Learn to sell. Learn to build.
If you can do both, you will be unstoppable.

- The Almanack of Naval Ravikant

Today's content

→ LRU Cache

→ Is linked list palindrome?

→ Intersection of two linked-lists.

L.R.U. Cache → temporary memory → small.

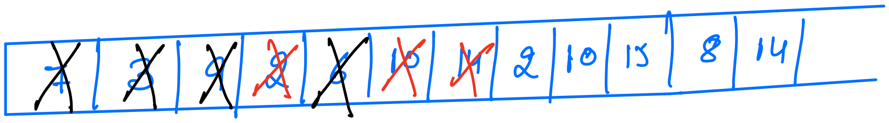
↓

Least Recently Used.

no's → 7 3 9 2 6 10 14 2 10 15 8 14

capacity → 5

size → ~~1~~ ~~2~~ ~~3~~ ~~4~~ 5



→ search

→ insert

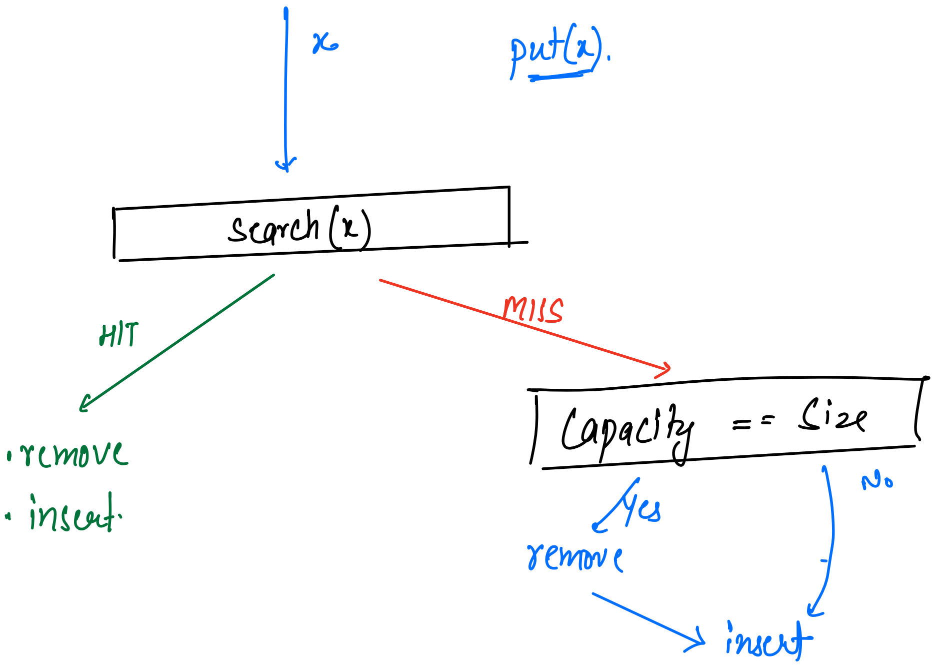
→ remove

HIT.

⇓

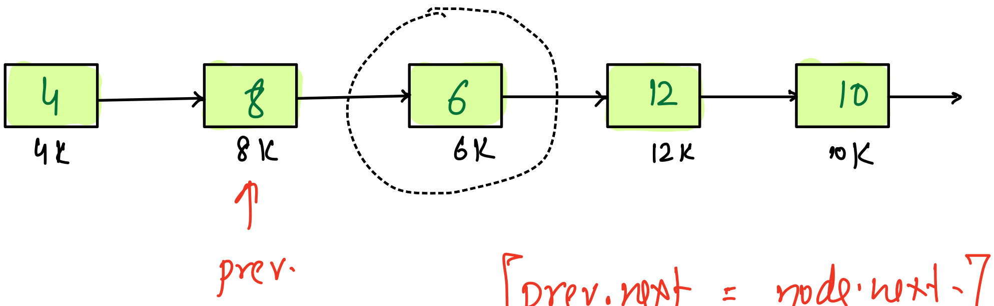
[already present in cache.]

MISS.

