LINKED LIST

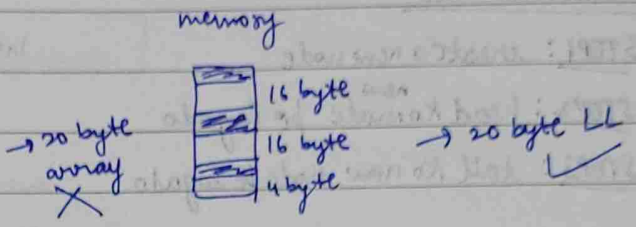
Linked list class-1 (Live class)

15 July 2024

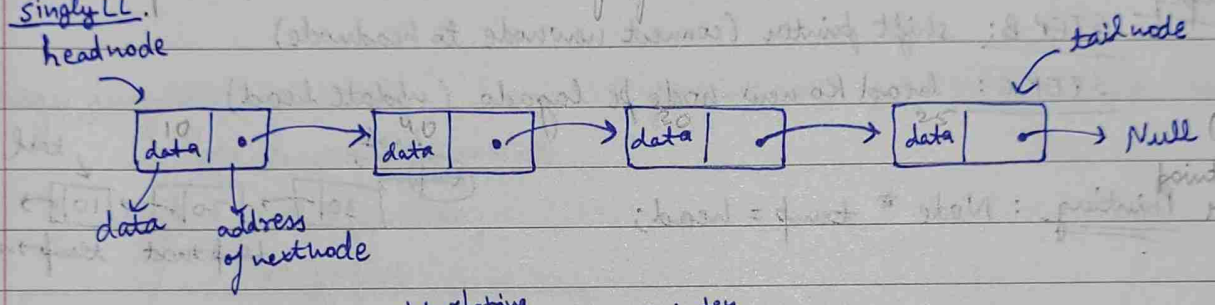
→ not asked as such but it's easy has only few types of problems

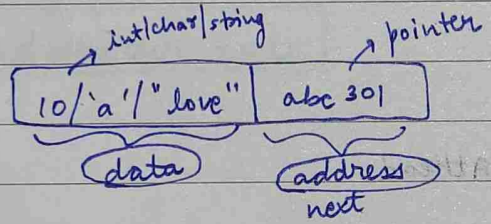
- collection of nodes (connected in a specific manner)
- array : continuous space

  linked list : non-continuous
  ↳ jayda hai



HW. → compaction, internal & external fragmentation. (read about it)



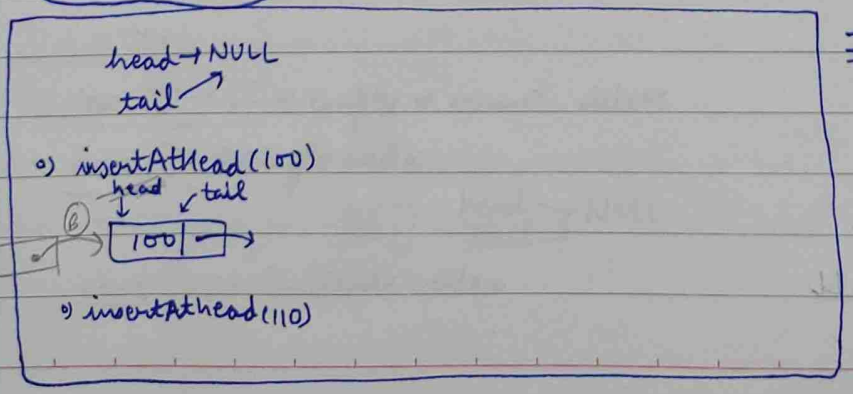



```
class node { int data;
             node * next;
           }
```

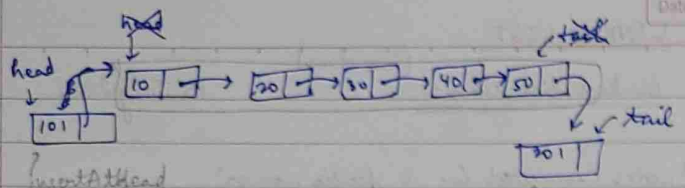
main ⇒ Node* first = new Node(); → dynamic memory

//stack ⇒ Node first.

⇒ L.L is Hindi



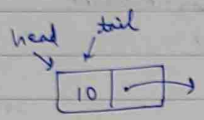
Insertion
- ↳ insertAtHead → insert at left-most position
- ↳ insertAtTail → insert at right-most position
- ↳ insertAtPosition



INSERTION

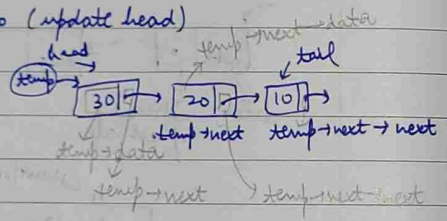
insertAtHead

- LL is empty
  - STEP 1: create a new node
  - STEP 2: head ko new node pe lagado
  - STEP 3: tail ko new node pe lagado



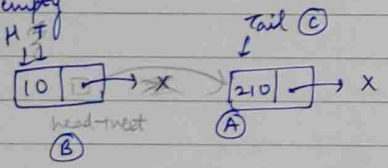
- LL is not empty
  - STEP A: create a new node
  - STEP B: shift pointer (connect newnode to headnode)
  - STEP C: head ko new node pe lagado (update head)

Printing : Node * temp = head;



insertAtTail

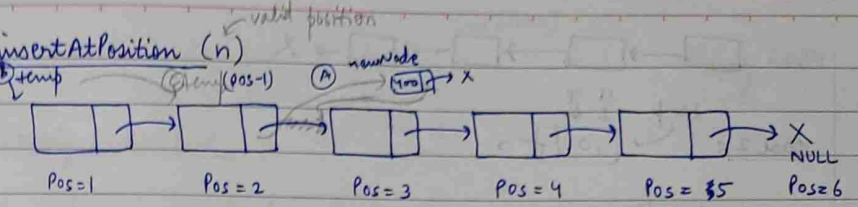
- LL is empty → same as insertAtHead
- LL is not empty



STEP A : create a new node
STEP B : tailnode ko new node se connect kro
STEP C : tail update kro

- L.L
  - Types :
    - Singly LL
    - Doubly LL
    - Circular LL

insertAtPosition (n)



[Pos: <1 or >6 → invalid positions]

- insertAtPos(1) → insertAtHead
- insertAtPos(6) → insertAtTail
- insertAtPos(3) =?

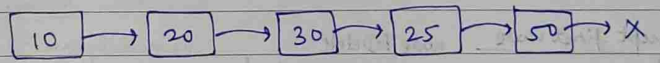
eg: insertAtPos(3,400); for(int i=0; i<pos-2) { temp = temp→next; }

newNode→next = temp→Next
temp→next = newNode;

- ORDER OF STEPS IS IMPORTANT IN LL

Searching

eg:



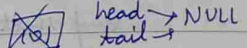
target = 25
ans → bool : T/F

⇒ Linear Search ✓

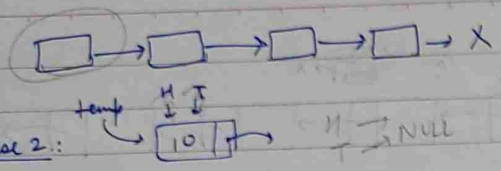
HW → return pos" at which target is found

DELETION

case ① → LL is empty ⇒ cannot delete
case ② → single node



case ③ → multiple nodes

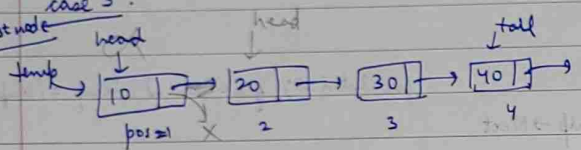


case 2 :

```
Node * temp = head;
head = NULL;
Tail = NULL;
delete temp;
```

case 3 :

① First node

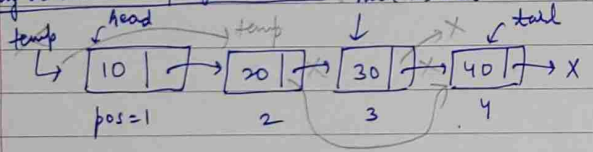


⇒ delete Pos (1)

```
Node* temp = head;
head = temp->next;
temp->next = NULL;
delete temp;
```

⇒ first target should be to isolate the node you want to delete.

