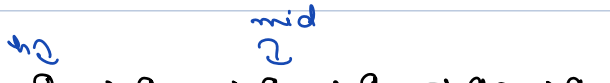
Ques Given a LL, find middle element.

e.g.)

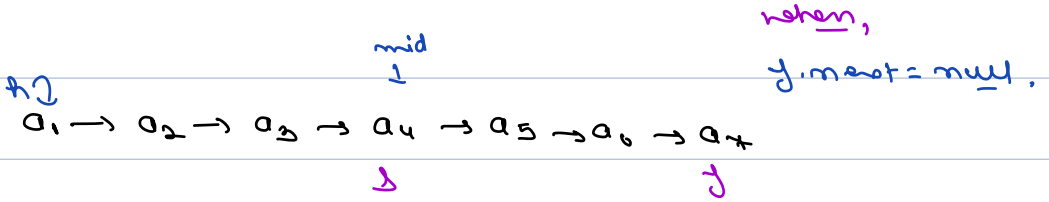


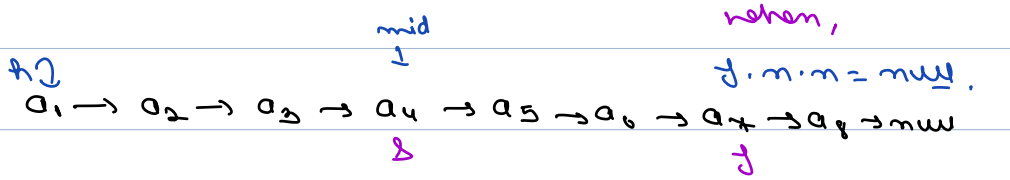
Soln 1:- find size of LL.

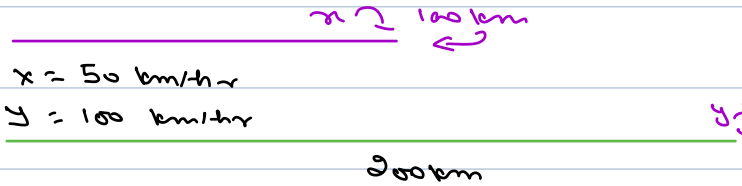
↳ Travel till half of it & return.

T.C → O(n), S.C → O(1).

Constraint :- Do it in 1 iteration

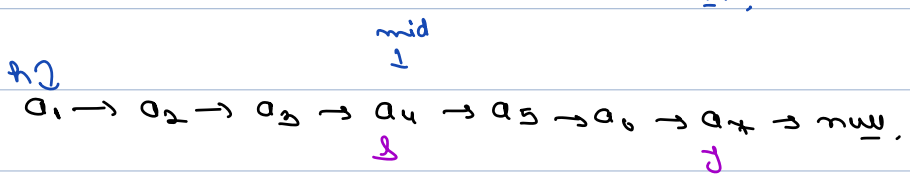


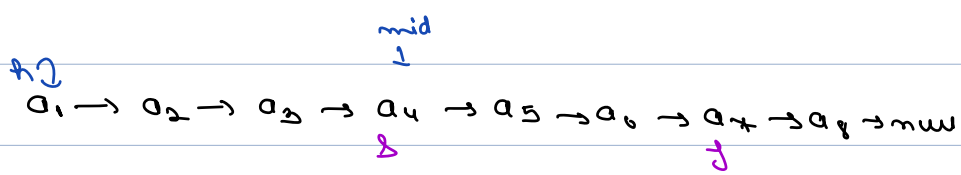


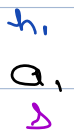


```
node mid (node h) {
    if (h==null) { return null; }
    node s = h;
    node f = h;
    while( f.next!=null && f.next.next!=null){
        s = s.next;
        f = f.next.next;
    }
    return s;
}
```

T.C → O(n)
S.C → O(1).