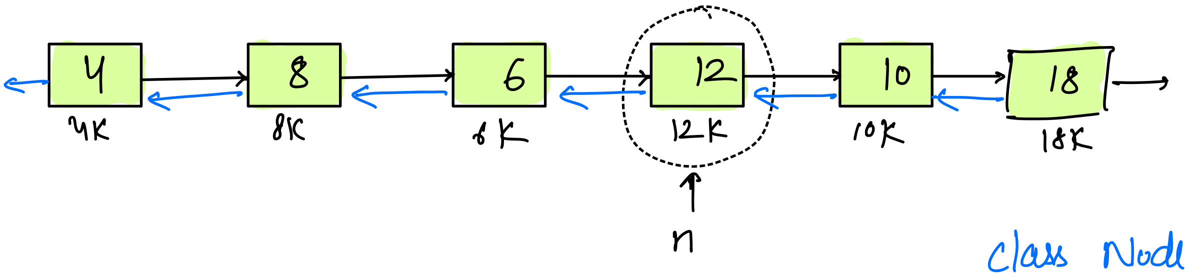


	array	L.L	Hashmap	LL + Hashmap	DLL + Hashmap
search	O(N)	O(N)	O(1)	O(1)	O(1)
insert	O(1)	O(1)	order is not maintained.	O(1)	O(1)
remove	O(N)	O(1) ⇓		O(N)	O(1)

[traversal is already happening in searching part.]



[prev.next = node.next.]



D.L.L.

[6K.next = n.next
10K. prev = n.prev]

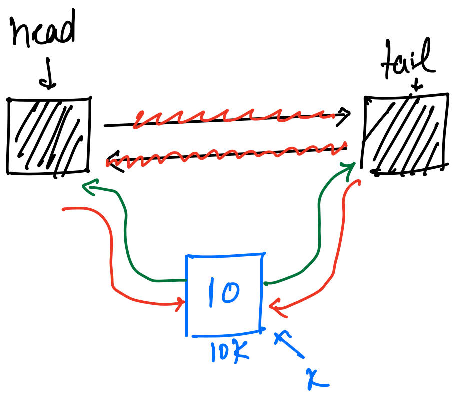
```
class Node {
    int val;
    Node next;
    Node prev;
```

Eg. → 10 15 19 20 15 18 23 20 19

head → ~~null~~
tail → ~~null~~

Capacity → 5.

Size → ~~0~~ ~~1~~ ~~2~~.
~~3~~ ~~4~~
5



HashMap.

<int , reference of Node>

~~< 10, 10K. >~~

< 15 , 15K >

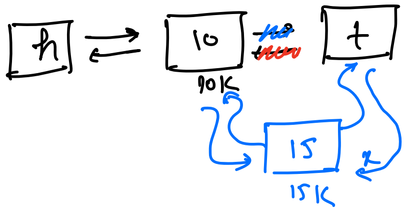
< 19, 19K >

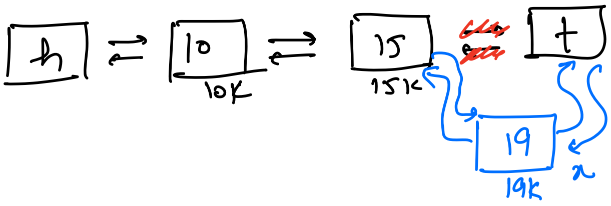
< 20, 20K >

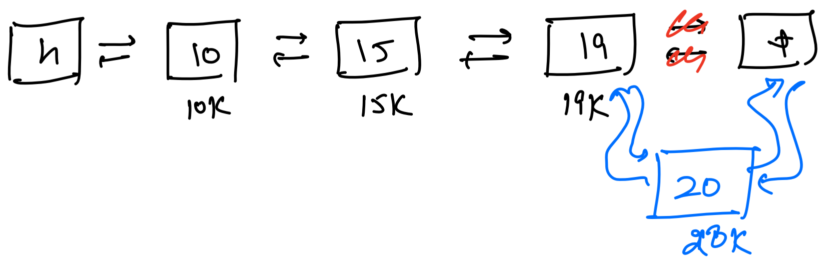
< 18, 18K >

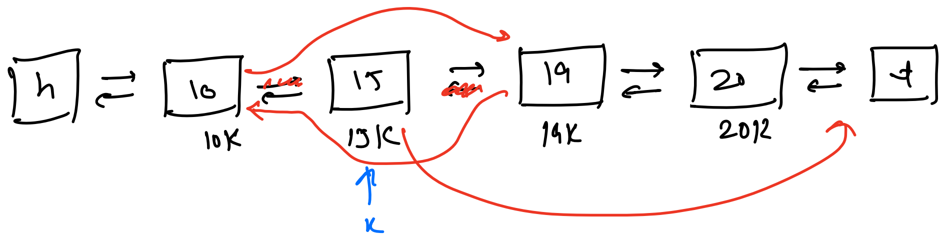
< 23, 23K >

```
void addToTail (Node x) {
        x.next = tail          ①
        x.prev = tail.prev     ②
        tail.prev = x          ③
        x.prev.next = x        ④
}
```



