PSP → 63.62 %

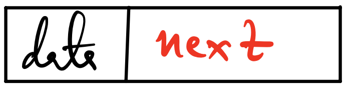
⇓ Saturday (70%)

⇓ Sunday (3 hrs)

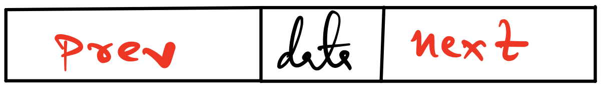
⇓ LL, BS, Sorting

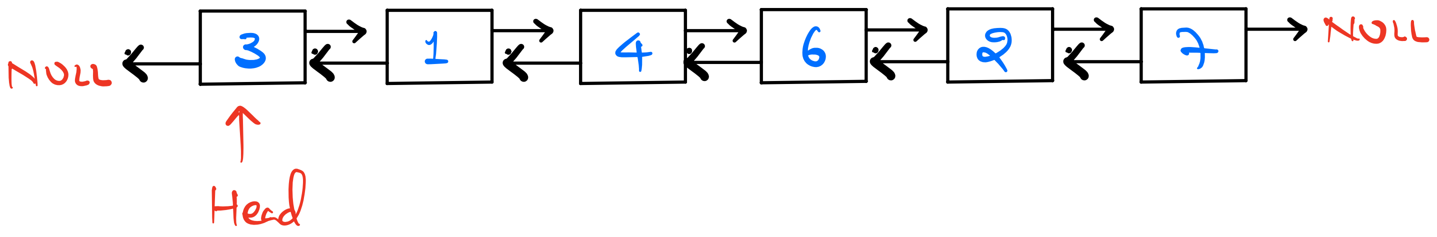
Doubly Linked List

LL ⇒



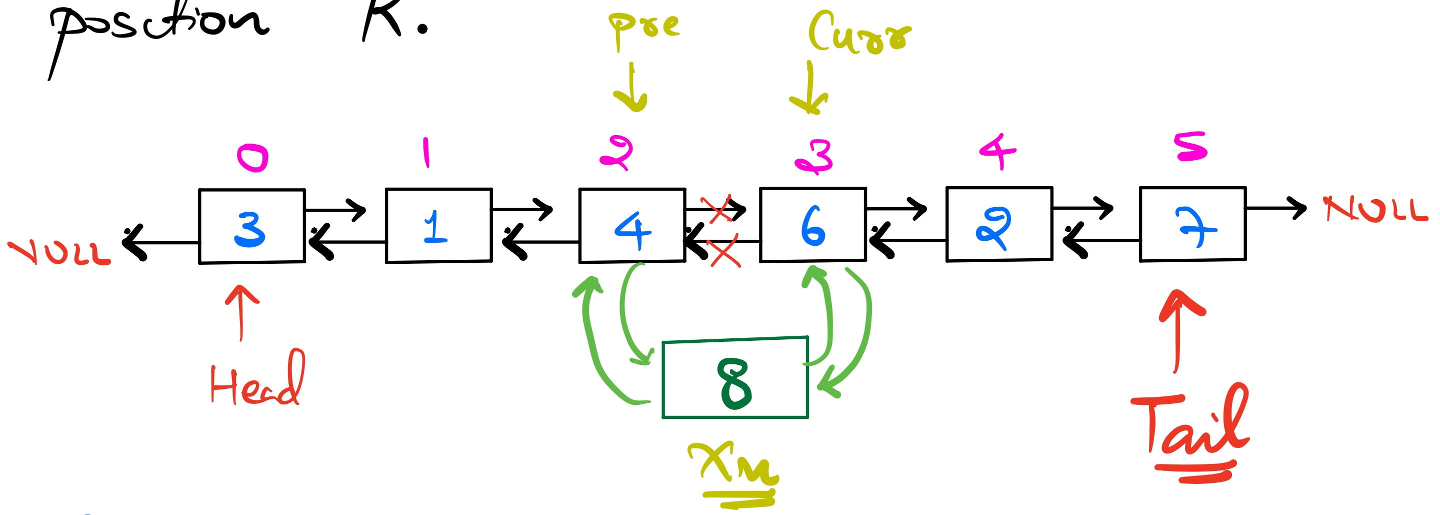
DLL →





```
class Node {
    int data;
    Node prev;
    Node next;
    Node (int x){
        data = x;
        prev = NULL;
        next = NULL;
    }
}
```

Q Given the Head Node of a DLL insert a node with data x at position K.



