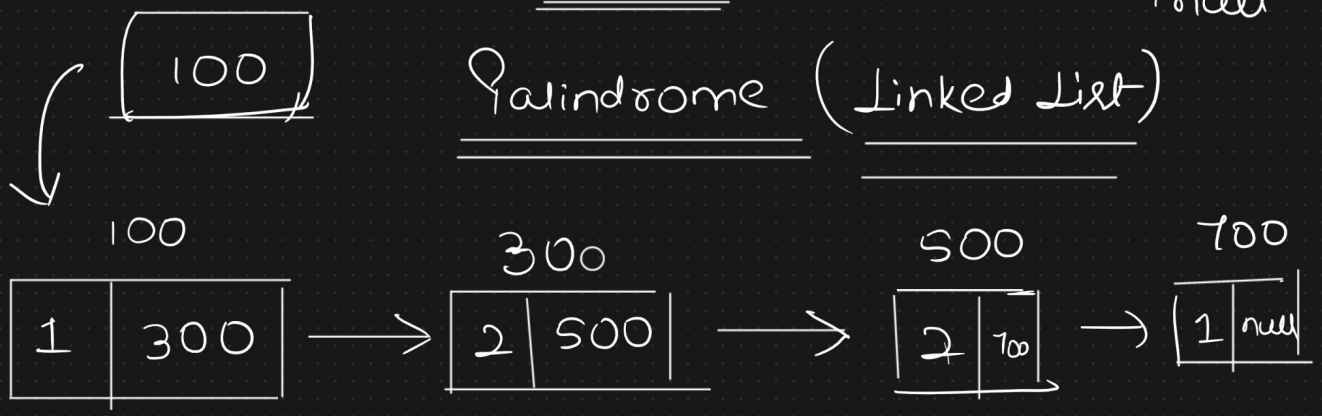
0	1	2	3
1	2	2	1

list 1

Palindrome (Linked List)

head

Current = ~~100~~ ~~300~~ ~~500~~ ~~700~~ null



Two pointer Approach

ArrayList

0	1	2	3
1	2	2	1

low

high

low = 0 ✗ 2

high = 3 ✗ 1

Time complexity

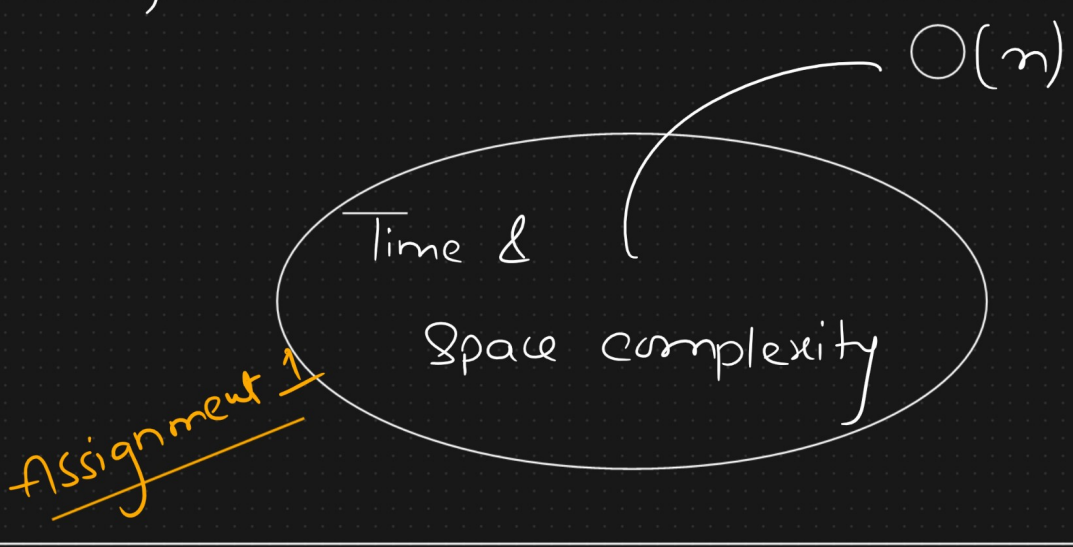
↳ O(n)

Space complexity

↳ O(n)

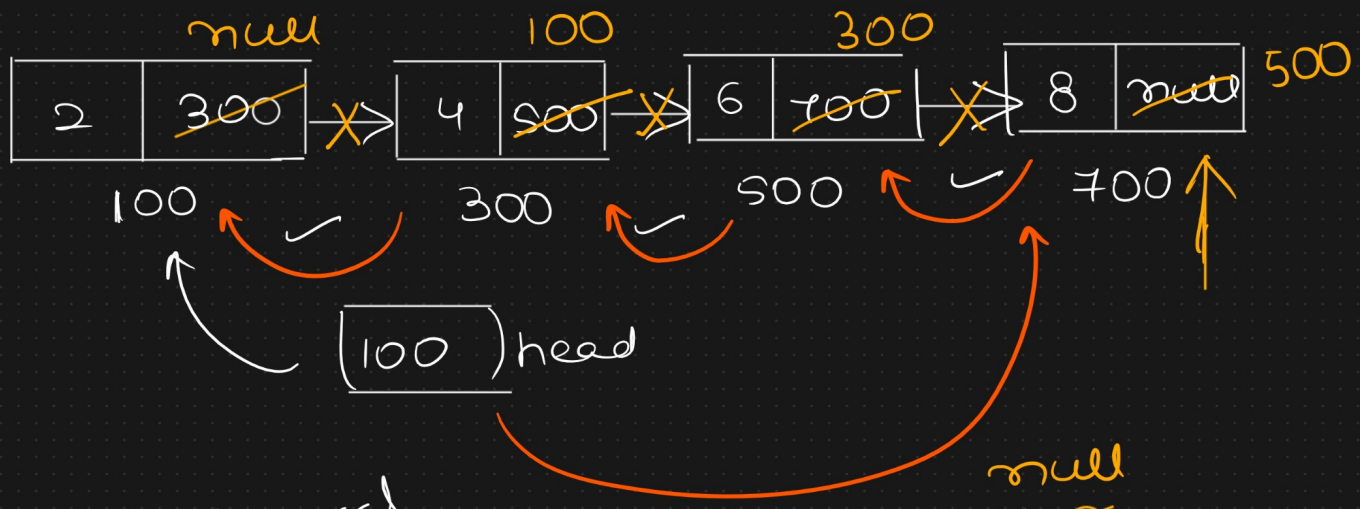
```
while(low < high){
    if(arr(low) == arr(high)){
        low = low+1;
        high = high-1;
    if( arr(low) != arr(high)){
        return false;
        break;
    }
}
return true;
}
```

Approach



More optimized approach

- Time complexity — O(n)
- Space complexity — O(1)



Reversal

```
curr = 100;   300   500   700   null
prev = null;  100   300   500   700
nextPtr = null;  300   500   null
```

Reversal in Linked List

$O(n)$

```
while(curr != null){
    nextPtr = curr.next;
    curr.next = prev;
    prev = curr;
    curr = nextPtr;
}
head = prev;
```

Question 2 → Assignment

https://leetcode.com/problems/reverse-linked-list-ii/