② any other node except first node



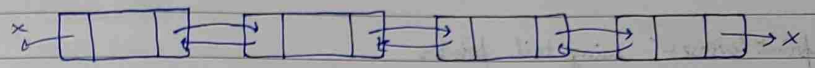
⇒ deletePos(3)

```
Node * temp = head;
for (i=0; i<pos-2; i++){
    temp = temp->next;
}
Node * nodetoDelete = temp->next;
temp->next = nodetoDelete->next;
nodetoDelete->next = NULL;
delete nodetoDelete;
```

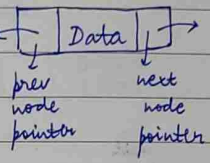

(live class)

Linked List - 2 Class 2 — 17 July 2024

Doubly linked list

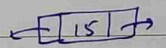


DLL :




```
class Node { Node* prev;
             int data;
             Node* next;
           }
```

```
Node * newNode = new Node(15);
```



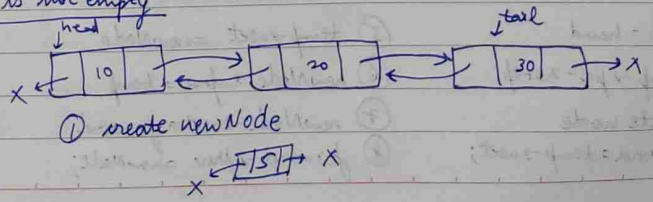
Insertion → O(n)
- insertAtHead
- insertAtTail
- insertAtPosition

insertAtHead

LL is empty : head → NULL, tail → NULL

① create
② head & tail lagado

LL is not empty



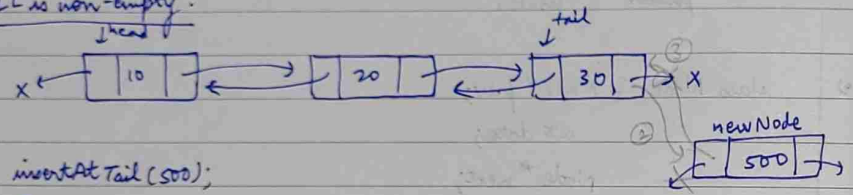
② newNode → next ko head pe lagado
③ ~~head update~~ shift pointer ⇒ head → prev ko newNode pe lagado
④ head update

- print reverse ✓ using tail pointer

insertAtTail

LL is empty ⇒ same as head

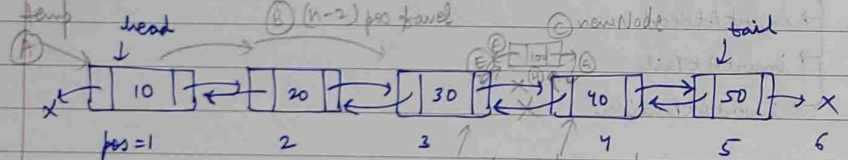
LL is non-empty :



insertAtTail(500);

① create node
② tail → next ko newNode pe lagado
③ newNode → prev ko tail pe lagado
④ tail ko newNode pe lagado

insertAtPosition(value)

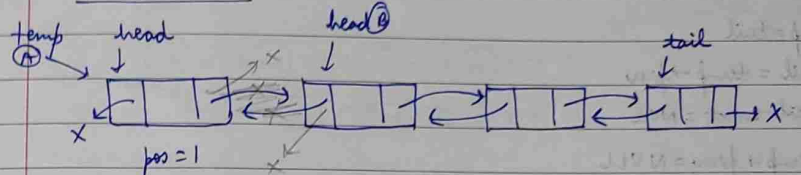


⇒ insertAtPos(1) → insertAtHead();
⇒ insertAtPos(6) → insertAtTail();
⇒ insertAtPos(4)

pos 4 → 2 step
pos 5 → 3 step
pos 8 → 6 step
pos n → (n-2) step

① temp = head
② temp → pos-2 step
③ create node
④ forward = temp → next;
⑤ temp → next = newNode
⑥ newNode → prev = temp
⑦ newNode → next = forward
⑧ forward → prev = newNode;

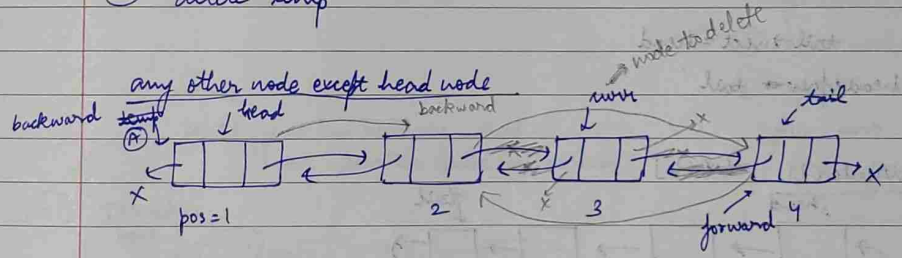
- searching : O(n) → same as singly LL
- deletion → LL is empty
  → single node ✓
  → O(n)
  ↳ head node
  ↳ baaki bache hue



deleteFromPos(1) ⇒ head node

① ~~Node~~ temp = head;
② head = head → next;
③ isolate ⇒ head → prev = NULL
④ temp → next = NULL
⑤ delete temp

any other node except head node

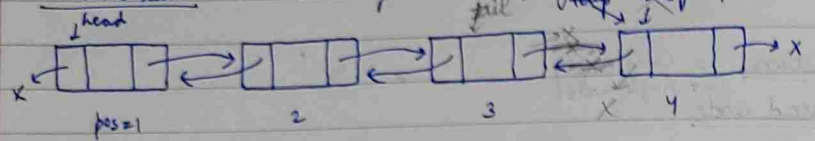


- deleteFromPos(3)

Ⓐ backward ~~temp~~ = head
Ⓑ backward ⇒ pos-2 step
Ⓒ curr = backward → next
Ⓓ forward = curr → next
Ⓔ backward → next = forward
forward → prev = backward
Ⓕ curr → prev = NULL
curr → next = NULL
Ⓖ delete curr.

[3 pos → 1 step
5 pos → 3 step] } (pos-2) steps

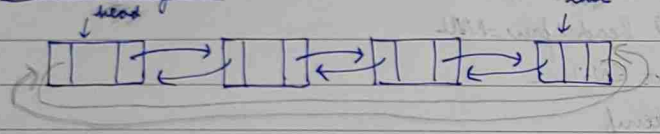
To delete tail (iska code alag se likhna padega) ⇒ if (position = length) --



deleteFromPos(4) ⇒

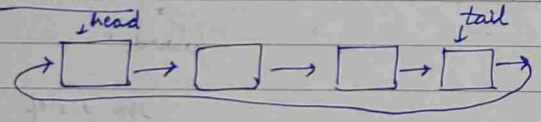
(A) temp = tail
(B) tail = temp → prev
(C) tail → next = NULL
(D) temp → prev = NULL
(E) delete temp

circular doubly LL [CDLL]



tail → next = head
head → prev = tail

circular LL [CLL]

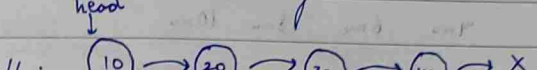


tail → next = head

Linked List Class-3 (LIVE) 18 July 2024

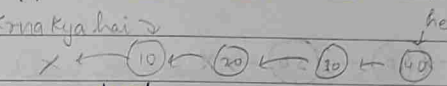
leetcode 206.

Q. Reverse a LL
↳ iterative way
↳ recursive way

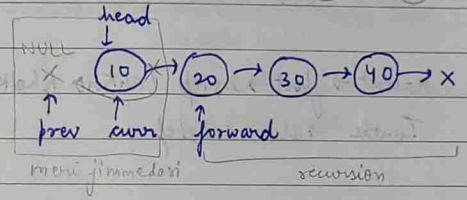


(ab se tail nhi likhenge)

Kya Kya hai ?



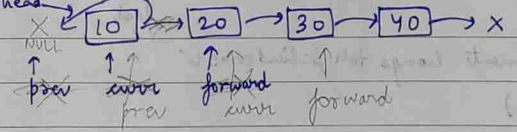
(M2)



(A) prev, curr, forward set
(B) prev = NULL
curr = head
forward = curr → next
(1 case) ⇒ (C) curr → next = prev
(D) updation
prev = curr,
curr = forward
or forward ko recursive call me update ho jayega

TC: same
par SC jyada hai recursive method ki
toh iterative better method hai

(M1) Iterative



(A) prev, curr ✓
(B) forward ✓
curr → next = prev ← while (curr != NULL)
prev = curr
curr = forward

leetcode 876.

Q. find the middle node of LL.

Tortoise Approach

	after 1 sec	2 sec	3 sec	4 sec	5 sec
1 m/s →	1m	2m	3m	4m	5m
2 m/s →	2m	4m	6m	8m	10m

Kadua (jeet gya) — Kharghos
0 — 5m — 10m

head
↓
10 → 20 → 30 → 40 → 50 → X

slow, fast at 10; slow → 20 → 30; fast → 30 → 50 (fast rukjao)

NULL hai ya nhi?

fast = ~~fast → next~~ → next
NULL check kro

rabbit → fast → 2 step [aankhe kholke]
Turtle → slow → 1 step

TC: O(n)

Q. Palindrome

head → 1 → 2 → 3 → 3 (slow) → 2 → 1 → X (fast)
prev at 3

(M-1) dly
reverse LL ✓ same element honge toh palindrome ✓
TC: O(n)
SC: O(n)

(M-2) Turtle approach

(A) prev → next = NULL
(find middle node & break)

head → ① → ② → ③ → X
slow → ③ → ② → ① → X

(B) reverse

① → ② → ③ → X
↑ newHead

(C) compare, equal → T; else → F

odd case (ho jayega ye bhi bina extra case lagaye)

head → 1 → 2 → 3 (slow) → 2 → 1 → X (fast)

1 → 2 → X

3 → 2 → 1 → X

(RTE)

Single node case or empty LL case

NULL ← head 1 → X
↑ middle

TC: O(n)

c) Flow for Problem solving in LL
↳ code
↳ null pointer exception (NULL → next ka mat karna kahi pe)
↳ edge case

(LIVE)

Linked list class-4

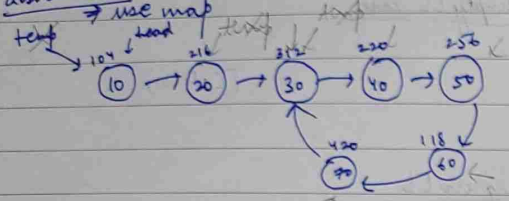
19 July 2024

Leetcode 141.