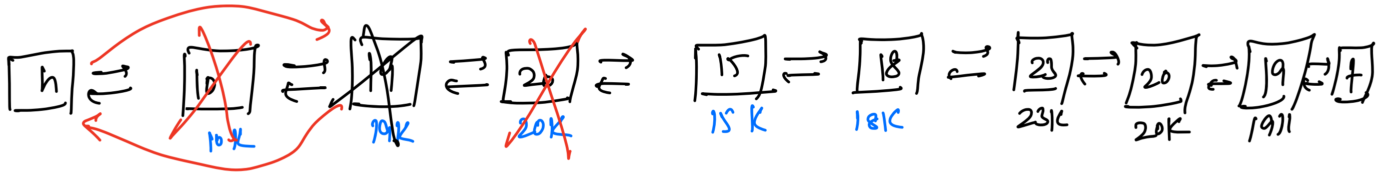




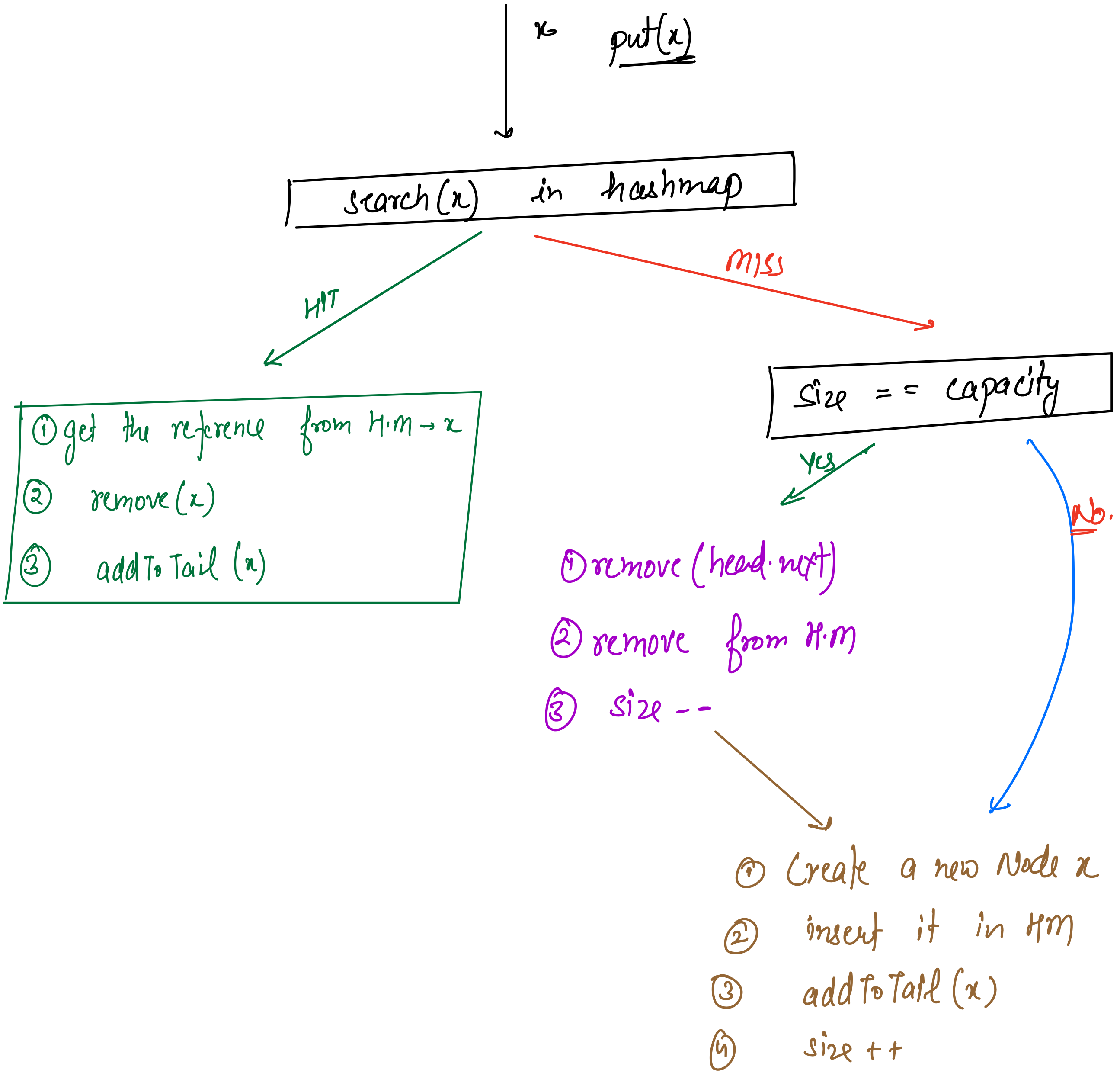


add to Tail (x):

```
remove (Node x) {
    x.prev.next = x.next
    x.next.prev = x.prev
}
```

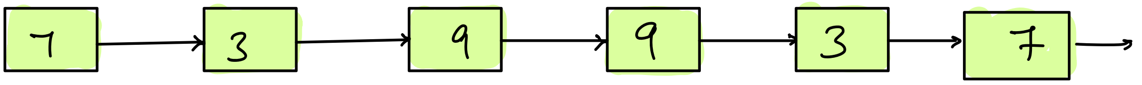


- → insert
- → search
- → remove


x
put(x)
search(x) in hashmap
HIT
MISS
① get the reference from H.M → x