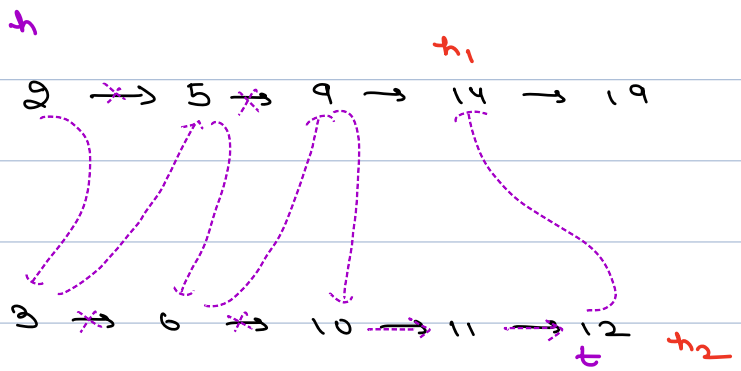
Ques Given two sorted LL, merge them into a single sorted LL.

e.g.) LL1) 2 → 4 → 6 → 8 → 10

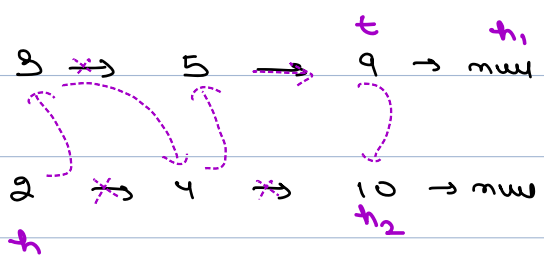
LL2) 1 → 3 → 5 → 7 → 9

O/P → 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9 → 10.

e.g



e.g,



t.next = h1
h1 = h1.next
t = t.next;

t.next = h2
h2 = h2.next
t = t.next

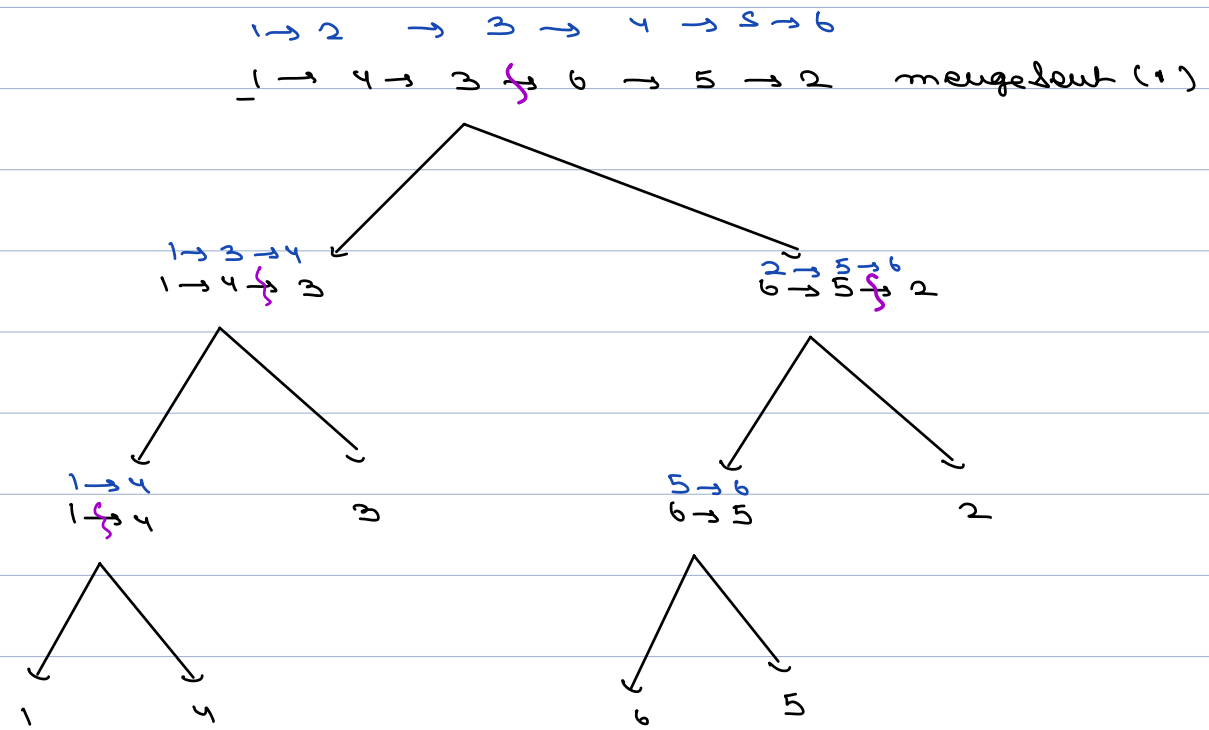
```
Node merge (Node h1, Node h2) {
    if (h1 == null) { return h2; }
    if (h2 == null) { return h1; }

    Node h, t;
    if (h1.data < h2.data) { h = h1, t = h1, h1 = h1.next; }
    else { h = h2, t = h2, h2 = h2.next; }

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

T.C → O(n+m)

S.C → O(1)

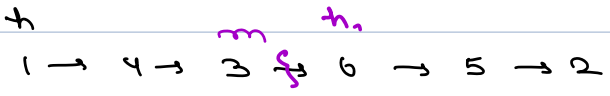

1 → 2 → 3 → 4 → 5 → 6
1 → 4 → 3 → 6 → 5 → 2
mergesort(1)
1 → 3 → 4
1 → 4 → 3
2 → 5 → 6
6 → 5 → 2
1 → 4
1 → 4
5 → 6
6 → 5
3
2
1
4
6
5

Ques Merge sort on LL

I/P $1 \to 4 \to 3 \to 6 \to 5 \to 2$

O/P $1 \to 2 \to 3 \to 4 \to 5 \to 6$

Step1 :- find middle



$h_1 = m.next$

$m.next = null$

Step2 :- call recursion and sort both of them,

$h = mergesort(h);$ h: 1 → 3 → 4

$h_1 = mergesort(h_1);$ h1: 2 → 5 → 6

Step3 :- Merge both sorted LL.

