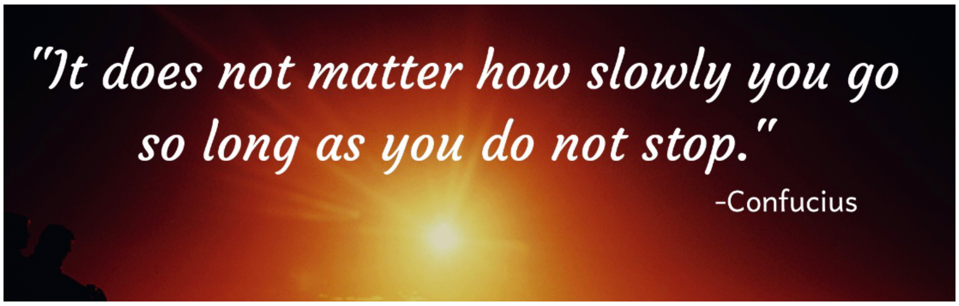

"It does not matter how slowly you go so long as you do not stop."
-Confucius

Today's content

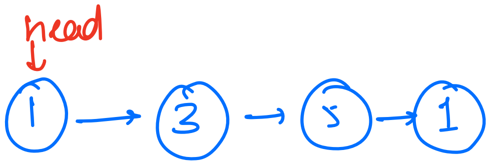
→ Is linked list palindrome?

→ Longest odd length palindrome in l.l ?

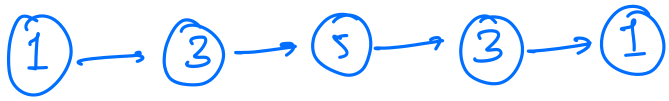
→ LRU Cache

Q: Check if the given linked-list is a palindrome?

S.C → O(1)



false.



true.

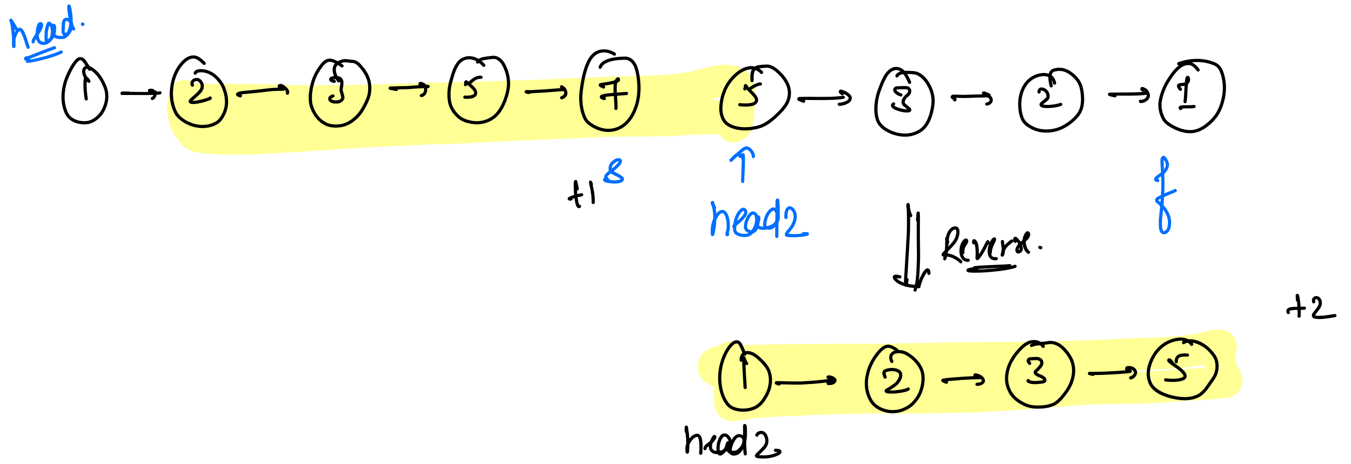
idea-1. → Store all the 'elements in arr[] and check if it is palindrome or not.