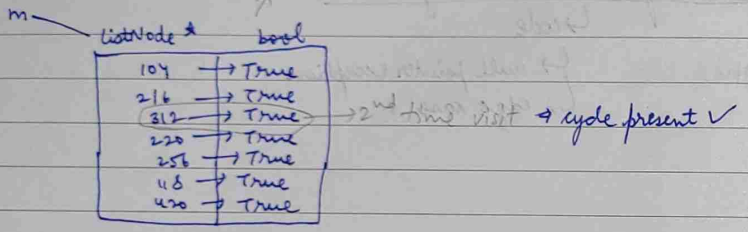
Q. Check cycle in LL (VVV...imp). → Bhot imp

TC: O(n+1) ⇒ O(n) , SC: O(n)

(M-1) desi tarecka ⇒ use map

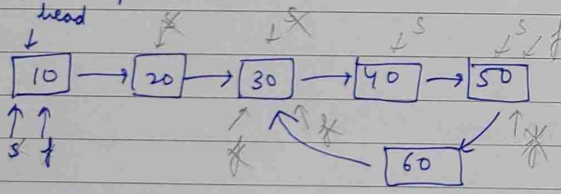


value jo bhi hai usse ghuma fasak nhi padhta



2nd time visit ⇒ cycle present ✓

(M-2) optimise space ⇒ small/fast

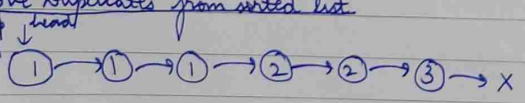


if cycle present ⇒ f/s ⇒ pointer same location pe ajayenge

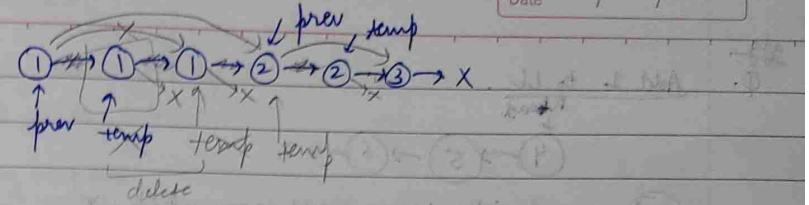
TC: O(n)
SC: O(1)

Leetcode 83.

Q. Remove Duplicates from sorted list

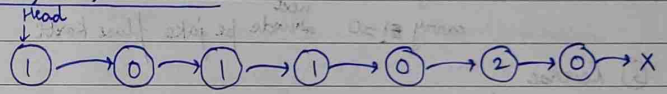


case 1: LL is empty ⇒ return head
case 2: single node ⇒ return head
case 3: > 1 node



(A) prev → next = temp → next
(B) temp → next = NULL
(C) delete temp
(D) if temp → data != prev → data
prev, temp → 1 step move
(E) temp = NULL → stop

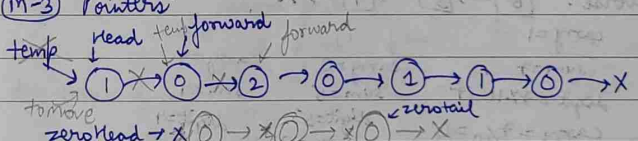
Q. Sort 0's, 1's, 2's in LL

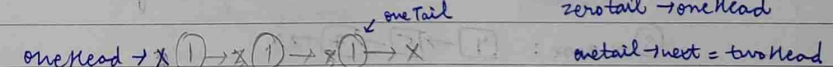


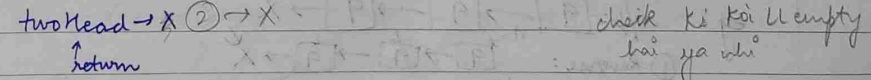
(M-1) → counting ⇒ New LL ⇒ TC: O(n), SC: O(n) — dry

(M-2) counting ⇒ overwrite data

(M-3) Pointers



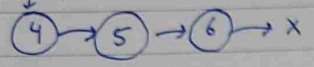




TC: insert ⇒ O(1) ⇒ total ⇒ O(n)
SC: O(1)

~~gfg~~.

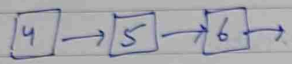
Q. Add 1 to LL.



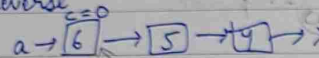
M-1 convert into integer → +1 → convert LL ✗ → dig

M-2 convert into string → +1 → " ✗ → dig

M-3 reverse karke → +1 → reverse ✓



① Reverse



② b → 1

7 , carry = 0

carry != 0 next node pe jake plus karte

③ Reverse

dry run: 4 → 5 → 6 → X temp → rukh jao

① reverse: 6 → 5 → 4 → X

carry = 1

② sum = 7, digit = 7%10 = 7, carry = 7/10 = 0

carry = 0, sum = 5, digit = 5, carry = 0

carry = 0, sum = 4, digit = 4, carry = 0

③ reverse: 4 → 5 → 7 → X

eg: i/p: 9 → 9 → 9 → X

① Reverse: 9 → 9 → 9 → X

carry = 1, sum = 10, digit = 10%10 = 0, carry = 10/10 = 1

carry = 1, sum = 10, digit = 10%10, carry = 10/10 = 1

carry = 1, sum = 10, digit = 10%10, carry = 10/10 = 1 ← special case

0 → 0 → 0 → 1 → X

③ Reverse: 1 → 0 → 0 → 0 → X

(Imp) HW → Add 2 nos. represented by LL.

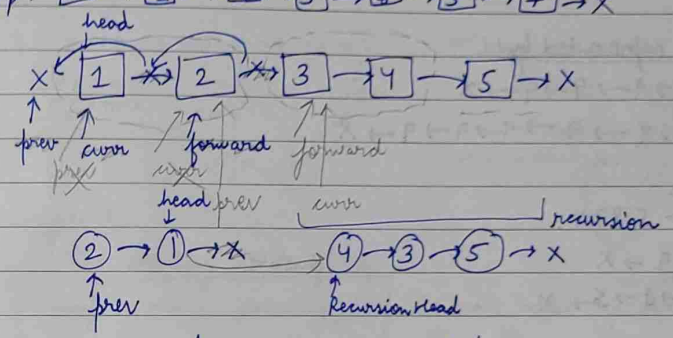
(maga) → Detect & delete loop

leetcode 25 (skip heavy & imp ques)

Q. reverse in groups of k (microsoft me aya tha ye)

i/p: 1 → 2 → 3 → 4 → 5 → 6 → 7 → X K = 2

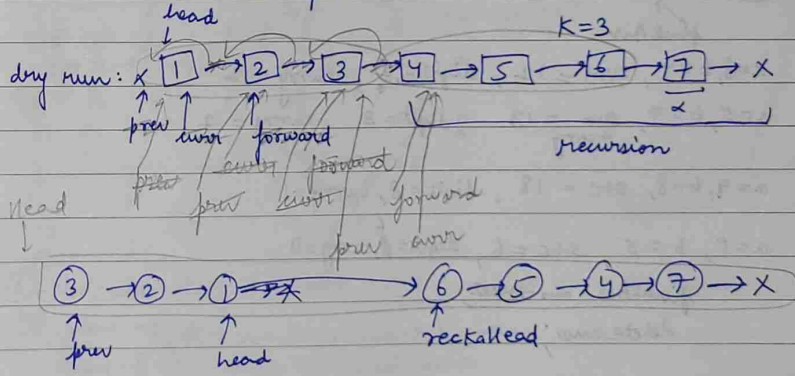
o/p: 2 → 1 → 4 → 3 → 6 → 5 → 7 → X



head → next = recursion Head

return prev

dry run: 1 → 2 → 3 → 4 → 5 → 6 → 7 → X K = 3



head → next = recta Head

return prev

TC: O(n)

HW → quicksort, Mergesort } which algo is preffered in → LL → array

Linked List Mega Class (LIVE)

28 July 2024

① Add 1 to LL (Head RE)

head → 5 → 6 → 7 → 8 → 9 → X

① head RECURSION
② add
③ carry update

leetcode2

② Add 2 nos. represented by LL

l1 → 9 → 9 → 9 → 9 → X

l2 → 9 → 9 → 9 → 9 → 9 → 9 → X

(M-1) iterative

Q: ⇒ 958 + 5874

l1 → 8 → 5 → 9 → X

l2 → 4 → 7 → 8 → 5 → X

ans = -1 → 8 → 2 → 3 → 8 → 6 → X

final ans

a=8, b=4, sum = 12, digit = 2, carry = 1

a=5, b=7, sum = 13 (+carry), digit = 3, carry = 1

a=9, b=8, s+c = 18, digit = 8, carry = 1

a=0, b=5, s+c = 6, digit = 6, carry = 0

finalAns = ans → next;
delete ans; → to prevent memory leak

(M-2) recursive way

l1 → 5 → 6 → 7 → X

l2 → 7 → 5 → 6 → 5 → X

recursive call list

sum = 12, digit = 2, carry = 1

(Imp) Loop detection — slow/fast ptr approach (methods)

① middle finding → done already
② check loop → T/F → done already in class
③ find the starting point of loop
④ Remove the cycle.