② remove(x)
③ addToTail(x)
size == capacity
Yes
No.
① remove(head.next)
② remove from H.M
③ size --
① Create a new Node x
② insert it in HM
③ addToTail(x)
④ size ++

L.R.U Cache.

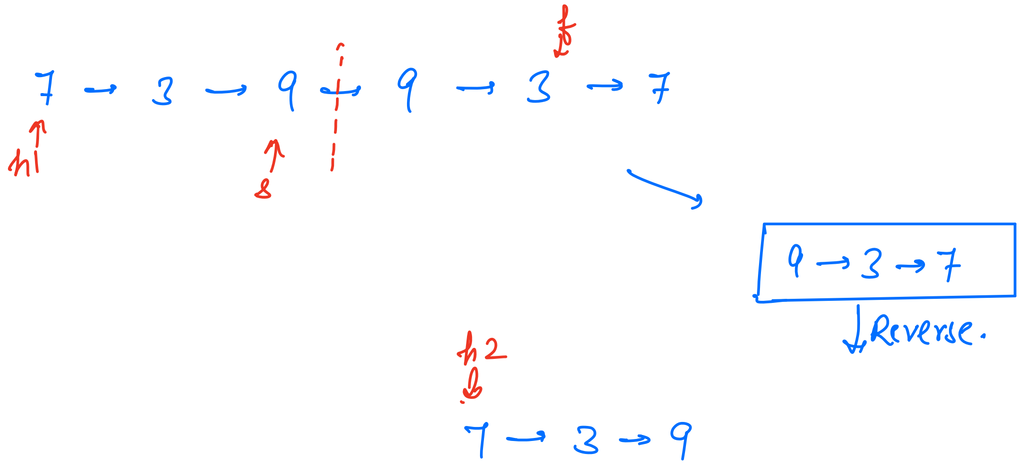
② Is linked list a palindrome ?



① extra space   ll to array

7	3	9	9	3	7

② No extra space.