T.C → O(N)
S.C → O(N)

1. Find middle node (using slow & fast pointer)
2. head2 = slow.next , slow.next = NULL;
3. head2 = reverse(head2);
4. Compare the two linked lists.
5. head2 = reverse(head2);
6. Make the connection b/w the linked-lists.
   slow.next = head2;

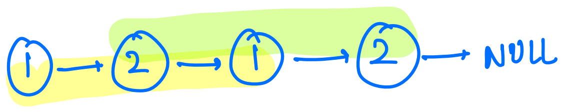
Re-constructing the original linked-list.

T.C → O(N)
S.C → O(1)

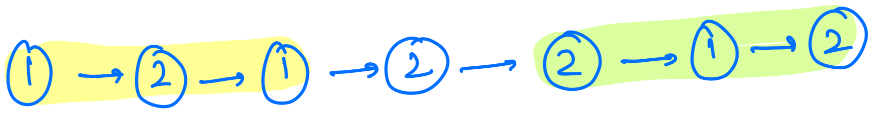


```
t1 = head,   t2 = head2 , ans = true
while( t2 != NULL){
        if( t1. val != t2.val){
            ans = false , break
        }
        t1 = t1.next;
        t2 = t2.next;
}
```

Q1 Find the length of longest odd length palindromic list in the given linked list.



ans = 3



ans = 3

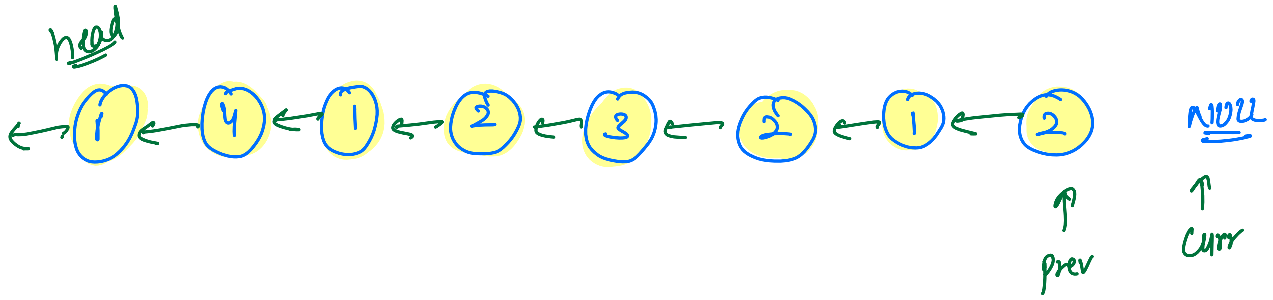
idea-1 →

1	2	1	2	2	1	2

Consider all subarrays of odd length & check whether it is a palindrome or not. T.C → $O(N^3)$, S.C → $O(N)$

idea-2. Consider every element as middle element & try to expand on both the sides. [T.C → $O(N^2)$, S.C → $O(N)$]

idea-3 Reverse the linked-list on the go.



ans = ~~1~~ ~~3~~ 5

```
prev → NULL, curr → head;
while(curr != NULL){
    temp = curr.next;
    count = matchingCount(prev, temp);
    ans = max(ans, 2*count + 1);

    curr.next = prev;
    prev = curr;
    curr = temp;
}
head = reverse(prev);
return ans;
```

```
int matchingCount(Node t1, Node t2){
    count = 0;
    while(t1 != NULL && t2 != NULL){
        if(t1.val == t2.val){
            count++;
        }
        else{
            break;
        }
        t1 = t1.next;
        t2 = t2.next;
    }
    return count;
}
```

T.C → $O(N^2)$
S.C → $O(1)$

How to find longest even length palindrome in given l.l ?

⇓

two middle points.

↙ prev, curr

↘ curr, temp.

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 = ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ 5

~~7~~	~~3~~	~~9~~	~~2~~	~~6~~	~~10~~	~~14~~	2	10	15	8	14

HIT.

when data is already present in cache.