② check loop

1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9
14 ← 13 ← 12 ← 11 ← 10

M-1 use map
→ already visited entry milgyi
→ loop detected.
M-2 → slow fast

Leetcode 142 LL cycle-2

③ Find the starting point of loop

(M-1) map → while detecting loop the node which is already visited & we are visiting that again is the starting point

(M-2) slow-fast ptr

S, F → loop detected

1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9 → 10
14 ← 13 ← 12 ← 11

S, F

Find
① Node where loop is detected (S → 1x, fast → 2x speed)
② slow → head
Move F & S with 1x speed now.

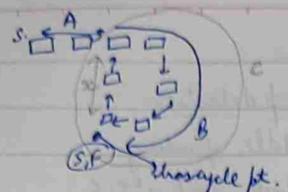
Algo: ① find cycle by checkCycle fn (leetcode 141 ✓) & return where S == F.
② Put slow to head & move S & F with 1x speed and return where they meet

(interviewer proof nahi aayega)

Proof.

① hascycle ⇒

distance travelled by fast ptr = 2 (" " " slow ptr)



⇒ $A + K_1 C + B = 2(A + K_2 C + B)$

$A + K_1 C + B = 2A + 2K_2 C + 2B$ (constant)

$(K_1 - K_2) C = A + B$

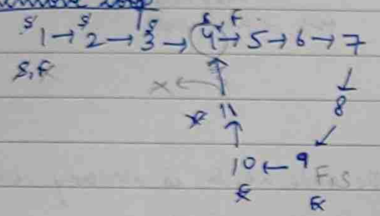
$\boxed{KC = A + B}$

by slow ptr; $K_1 > K_2$; no. of cycle by fast ptr

⇒ displacement wise $A + B = C$ & $B + x = C$

⇒ $A + B = B + x$ → $\boxed{A = x}$

④ Remove loop



Prev = X ~~9~~ ~~10~~ 11

prev → next = NULL

(gfg) → D14

leetcode 725

⑤ Split linked list in Parts

① LL, K → given

↳ K → different parts me divide

i/p ⇒ 1 → 2 → 3 → 4 → X K = 2

o/p ⇒ 1 → 2 → X , 3 → 4 → X

K_parts ⇒ K_1 K_2 K_3

$K_1 = K_2$, $|len(K_1) - len(K_2)| \le 1$

eg : 1 → 2 → 3 → 4 → X K = 10

[1], [2], [3], [4], [ ], [ ], [ ], [ ], [ ], [ ]

eg : 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9 → 10 → X K = 3

ideal case ⇒ all k parts have equal no. of elements

part size = N/K = 10/3 = 3 extra nodes

N % K = 10 % 3 = 1

[3+1] [3] [3]

↑ currentPartSize

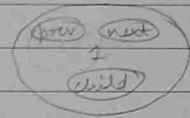
vector <ans> [NULL | 0 | 0] ans[i] = it;

1 → 2 → 3 → 4 → 5 → 6 ----

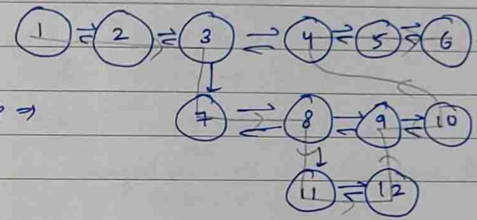
4 parts → loop 3 bar chalega

leetcode 430

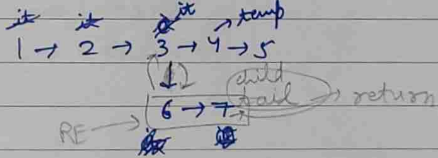
⑥ Flatten a Multilevel Doubly linked list



i/p ⇒



o/p ⇒ 1 → 2 → 3 → 7 → 8 → 11 → 12 → 9 → 10 → 4 → 5 → 6



1 → 2 → 3 → 6 → 7 → 4

```
temp = it → next;
it → next = it → child
it → next → prev = it;
childtail → next = temp;
if temp ≠ NULL
    temp → prev = childTail;
it → child = 0;
```

(⇒ 1 → 2 → 3 → X with child 6 → 7)

→ flatten Kardo (koi child nhi hona chiye)

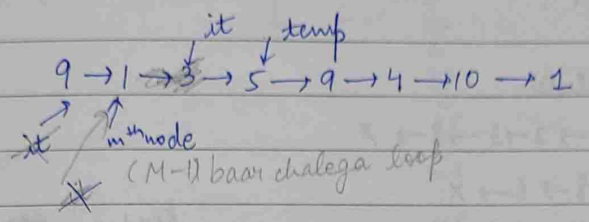
Week-10 Assignments

(gfg)

① Delete N nodes After M nodes

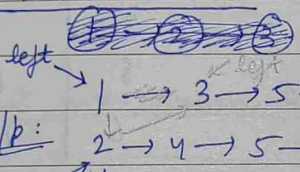
eg: 9 → 1 → 3 → 5 → 9 → 4 → 10 → 1 → X : i/p [M=2, N=1]

o/p: 9 → 1 → 5 → 9 → 10 → 1 → X



Leetcode 21.

② Merge Two Sorted Lists

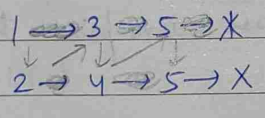


(move pointers) (not change data)

merged ~~ans~~ = -1 (mptr) ✓

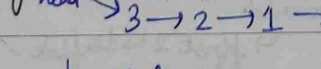
(Compare left and right values)

```
if (left->val <= right->val) {
    mptr->next = left;
    mptr = left;
    left = left->next }
else { mptr->next = right;
    mptr = right;
    right = right->next
}
```



③ Print Kth Node from the end

Hackerrank: get node value



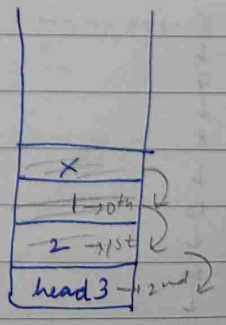
pos ⇒ 2

0 indexed

M-1 recursion:

```
if (head == null) { return; }
```

↳ Base case



M-2 : iterative (diy)

head → 1 → 2 → 3 → 4 → 5 → X
(5th, 4th, 3rd, 2nd, 1st)

Pos(K) = 3

length = 5

ans = len − K(Pos) → 5 − 3 = 2

move from head now (2+1) steps

(gfg)

④ Intersection of Two linked lists

L1 → 9 → 6 → 4 → 2 → 3 → 8 → X

L2 → 1 → 2 → 8 → 6 → X

order should be acc. to L1.

map <int, int>

① hash L2 items

② iterate L1 & eliminate items not in L2.

③ get the final intersection list.

ans → ④ → ② → ⑧ → X

TC: O(m+n)

Key L2	~~ooo~~ no. of occ. (store occ. if there are duplicate elements in LL)
1	1
2	1
8	1
6	1

(gfg)

⑤ Flatten Linked List

next → 5 → 10 → 19 → 28 → X (next)

Bottom ↓

7	20	22	35
↓	↓	↓	↓
8	X	50	40
↓		↓	↓
30		X	45
↓			↓
X			X

use Bottom pointer to merge.

→ 5 → 19 - - -

Bottom ↓

7 ↓ 8 ↓ 10 ↓ 20 ↓ 30 ↓ X

Algorithm :

↳ main LL

↳ first 2 nodes pick ✓

root , root → next

↳ merge with bottom

① 5 → 7 → 8 → 30 → X (Bottom)

② 10 → 20 → NULL

ans = NULL;

ans → 5

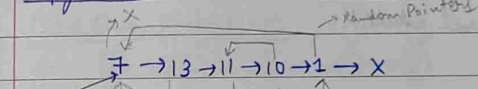
ans → Bottom = (--) (rec. call)

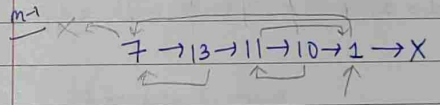
merge from last → 5 → 10 → 19

19 ↓ 22 ↓ 28 ↓ 35 ↓ 40 ↓ 45 ↓ 50 ↓ X

Leetcode 138.

⑥ Copy List with Random Pointer





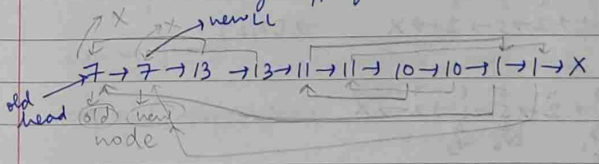
Map	
oldPtr	newPtr
7	7
13	13
11	11
10	10
1	1

TC: O(n)

SC: O(n)

M-2 SC: O(n) → O(1)

without using mapping

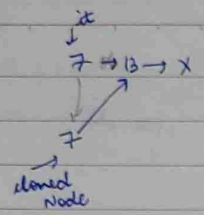


step-1 : clone A → A'

old, new node

step-2 : assign random links on A' with help of A.

③ detach A' from A



leetcode 61.