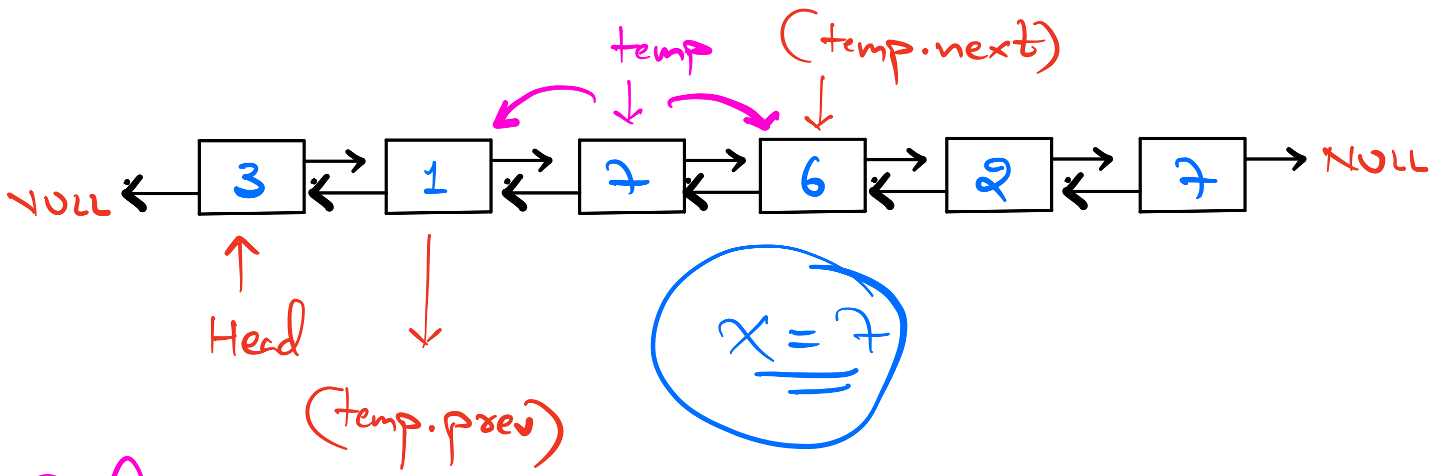
$x = 8$
$K = 3$

Code

```
Node insert (Node Head, x, K) {
    Node xn = new Node (x);
    if ( Head == NULL) {
        return xn;
    }

    if (K==0) {
            xn.next = Head;
            Head.prev = xn;
            return xn;
    }

   Node curr = Head.next
   Node pre = Head;
   for(i=0; i<(K-1) ; i++) {

           curr = curr.next;
           pre = pre.next
   }
   pre.next = xn;
   xn.prev = pre;

   xn.next = curr;
   if (curr != NULL) {
          curr.prev = xn;
   }
   return Head;

}
```

T.C. = O(N)
S.C. = O(1)

Q Given a DLL of length N.
Delete the first occurence of data X in the LL.