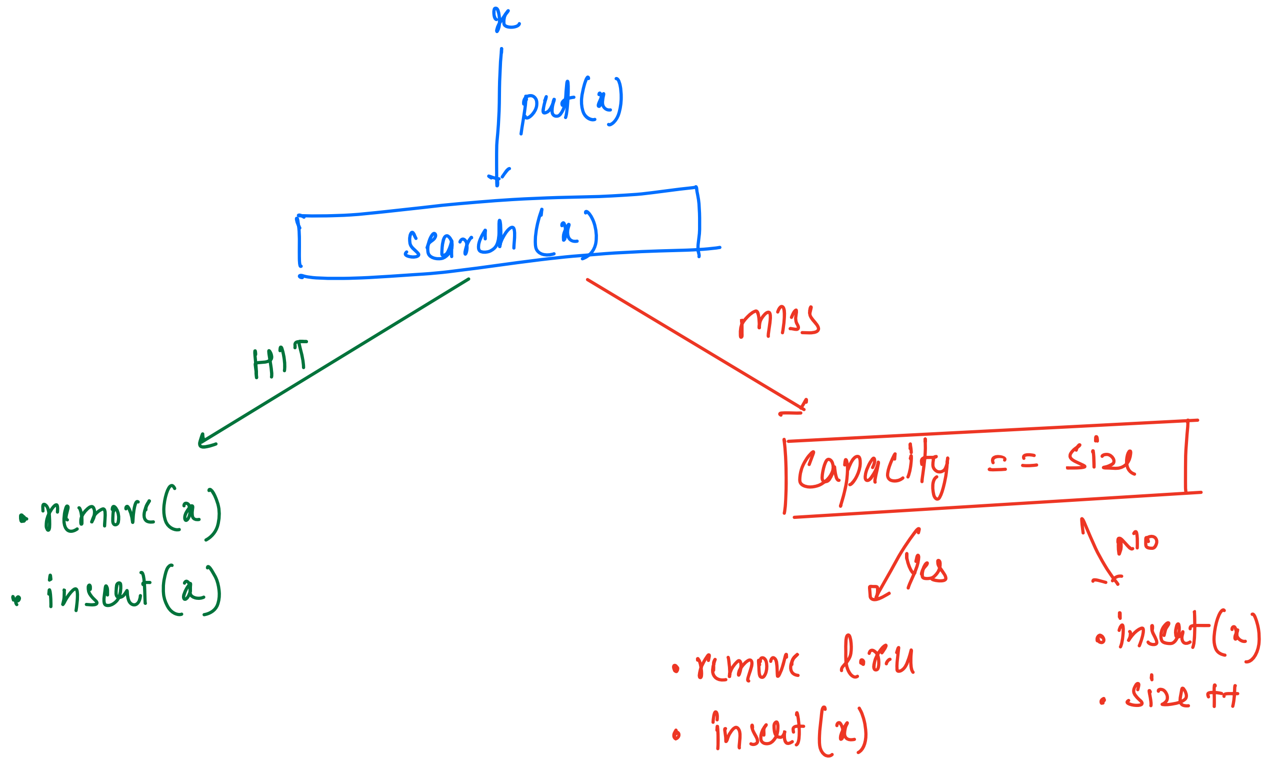
MISS.

when data is not already present in cache.

→ search

→ remove

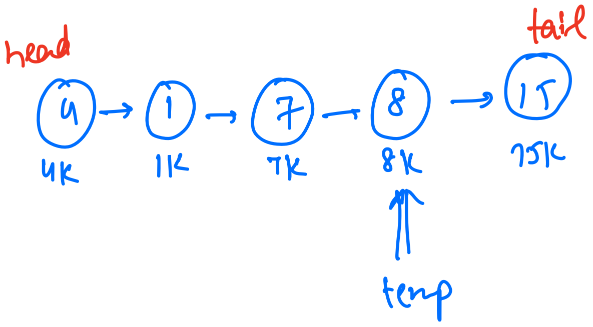
→ insert



	Array	Linked-list	Hashmap/ Hashset	L.L+ Hashmap	D.L.L+ Hashmap
Search	O(N)	O(N)	O(1)	O(1)	O(1)
insert	O(1)	O(1)	order is	O(1)	O(1)
remove	O(N)	O(1)	not maintained	O(N)	O(1)

⇓

As traversal is already happening in search().