```
Node mergeSort (Node h) {                    → T(n)
    if (h == null || h.next == null) {
        return;
    }
    Node m = middle(h);                      → n
    h1 = m.next;
    m.next = null;
    h = mergeSort(h);                        → T(n/2)
    h1 = mergeSort(h1);                      → T(n/2)
    h3 = merge(h, h1);                       → n
    return h3;
}
```

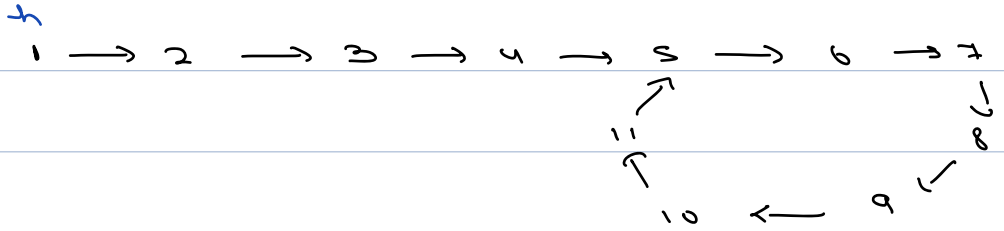
→ merge sort

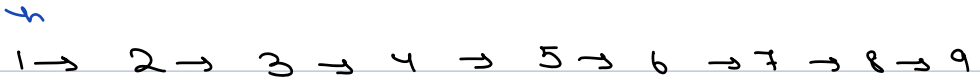
$T(n) = 2T(\frac{n}{2}) + n$

T.C → $O(n \log n)$

S.C → $O(\log n)$

Ques Detect cycle in a LL.

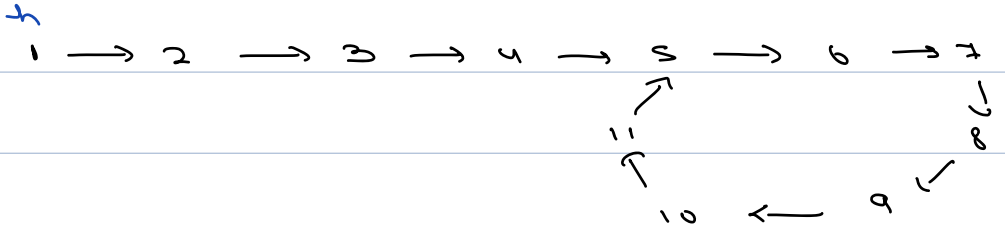




idea :- put a temp at the head.

if temp reaches null (temp = temp.next)

not a cycle cycle



idea 2 :- HashSet < Node >.

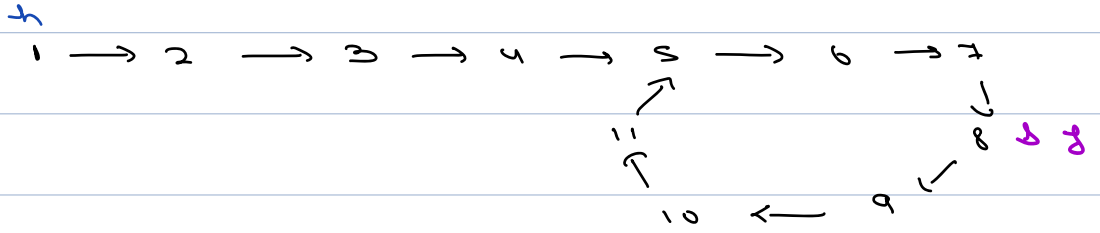