⑦ Rotate List

head → 1 → 2 → 3 → 4 → 5

① 5 → 1 → 2 → 3 → 4 K=2

② 4 → 5 → 1 → 2 → 3 ∞

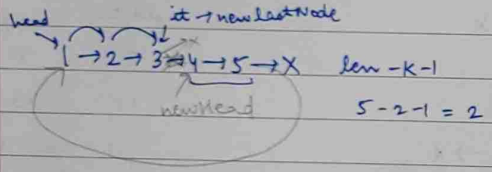
k = listlength → no rotation needed

K % len == 0

K % len → 0 1 2 3 4

(len: 5)

K % len ⇒ actual rotate K



len - K - 1

5 - 2 - 1 = 2

leetcode 2058

⑧ Find min / max number of nodes between critical points (cp)

eg: 3 → 1 → X no critical points → [-1, -1]

eg: 3 → 1 → 5 → X → [-1, -1]

cp ✓ min maxDist

eg: 5 → 3 → 1 → 2 → 5 → 1 → X → [2, 2]

cp1 cp2

eg: 5 → 3 → 1 → 2 → 5 → 1 → 2 → X

p c next cp1 cp2 cp3

max ⇒ lastcp - firstcp

min ⇒ 2 1

leetcode 2181

⑨ Merge Nodes in Between Zeros

i/p: 0 → 3 → 1 → 0 → 4 → 5 → 2 → 0 → X

head 4 11 lastNode

o/p: 4 → 11 → X

slow fast

0 → 3 → 1 → 0 → 4 → 5 → 2 → 0 → X

4

lastNode slow fast fast fast

sum = 0 + 3 + 1 = 4

(fast ko zero mil gya) → slow ko modify karo → slow++

sum = 4 + 5 + 2 = 11

lastNode → next = NULL ⇒ 4 → 11 → NULL

baaki ki LL delete (to avoid memory leak)

TC: O(n)

leetcode 328

⑩ Odd Even linked list

i/p → 1 → 2 → 3 → 4 → 5

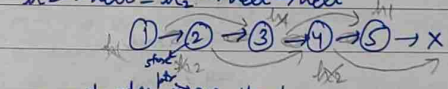
h1 h2

o/p → 1 → 3 → 5 → 2 → 4

h1 → next = h2 → next h1 = h1 → next

h2 → next = h2 → next → next h2 = h2 → next

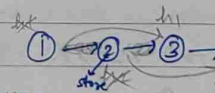
h1 → next = evenHead



odd no. of nodes ⇒ evenHead

stop ⇒ when h2 is NULL

even no. of nodes



stop ⇒ h2 → next == NULL

evenHead

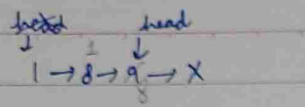
leetcode 2816

⑪ Double a Number Represented as a linked list

i/p: 1 → 8 → 9 → X

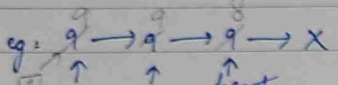
o/p: 3 → 7 → 8 → X

189 × 2 = 378



① use Recursion to access from R→L

```
int prod = head->val X 2 + carry
head->val = prod %.10;
carry = prod /10;
```



① prod = 9 X 2 + 0 = 18
head→val = 18 %.10 = 8
c = 18 /10 = 1

② prod = 9 X 2 + 1 = 19
h→val = 19 %.10 = 9
c = 19/10 = 1

③ prod = 9 X 2 + 1 = 19
val → 19 %.10 = 9
c = 19/10 = 1

```
if (c!=0) {
    insertAtHead(carry); }
```

Interview perspective

Leetcode 1171

(12) Remove Zero Sum Consecutive Nodes from LL

eg: 1 2 -3 3 1
array 0 1 2 3 4 5 6
eg: -9 | 2 | 3 | -2 | -3 | 9 | 1
prefix sum -9 | -7 | -4 | -6 | -9
0 ZERO

⇒ -9 | 9 | 1

p.sum ⇒ -9 | 0 ⇒ 1 TRUE

map
Key value
-9 → True
-7 → True
-4 → True
-6 → True

LL: -9 → 2 → 3 → -2 → -3 → 9 → 1 → X
oldNode ↑
csum ⇒ -9 -7 -4 -6 -9
1 it

map
int, ListNode*
-9, -9
-7, -7
-4, -4
-6, -6

① if csum is in map : sanitiseMap(oldNode→next, csum)
oldnode→next = it→next
-7 -4 -6 -9
csum2 → delete entries
[csum2 == csum break; }

② if csum == 0
head = it→next;
mp.clear();

-9 → 9 → 1
it

Algo: csum = 0
while (it) {
    csum = it→val;
    if (csum == 0) {
        head = it→next;
        mp.clear(); }
    else if (mp.find(csum) != mp.end()) {
        sanitize map()
        mp[csum]→next = it→next; }
    else { mp[csum] = it;
    it++;
}

TC: O(n)

Leetcode 1721

(13) Swapping Nodes in a LL

1 → 2 → 3 → 4 → 5 K=2
K=2 K=2

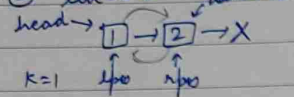
M-1 ⇒ swap(it1→val, it2→val) → simple dry
M-2 changing links
1 → 2 → 3 → 4 → 5 ⇒ 1 → 4 → 3 → 2 → 5

case
C① Null || head→next == 0 return head;
X 1 → X

C② 1 → 2 → 3 → 4 → 5 → X K=3 ⇒ lpos == rpos return head;
rpos
lpos

leftpos, rightpos
K=2 ⇒ lpos = 2 (pos from head)
rpos = len - K + 1 ⇒ 5 - 2 + 1 = 4

C③ len == 2

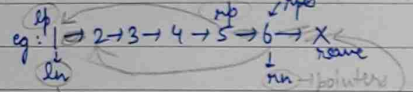


K = 1 or 2
Kuch bhi ho swap toh karna hi hai

```
nextNode → next = head;
head → next = NULL;
head = nextNode;
```

C④ lpos == 1 ⇒ K=1 , K=N

1 → 2 → 3 → X → if (rpos < lpos) { swap ; }

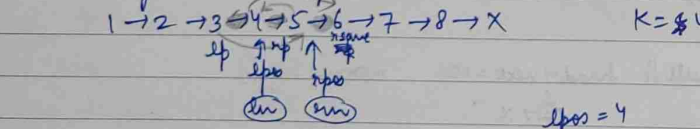


loop → rpos - 2 times

① rn → next = ln → next ;
rp → next = ln ;
ln → next = rsave ;
head = rn ;

6 → 2 → 3 → 4 → 5 → 1 → X

C⑤ no. of nodes btw swap nodes == 0

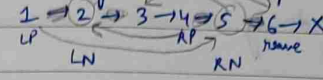


lpos = 4
rpos = 5
rpos - lpos - 1

```
lp → next = rp → next;
rn → next = rp;
rp → next = rsave;
```

1 → 2 → 3 → 5 → 4 → 6 → 7 → 8 → X

Case⑥ No. of nodes Btw swap Nodes >= 1



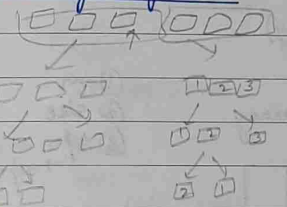
K = 2
lpos = 2
rpos = 5
↳ len - K + 1 = 6 - 2 + 1 = 5

```
lp → next = rn;
rn → next = ln → next;
rp → next = ln;
ln → next = rsave;
```

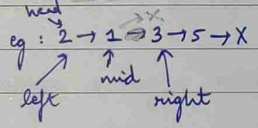
⇒ 1 → 5 → 3 → 4 → 2 → 6 → X
TC: O(n)

leetcode 148.

⑭ (Imp) Sort list using merge sort



① mid
② using mid divide array
③ RE divide
④ merge.



TC: (nlogn)

STACKS

Stacks Class-1 (LIVE)

29 July 2024

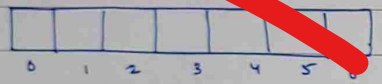
- Data Structure that follow LIFO ordering.

- STL (done already in STL videos)

s.push(10)
s.push(20)
s.push(30)
s.pop()
top() → 20

- Implementation (implement using array)

- i/p → size
  - create dynamic array



0 1 2 3 4 5

① arr
② size
③ top → index ko return
↳ top element

top = -1

push → top++
arr[top] = val

pop → arr[top] = -1
top--

size = 10

push → s.push(11)
s.push(50)
↳ if (top == size-1) {Stack Overflow}

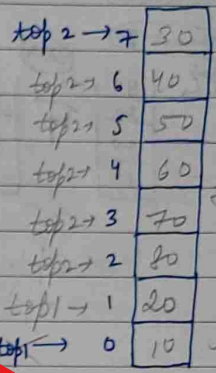
pop → s.pop()
↳ if (top == -1) {stack underflow}

getSize → return top+1

getTop → return arr[top]
↳ if (top == -1) → stack empty

isEmpty → if top == -1 → return true.
else → false.