value		Node reference
4	→	4K
1	→	1K
7	→	7K
8	→	8K
15	→	15K

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

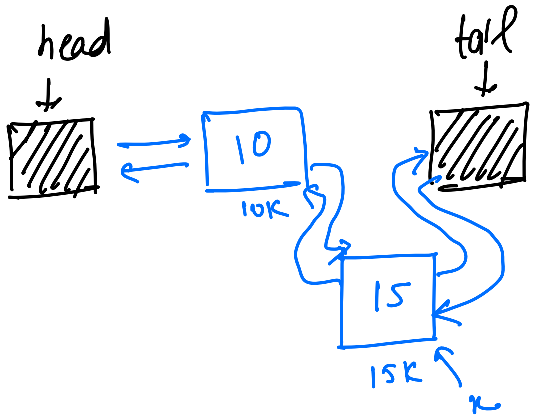
Eg →

	✓	✓	✓	✓	✓	✓	✓	
[	10	15	19	20	15	18	23	20]
	0	1	2	3	4	5	6	7

Cap → 5

size → ~~0~~ ~~1~~ ~~2~~ ~~3~~
~~4~~ 5

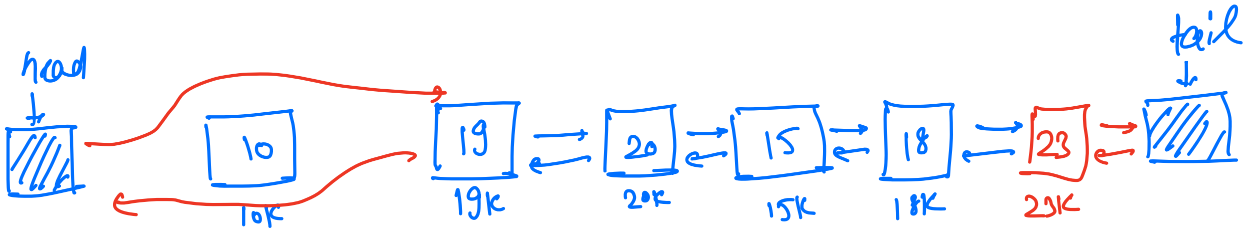
head → null
tail → null



Hashmap

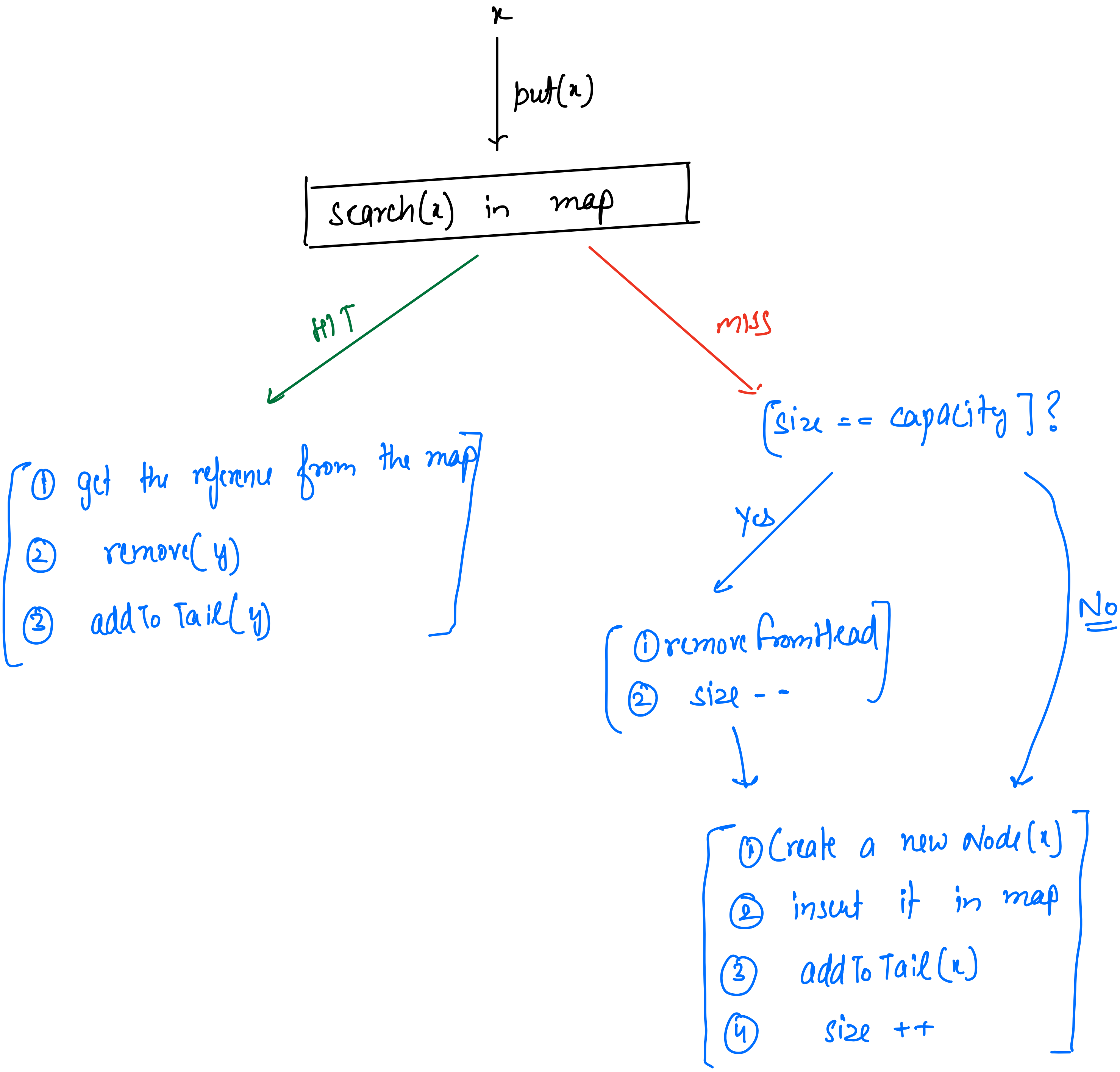
integer		node reference
~~10~~	~~→~~	~~10K~~
15	→	15K
19	→	19K
20	→	20K
18	→	18K
23	→	23K

```
void addToTail(Node x){
        x.next = tail;
        x.prev = tail.prev;
        tail.prev = x;
        x.prev.next = x;
}
```



```
Node y → map.get(15);

remove( Node y){
        y.prev.next = y.next;