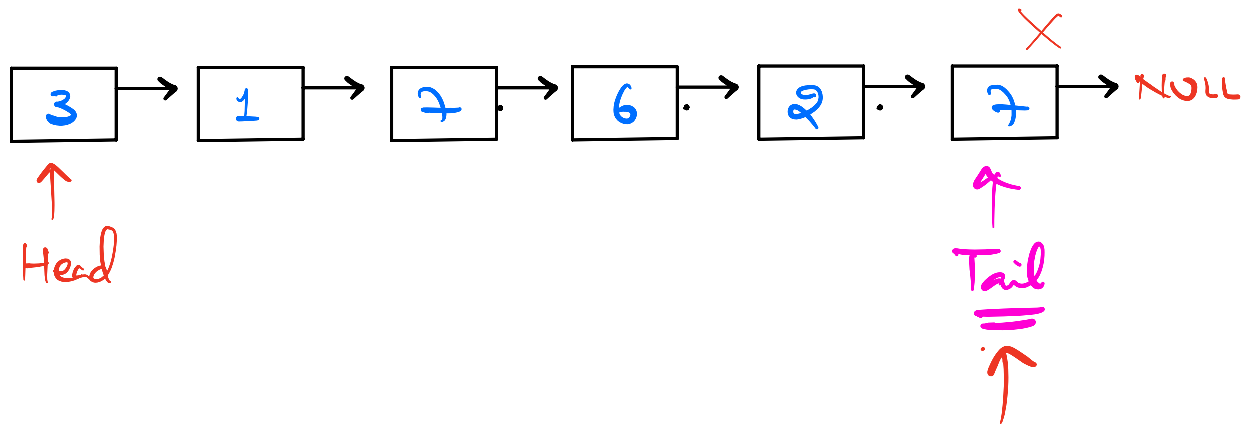


Code

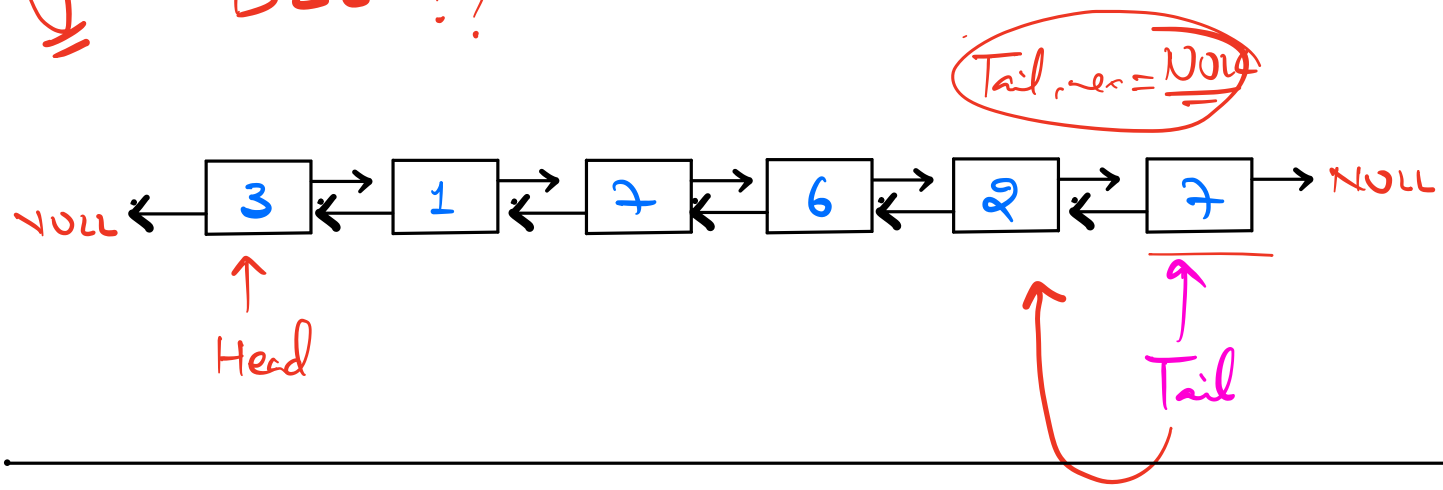
```
Node delete (Node Head, int x) {

    if (Head == NULL) {
                return Head;
    }

    if (Head. data == x) {
                Head = Head. next;
                Head. prev = NULL;
                return Head;
    }

    Node temp = Head. next;

    while (temp != NULL) {
        if (temp.data == x) {
                temp.prev.next = temp.next;
                if (temp.next != NULL) {
                    temp.next.prev = temp.prev;
                }
                break;
        }
        temp = temp.next;
    }
    return Head;
}
```

T.C. = O(N)
S.C. = O(1)

Q. Can we optimise deletion at the last pos in a SLL.



Q. DLL ??



Music App → — Spotify
Jio Saavn
Wynk
Apple Music
— YT Music
Amazon

⇓

Cache Memory

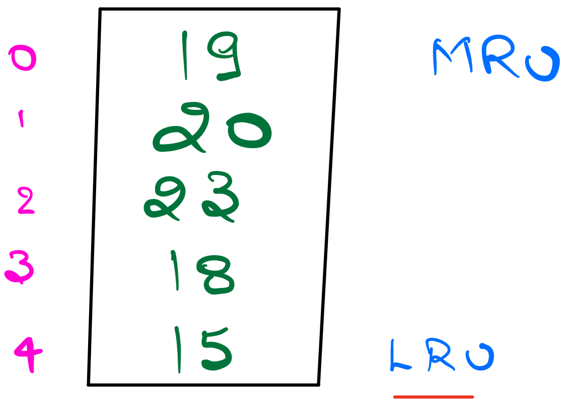
⇓

Eviction Strategy ⇐ LRU (Least Recently Used)

Input ⇒ Running Stream of integers

Cache Size = 5

Eg :→ 10, 15, 19, 20, 18, 23, 20, 19, 17, 17, 10.
↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑



I/p (x)

x is present

1) Remove x from cache (Delete from random index)
2) Insert x back to MRU (Insertion at the front)

x is not present in cache

Cache is not full

1) Insert x at MRU (front)

Cache mem is full

1) Delete element at LRU (End)
2) Insert x at MRU (front)

	Array ✗	LL	DLL ✓
Search	O(N)	O(N) O(1) ⇒ H.M.	O(N) O(1) ⇒ H.M.
Insert at front	O(N)	O(1)	O(1)
Delete	O(N)	O(1) O(N) ⇒ Iterate to get prev	O(1) O(1)

Searching can be optimised in LL & DLL using a HashMap.

$$\begin{array}{cc} \text{HashMap} < \underset{\downarrow}{\text{int}}, & \underset{\downarrow}{\text{Node}} > \\ \text{Element} & \text{Reference} \end{array}$$

Code

capacity (i/p)

```
Node Head = NULL;
Node tail = NULL;

HashMap <int, Node> hm;
size = 0;

void insert ( int x) {

    if ( hm. containsKey( x)) {

        Node temp = hm.get(x);
        if (temp== Head) { return; }

    else ( temp.prev != NULL) {
            temp.prev.next = temp.next;
    }
```

```
        if (temp.next != NULL) {
            temp.next.prev = temp.prev;
        }


        temp.next = Head;
        Head.prev = temp;
        temp.prev = NULL;
        Head = temp;
    }
    else {
        if (size == capacity) {

            temp = Tail.prev;
            temp.next = NULL;
            Tail = temp;
            size --;
            hm.delete(temp.data);
        }
        Node xn = new Node(x);
        xn.next = Head;
        if (Head != NULL) {
            Head.prev = xn;
        }
        Head = xn;
        size ++;
        hm.insert(x, xn);
    }
}
```

H.W. $\Rightarrow$ figure out how to update **Jal**.