Q. implement 2 stacks in an array



top2 → 7 30
top2 → 6 40
top2 → 5 50
top2 → 4 60
top2 → 3 70
top2 → 2 80
top1 → 1 20
top1 → 0 10

part-1
part-2

M-① divide array in 2 parts

M-② 2 top indexes
↳ top1, top2
⇒ jab top1 & top2 mil jaye ⇒ stack full
top1 == top2
⇒ top2 - top1 == 1

Stack 1 → push
top1++
arr[top1] = val

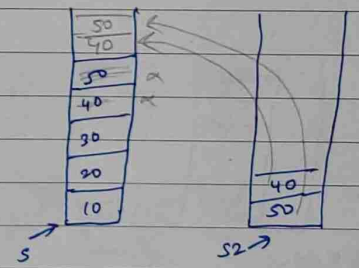
Stack 2 → push
top2--
arr[top2] = val

Q. Reverse ... top and pop the retrieved char.

Q. Print Middle. code in stack class-2

size → 5
size → 2

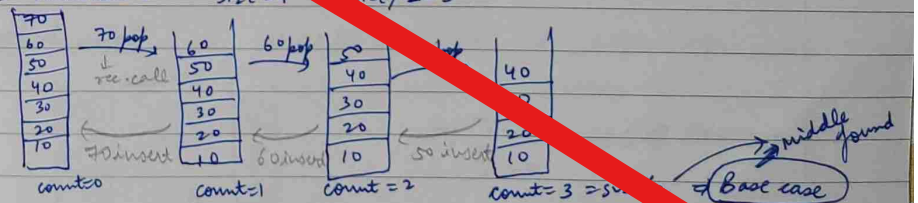
M-① use 2 stack



50 40 30 20 10 → S
40 50 → S2

M-② recursion

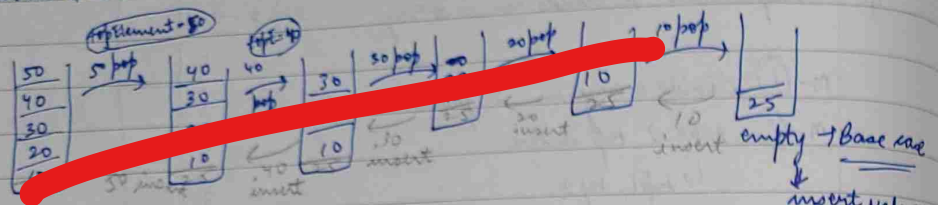
size = 7 size/2 = 3



70 60 50 40 30 20 10
70 pop ↓ rec. call
70 insert
60 pop
60 insert
50 pop
50 insert
count = 0
count = 1
count = 2
count = 3 = s...
middle found
Base case

SC: O(n)
TC: O(n)

eg: insertAtBottom(value) value → 25



(Live)

Stacks Class-2

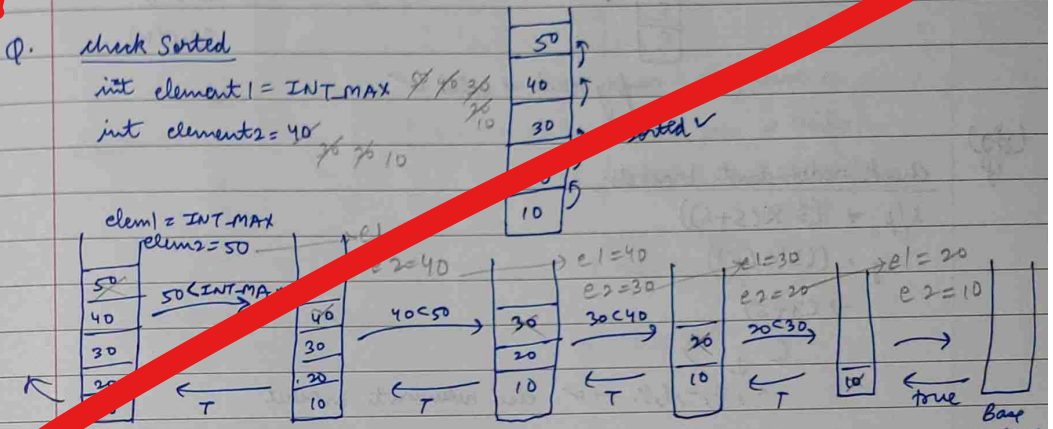
31 July 2024

1. Send Stack by reference
2. print middle. (theory done in class-1)

↳ code github

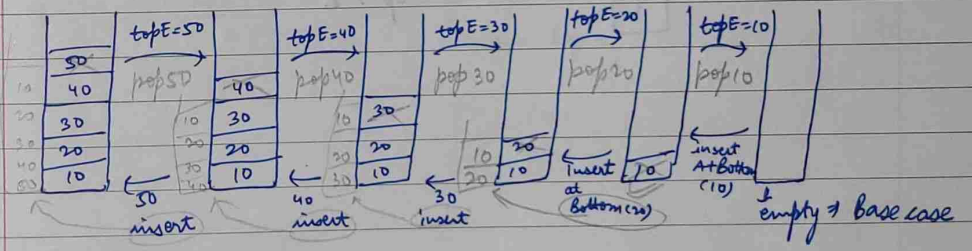
Q. check Sorted

int element1 = INT_MAX
int element2 = 40



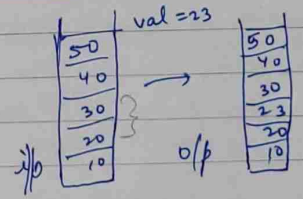
TC: O(n)
SC: O(n)

Q. reverse a stack. → insertAtBottom



Q. sorted insert

element jo 23 se just chota hoga waha insert kardena

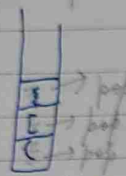


s.top() < val → Base case
↓ insert

also make sure that stack is not empty

(Very imp)
Q. Valid Parenthesis → repeated many times

eg: ([{}])



empty stack ⇒ valid ✓

TC: O(n)
SC: O(n)

(q/a)
Q. check redundant brackets

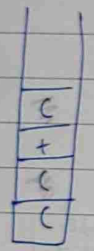
i/p ⇒ (3 x (5+6))

eg: ((3)+(5)) → 2 pair

⇒ (3+5)

( )

+, x, -, /, % → ✓ else redundant bracket



((3)+(5))
ignore ignore
operator found in btw
⇒ (+)

pairs → +1+1 ⇒ 2

op → 1

Stack class-3 (LIVE)

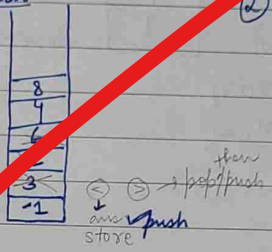
2 Aug 2024

Q. next smaller Element (brute)

↳ i/p ⇒ ans: {8, 4, 6, 2, 3}
↓ ↓ ↓ ↓ ↓
4 2 2 -1 -1

o/p ⇒ {4, 2, 2, -1, -1}

Approach



(2) O(n) → single pass (double pass ⇒ O(2n))

jab tak element < s.top()
↳ pop

else > s.top()
↳ stop

i=(n-1) → $ 0

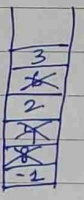
eg: 7 3 5 1 2 -1
↓ ↓ ↓ ↓ ↓
3 1 1 -1 -1

① Brute force: O(n²) → check for 7 then 3 then 5 and so on...

Q. prev smaller element

eg: -1 | 8 | 4 | 2 | 6 | 3
0 1 2 3 4
-1 -1 -1 2 2

⇒ go left to right. i=0 → n-1
→ same logic

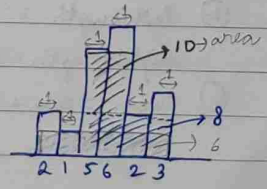


TC: O(n)
SC: O(n)

Leetcode 84.
Q. Largest rectangle in Histogram

i/p ⇒ {2, 1, 5, 6, 2, 3}
nextArr → {1, -1, 4, 4, -1, -1} ← index
prevArr → {-1, -1, 1, 2, 1, 4} ← index

TC → linear



(imp)
Q. Min stack

Stacks Mega Class (LIVE)

11 Aug 2024

~~header file (half hour–45 mins)~~

leetcode 155

① **Min stack**

eg: 5, 4, 1, 5, 2, 4, 2, 0, 6

→ use pair

↳ store both {value and min till now}

↓ top se → O(1)

5 → push → top min till now?

↓

1

pop
6,0
0,0
2,1
4,1
2,1
5,1
1,1
4,4
5,5
push

min: 5, 4, 1, 1

leetcode 907

② **Sum of subarray minimums**

eg: arr = [3, 1, 2, 4]
(indices 0 1 2 3)

(M1) Brute force → find all subarrays & get min of all → TC: $O(n^2)$

(M2) How many times a particular element is included in my ans?

eg: 0 | 2 | 3 | 4 | 1 | 4 | 3 | 2 | 0 | 7

prev smaller of 1 ←—— 1 ——→ next smaller of 1

4 × 4 = 16 subarray in which 1 is minimum

Algo: ① for every index → monotonic stack
↳ prev smaller index (PSI)
↳ next " " (NSI)

② leftlength = i − PSI
rightlength = NSI − i

③ sum = sum + (left × right)

TC: O(n)

leetcode 2104

③ **Sum of subarray ranges**

eg: 4 | −2 | −3 | 4 | 1
(indices 0 1 2 3 4)

→ max − min of all subarray

① → Total Max Sum → Σ sub → Max

~~Total Max Sum → Σ sub → max~~
~~" Min " → Σ sub~~

② Total Min Sum → Σ sub → Min

③ Total Max − total min Sum → return ✓

→ How to find ① & ② in O(n)?
↳ LC 907

eg: 100 | 2 | 3 | 4 | 10 | 4 | 3 | 2 | 100
(indices 0 1 2 3 4 5 6 7 8)

prev greater Element ← 4 × 4 = 16 → same ✓ → next greater Element

leetcode 224

④ **Basic calculator**

eg: "1 + (2 − (3 + 4)) + 5"

stack → keep track of the result & sign.