① Find middle node. [using slow & fast]

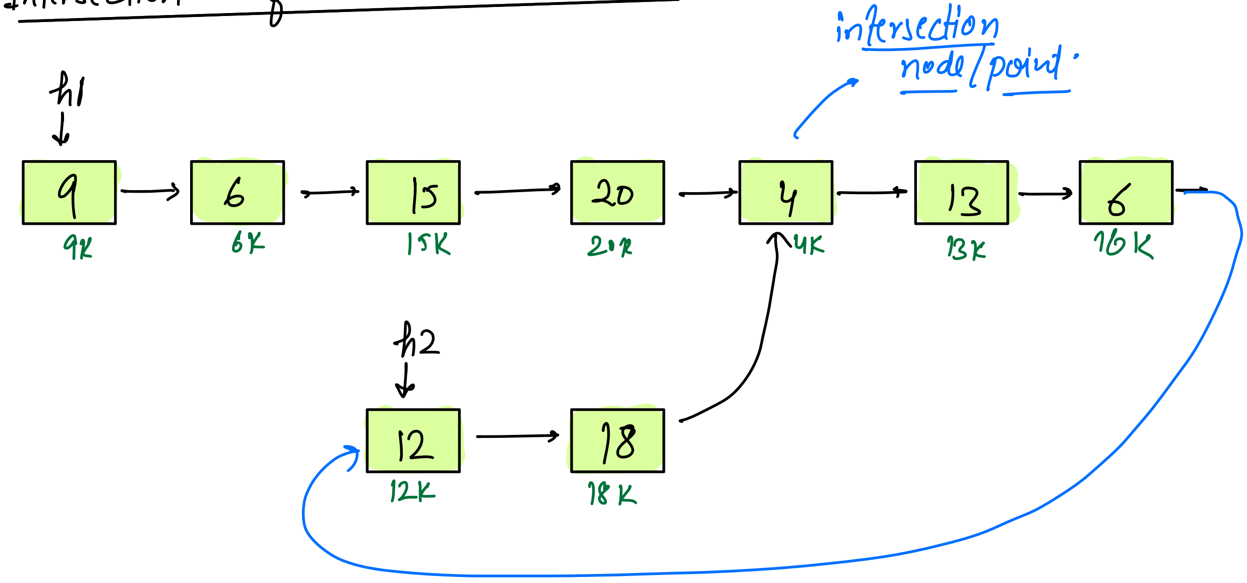
② h1 → head,   h2 = slow.next   [slow.next = null]

③ Reverse the second linked list. [ReverseLL(h2)]

④ Compare two linked-lists.

⑤ Re-construct the linked list. ← [ReverseLL(h2) slow.next = h2]

Intersection Of Two Linked - Lists →



idea.1. Using Hashset/ hashmap.

T.C → O(N)
S.C → O(N)

9K
6K
15K
20K
4K → intersection node.
13K
16K

idea.2.

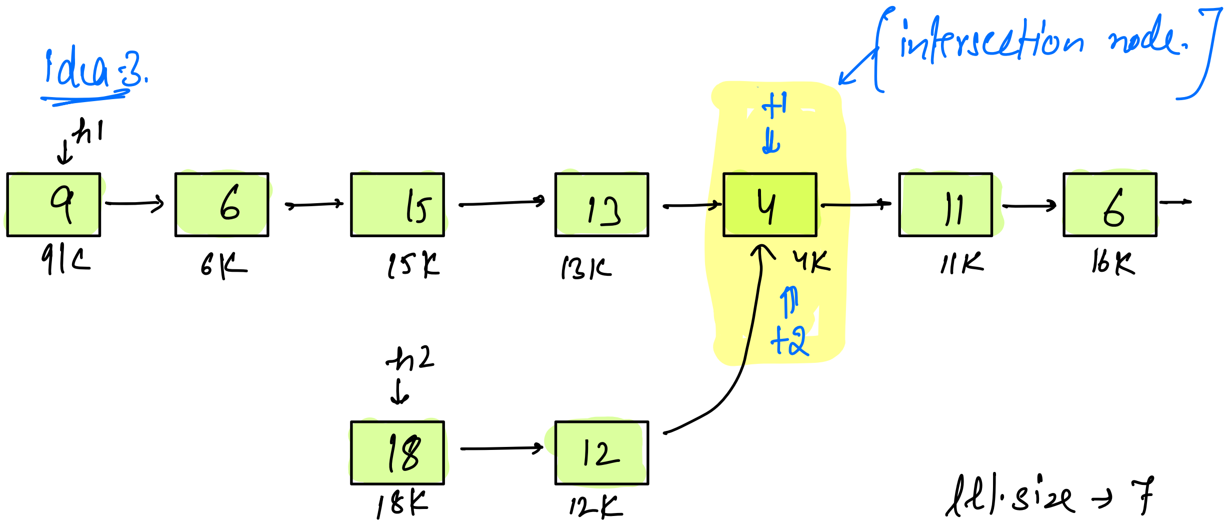
① Create a connection b/w tail of one linked-list & head of another linked-list. (loop will be created)

② find the starting point of loop. ⇒ Intersection point of 2 L.L's.