idea :- Iterate on LL, keep on storing nodes. If you reach null ↓ no cycle

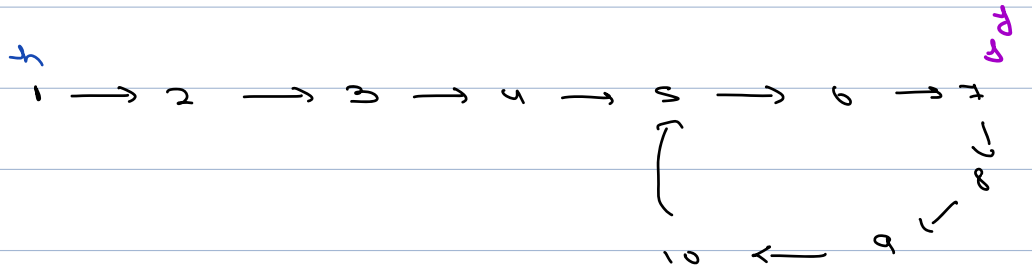
T.C → O(n)

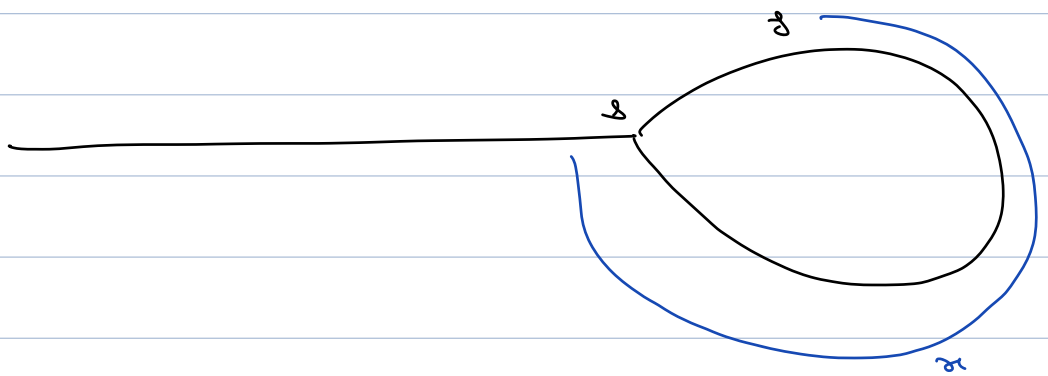
S.C → O(n).

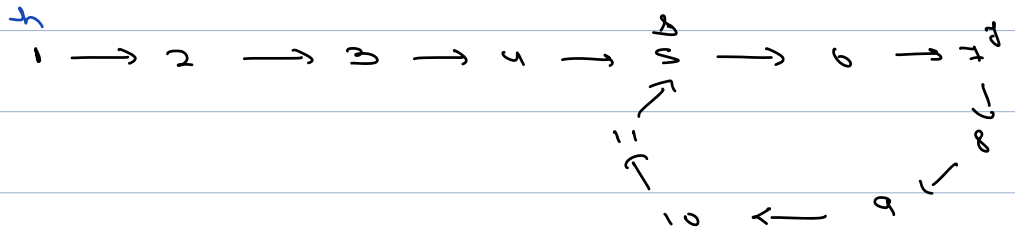
if address starts repeating, there is a cycle.

Idea 2 :-





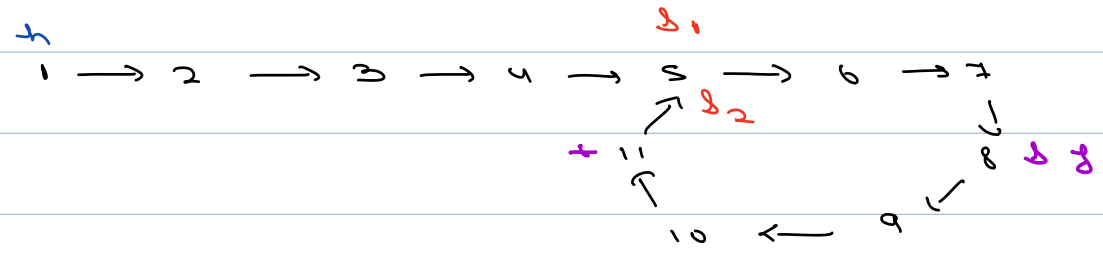


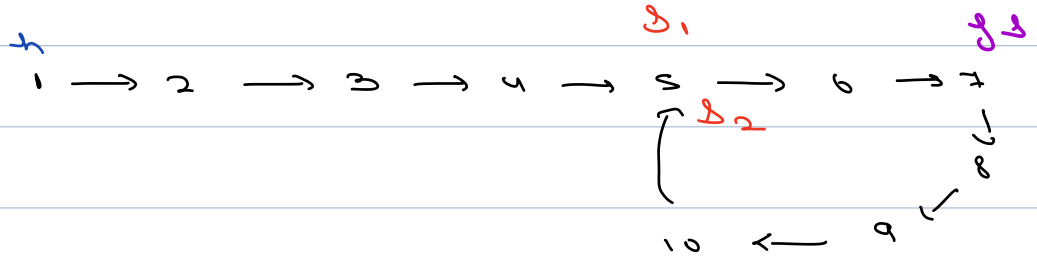


why will they intersect ?

when slow pointer enters the cycle, let's say the distance between slow and fast as X. Now we observe that after every step distance will reduce by one so eventually they will meet .

Ques find Starting Point of a cycle.





Start of cycle

Take two pointers s_1 & s_2.

s_1 → head.

s_2 → Intersection of s & f.

move put the pointers one step at a time. The point there where they will meet will be start of the cycle.