(→ maintain

) → pop and combine ans to result

(Brackets sahi se diye honge → assume)

stack
− {−1}
2
+ {1}
1

−7 + 2 = −5

−5 + 1 = −4

ans → −4

3 + (+ × 4)
→ 7

result = ~~0 1 0 2 0 3 7~~ −5

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

sign = ~~1 −1~~ +1
→ int

if (digit) { num = num + (c − '0') }

else if (c == '+') { result = result + sign × num;
sign = 1; num = 0;
}

else if (c == '−') { result = result + sign × num;
sign = −1; num = 0;
}

if '(' → result → Push
sign → Push → (+1), (−1) ; result = 0; sign = 1

if ')' → result = sign × num;

// pop sign
↳ result *= sign;

result += st.top();
num = 0;

→ result = result + numXsign

Week-11 Assignment

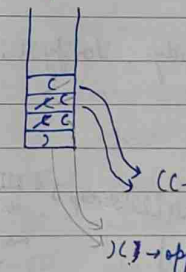
① **Minimum Bracket Reversal**

S =) (( )) (((
0 1 2 3 4 5 6 7
↳ ((( )) ( )))
0 1 2 3 4 5 6 7

① ((→ same characters
↳ () → ① count of reversal

②)(→ diff characters
↳ () → ② → count

① → valid parentheses logic will be used → remove valid pairs
② if stack is non-empty ⇒ try to find reversal count
⇒ if odd size string ⇒ return -1.

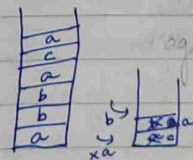


if (ch == '(') push
else {
if (s.empty() == 0 && st.top() == '(') st.pop();
else push('ch');
}

Leetcode 1047.

② **Remove all adjacent duplicates in Strings** (already done in strings class-2)

eg: abbaca
↳ aaca → ca → ans ✓



(skip)

③ **Celebrity Problem**

	0	1	2
0	0	1	0
1	0	0	0
2	0	1	0

N = 3 → 3 persons
M[0][1] = ① → 0th person knows 1st person
i th " j th "
0 → 1 → celebrity
↑ 2

(M1) BruteForce

(a) celebrity ⇒ row all 0s → celebrity doesn't know anyone

& (b) celebrity ⇒ columns all 1s → everyone knows celebrity
↳ except diagonal

algo → check rows & cols of each person [0, n)

TC: $O(n^2)$

(M2) ① Put all persons in stack.

② while (st.size != 1)

③ A → s.top();
B → s.top(); → M[A][B]
if (A knows B) { A is not celebrity → discard A, B → push B again }
else { B is not celebrity → discard, push A back }

③ that single person in stack might be a celebrity. Verify it.

TC: $O(n)$

leetcode 1019

④ Next Greater Element in LL

2→1→5→X

ans → [5|5|0] (0 1 2)

eg: 2→1→7→4→3→5→X

ll → [2|1|7|4|3|5] (0 1 2 3 4 5)

ans → [7|7|0|5|5|0]

Stack (index store): 2, 3, 4

while (ll[st.top] < ll[i]) {
st.pop()
ans[i] = ll[i];
}

if (ll[i] > ll[st.top]) {
pop till ll[st.top] < ll[i] ... ll[st.top]'s greater elements will be ll[i];
} else { st.push[i]; }

7 > 1?

(Imp) ⑤ N stacks in an array

N → no. of stacks

S → size of array

S/N, S/N, S/N — array of size S

(M1) ① 1 part size ⇒ S/N length

(M1) → not spaced optimised

eg: N=3, S=12

S/N = 12/3 = 4

S1: [10|11|12|13] S2: [15|16|18|] S3: [19|20|31|] (0 … 11)

push(14) → s1 stack is overflow ↳ fragmentation

(M2) Two additional arrays.

① top[] → size → (N) → it stores index of top element of ith stack

② next[] → (a) it can point to next element after top element. (size S)
(b) it can point to next free space.

S → size of main array

⇒ initialize → N = no. of stacks → 3 → S_1 S_2 S_3

S → size of array → 6   freespot = 0

array → [| | | | |] (0 1 2 3 4 5)

top → [-1|-1|-1] (0 1 2) → stores index of top element in stack

next → [1|2|3|4|5|-1] (0 1 2 3 4 5) → stores next free spot kaha hai?

⇒ PUSH() → x, m

→ I. find index ⇒ int index = freespot;

→ II. update freespot ⇒ freespot = next[index];

→ III. insert in array ⇒ a[index] = x.

→ IV. update next ⇒ next[index] = top[m-1];

→ V. update top ⇒ top[m-1] = index;

⇒ POP (m) 1 indexed. → opp of push

→ I. int index = top[m-1];

II. top[m-1] = next[index];

III. next[index] = freespot; IV. freespot = index;

V. return a[index];

Dry run: [5|10|6| | |] (0 1 2 3 4 5)   N=3, S=6

top: [-1|-1|-1]

next = [1|2|3|4|5|-1] (0 1 2 3 4 5)

freespot = 0 (fs)

① PUSH (5,1)
index = 0
fs = next[index] = 1;
next[index] = top[m-1];
top[m-1] = index;

② Pop(1) index = 2
top[0] = next[2]
next[2] = fs (3)

leetcode 901

⑥ Online Stock Span → monotonic stack

eg: 7 [2 1 2 2] ← Today → 4 ans
↓ 1st Jan, 2, 3, 4, 5th Jan

eg: 100, 80, 60, 70, 60, 75, 85 ↓ Price 100

100	80	60	70	60	75	85

M-1 vector
① 100 → 1 (span)
② 80 → 1
③ 60 → 1
④ 70 → 2
⑤ 60 → 1
⑥ 75 → 4
⑦ 85 → 6

TC: O(n²)

M-2 stack

Stack: 85,6 → span = 1+4+1 = 6; 75,4 → span = 1+1+2 = 4; 60,1; 70,2; 60,1; 80,1; 100,1

span = 1
while (st.top() → price <= price) { span += st.top() → span;
st.pop() };
st.push(price, span);

TC: O(n)

leetcode 1003

⑦ Check if word is valid after substitutions

Given → (s) valid → t = " "

can transform t → (s) after doing below operations:

① t → "abc" insert
t → t_{left} + "abc" + t_{right}
t_{left} can be empty; t_{right} empty ✓

eg: aabcbc → t = " "
① t ⇒ t_l = " " ; t_r = " " ⇒ t = abc
② t ⇒ t_l = "a" ; t_r = "bc" ⇒ t = a abc bc ✓

M-1 recursion → TC: O(n²)
M-2 stack
eg: abcabcababcc
eg: ababccb
if s.empty() → return true
if s[0] != 'a' → " false.

SC: O(n)
TC: O(n)

leetcode 394

⑧ Decode String

eg: 3[a] 2[bc]
↓
aaabcbc

eg: 3[a2[c]] → 3[acc] → accaccacc
eg: 3[a2[bc]]

→ cb 2 (actual string; numeric digit)

① cbcb ② cbcb + a + [ignore + 3 → times ⇒ "cbcbacbcbacbcba" reverse

(stack: c, b, [, 2, [, a, [, 3 — pushed, pop)

leetcode 853

⑨ Car Fleet (monotonic Stack)

eg: target = 12 ; pos = 10|8|0|5|3 ; speed = 2|4|1|1|3

(number line 0 … 12 → target destination)

(4,2) → ① fleet ; (3,1) → ② ; (1) → ③ ⇒ ans: 3
speeds

sortPos ⇒ 0|3|5|8|10   sortSpeed ⇒ 1|3|1|4|2

Time = dist / speed = (target − Pos[i]) / speed[i]

aloneTime ⇒ 12|3|7|1|1 (agar sirf wahi car ho raste pe)

⇒ i=0, pos=0, 12hr
i=1, pos=3, 3hr
i=2, pos=5, 7hr
i=3, pos=8, 1hr
i=4, pos=10, 1hr

while (time ≥ st.top) { st.pop(); }
st.push(time);

(stack: 1 → ① fleet; 1; 7 → ②; 3; 12 → ③)

leetcode 1776

⑩ Car Fleet-II

Pos → 3|5|6|9 ; speed → 4|4|3|1 ; collision Time → 2|1|1.5|-1

is there any car ahead of me with which I can collide?