        y.next.prer = y.prev;
}

 addToTail(y);

void removefromHead( ){
        map.remove(head.next);
          remove(head.next);
}
```

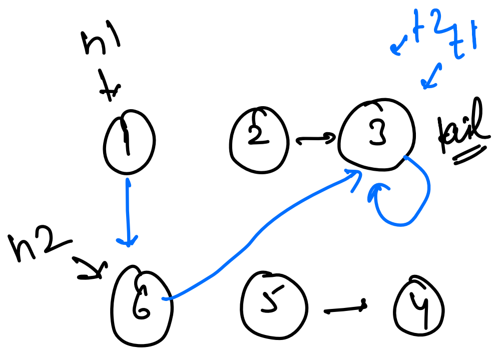
```
class LRU {
     DLLNode head, DLLNode tail;
      size = 0
      Hashmap< Integer, DLLNode > map;

      insert() {
      }
      remove() {
      }
      removefromHead() {
      }
      search()
}
```

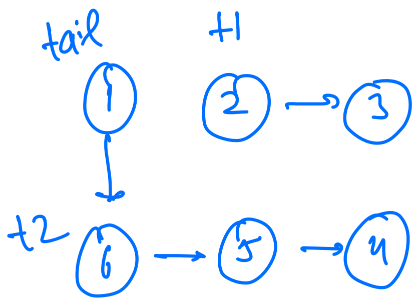


L.R.U

[code → # todo]

→ target → attempting all L.L problems by Tuesday.





while(t1 != null && t2 !=null) {
tail.next = t2;
tail = t2;
~~t1 = t1.next~~; t2 = t2.next ;
tail.next = t1;
tail = t1;
~~t2 = t2.next~~; t1 = t1.next ;
}
tail.next = t2;