③ Break that connection (built in ①)

T.C → O(N)
S.C → O(1)

idea 3.



ll1.size → 7
ll2.size → 5

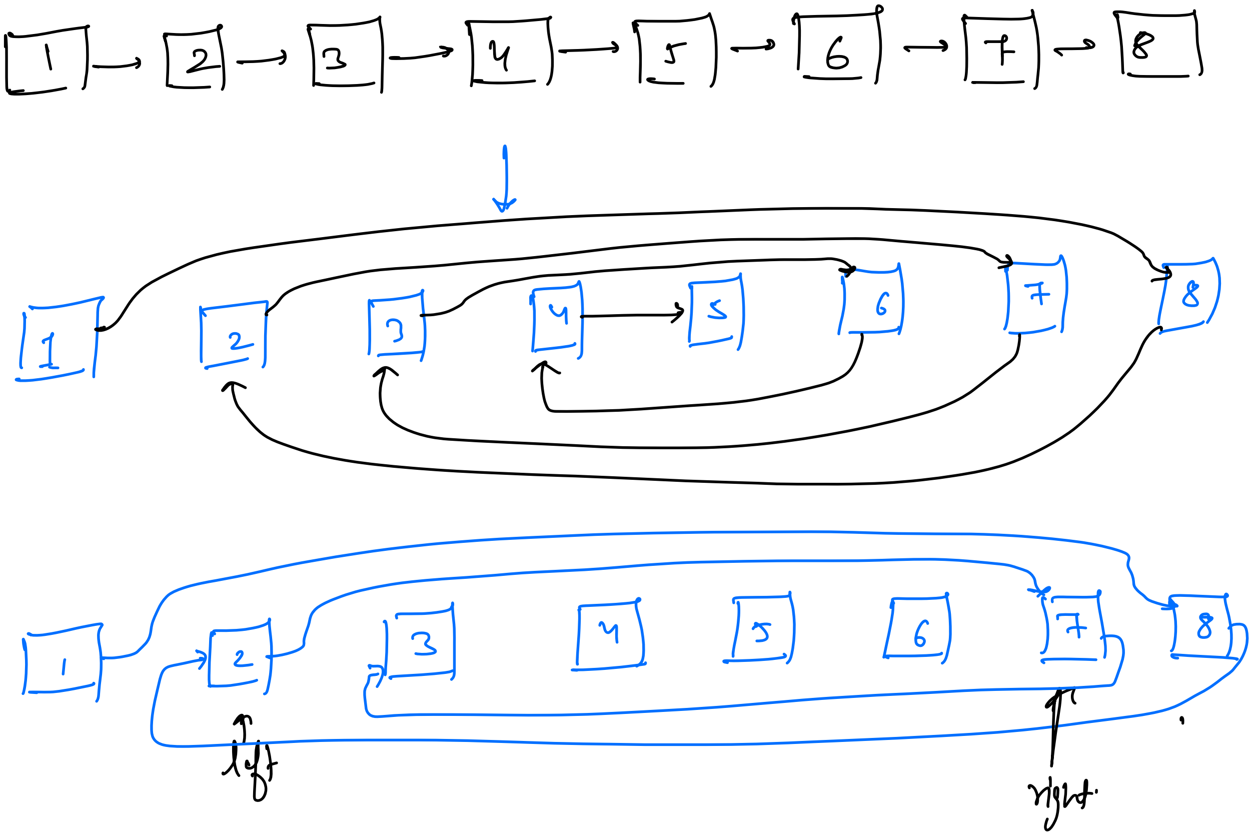
diff → 2.

idea 3

① Find the size of two linked-lists.

② Take t1 = h1, t2 = h2
and move the pointer corresponding to longer linked-list by diff = | ll1.size - ll2.size | steps.

③ Move t1 & t2 simultaneously by one step at a time till they are pointing to same node reference.


1
2
3
4
5
6
7
8
1
2
3
4
5
6
7
8
1
2
3
4
5
6
7
8
left
right