```
bool detectfindRemoveCycle (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;
        }
    }
```

→ O(n)

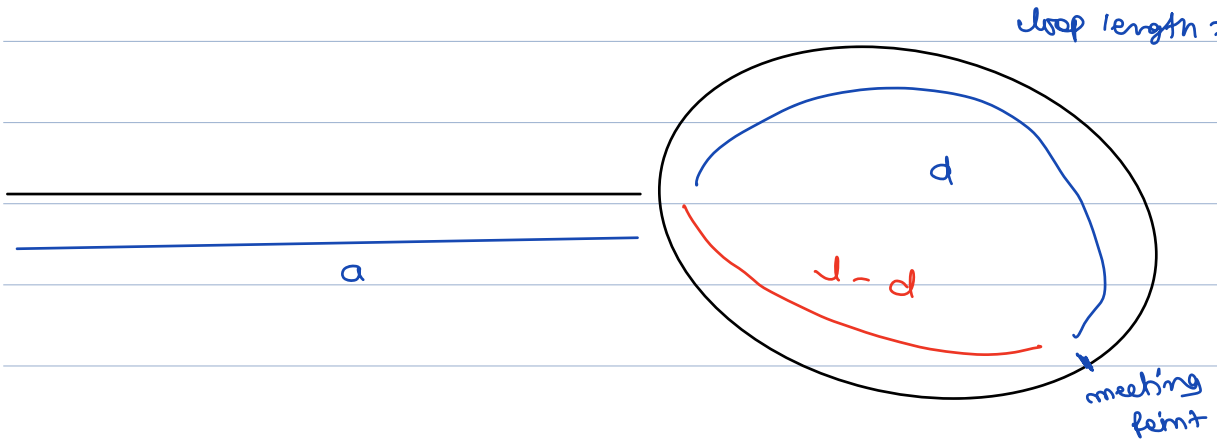
```
    if (isCycle){
        return false;
    }

    node s1 = h, s2 = s;
    while (s1 != s2) { s1 = s1.next, s2 = s2.next}
```

→ O(n)

```
    node t = s1;
    while (t.next != s1) {
        t = t.next;
    }
    t.next = null;
    return true;
}
```

→ O(n)



$d_s = a + d$

$d_f = a + c*l + d$

$d_f = 2 * d_s$.

$a + cl + d = 2a + 2d$

$cl = a + d$

$\Rightarrow a = cl - d + l - l$

$\Rightarrow a = cl - l + l - d$

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

$\Rightarrow a = lx + l - d$