2nd car ⇒ 9−6 / 3−1 = 3/2 = 1.5sec ; 1st car ⇒ 6−5/4−3 = 1sec

0th car ⇒ 0th → 1st ✗ , 0th → 2nd ✓ ⇒ 6−3/4−3 = 3sec ✗
0th → 3rd ✓ ⇒ 9−3/4−1 = 2sec ✓ ans

if (operand of me is / order) {
pop }
else { ans + pop }
if (current time <= st. top val) { current time = ... }
break;

TC: O(n)

leetcode 71

(11) Simplify Path

eg : /a/./b/../../c/

① /a/./b/ home —/a→ a —/.→ a —/b→ b

② /../.. home —/a→ a —/.→ b

③ /c home —/c→ c
→ simplified

TC: O(n)
SC: O(n)

* pushing & popping based on directory name
① /.. → comes out from curr dir ② /name → going to dir

leetcode 85

(12) maximal Rectangle (max rectangle in binary matrix)

eg : 1 0 1 0 0 area → 3 → ... (6) ... 4
1 0 1 1 1 max
1 1 1 1 1
1 0 0 1 0

SC : O(m×n)

Ist row → 1 0 1 0 0 longest area → ① ✓

IInd row → 2 0 2 1 1 longest area → ③ ✓

IIIrd row → 3 1 3 2 2 → ⑥ ✓

IVth row → 4 0 0 3 0 → ④ ✓

leetcode 739.

(13) Daily Temperature

eg : 73 | 74 | 75 | 71 | 69 | 72 | 76 | 73

ans 1 | 2 | 6 | 5 | 5 | 6 | 0 | 0
(next warmer day) 1 | 1 | 4 | 2 | 1 | 1 | 0 | 0 ✓

next Greater Element while (st empty & temp[i] > temp[st.top()])
{ ans [st.top()] = i - st.top();
st.pop(); } st.push(i);

leetcode 402.

(14) Remove K digits

eg : "1432219" K = 3
→ 1329, 3221, 1219, ...
smallest

(M1) find all combinations with 3 char removed from string
no. of ways to choose 3 pos out of 7 → $^7C_3$ = 35

(M2) better way ✓

eg : 9 8 7 6 5 4
st.top() > digit → pop()

eg : 1 2 3 4 5 6 7

eg : 10200 K=1 → 0020
→ 200 ans

leetcode 921 (imp)

(15) Minimum Add to make Parentheses valid (Parentheses ka ques aye or aap stack ke baare me na socho, esa ho sakta hai kya?)

① empty ✓ valid

② A B → AB ✓ (()) + () → (())() → valid ✓

③ A → (A)

eg : s → "()))" → valid → ((()))
valid → hata do stack se

Approach : ans = 0;
if (ch == '(') { ans++; st.push(ch) }
else { if (!st.empty) { // match found
st.pop(); ans--; }
else { ans++; }
}

SC : O(n)
TC : O(n)

eg ① (((
eg ② (()
eg ③ ()()))

leetcode 32.

(16) Longest Valid Parentheses TC: O(n)

eg : ( ) → 1 - (-1) = 2
-1 0 1

eg :))) (( )

if (st.empty) →) → invalid bracket (opening bracket nhi tha present)
→ st.push(i);

Graphs:
→ len = i - st.top();
= 4 - 2
= 2

Page No. 46

leetcode 735

(17) Asteroid Collision

eg: 5|10|-5 → 5|10

eg: -8|8 → -8|8

eg: 1|-2|10|-5 → -2|10 ans

- if (Right move) → push ; else collide
- collision possible if st.top → +ve ; asteroid → -ve.
- top -ve ; asteroid +ve ; → push

A → B → A

A B → Null (if A == B)

leetcode 1472

(18) Design Browser History

- → only one tab
- → starts at home page (HP) → clear

back / forward

HP → FB → Ig ; FB → Yt

Browser(BS) → HP(LC)
visit → Google (G)
Visit → FB
Visit → Yt
Back → 1 → at FB now
Back → 1 → G
Fwd → 1 → FB
visit → LinkedIn (LI) → clear fwd history
Fwd → 2 → LI
back → 2 → at G
Back → 7

BS stack: LI, FB, Yt, FB, G, LC

Fwd stack: G, FB, Yt, LI

→ Back()
↳ if(BS.size>1) { Fwd stack.push() }
else Break;

Page No. 47

QUEUES

Queue Class-1 (LIVE) 4 Aug 2024

- Queue : insert at rear end.
  : remove from front.
- follow FIFO ordering.

Types
- normal Queue
- circular Queue (less wastage of memory)
- deque → push back, push front

- STL → circular queue → (normal queue hota hi nahi bas samjhne ke liye)
- Implementation → array or LL.
  ↳ array, size, front, rear.

front, rear: | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | (index 0 1 2 3 4 5 6 7); rear = -1, front = -1

insertion → rear → ① overflow → if (rear == size-1) → cout << "overflow";
② first element insert → arr[rear] = 10 (front & rear dono update honge)
③ normal insertion → update rear ; arr[rear] = val

q.push(10), q.push(20), q.push(100)

removal: front, rear | 10 | 20 | 30 | 4 | 50 | | | | (index 0 1 2 3 4 5 6 7)

pop → ① empty → underflow
② single element present → front = rear = -1
③ normal flow → update front ;
arr[front] = -1 ; → optional

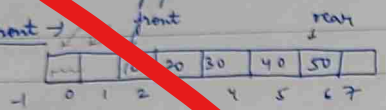
get size → rear - front + 1
Pop/Push → TC: O(1)
↳ same operation → O(1)

Deque implementation

pushBack → same as push in queue

popFront → " " pop " "

* pushFront →



① overflow → if (front == 0) cout << "overflow";

② first Element → if (f == -1 && r == -1) f++, r++, arr[f] = val;

③ ~~normal~~ flow → f--, arr[f] = val;

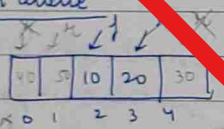
* popBack

① underflow → if (front == -1 && r == -1)

② single element → if (f == r) → f = -1; r = -1;

③ normal → r--;

circular Queue



Push

①overflow → r = front - 1. (after circular rotation)

front = 0 & r = n-1

② single element

③ circular nature → r = n-1 → ghum ja par front = 0 na ho

④ normal

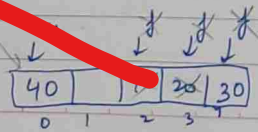
Pop → ①underflow → f == -1 & r == -1

② single element → f == r → f = -1, r = -1

③ circular nature.

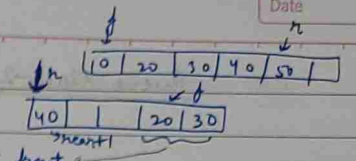
④ normal → f++;

circular nature



getSize: normal → r - f + 1

circular → r < f

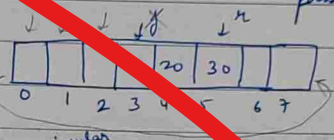


S - f + r + 1

Implement deque + circular nature (De Circular Queue)

popFront → same as CircularQueue ~~pop~~ pop

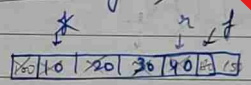
pushBack → " " " " push.



pushFront → circular f == 0 && r != -1 → f = n-1 ; arr[f] = val;

normal → f--, arr[f] = val;

popBack :



normal → arr[r] = -1, r--

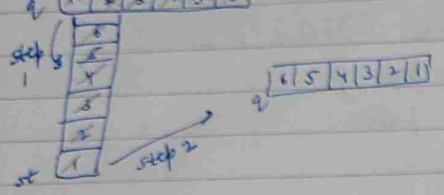
circular → if (rear == 0) arr[0] = -1, r = n-1

Queue Class-2 (LIVE)

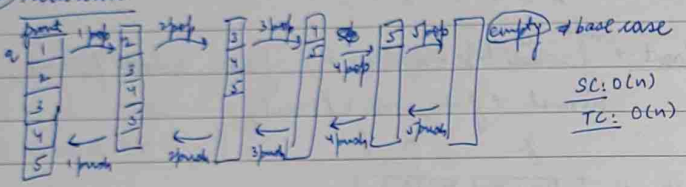
7 Aug 2024

Q. reverse a queue

SC: O(n), TC: O(n) → (M-1) using stack ✓



(M-2) recursion

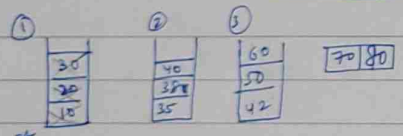


SC: O(n)
TC: O(n)

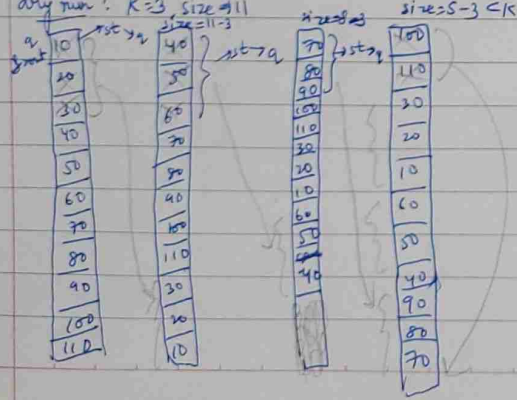
Q. reverse in "K" group

K=3, q → 10 20 30 35 38 40 42 50 60 70 80

o/p → 30 20 10 40 30 35 60 50 42 70 80



dry run: K=3, size = 11



~~TC: O(n)~~

TC: O(n)

Q. Interleave the first half of the queue with second half.

eg: 2 4 3 1 (Bright Knowledge even in bags)

2 4 | 3 1

2 3 4 1

q: 10 20 30 40 50 60 ← second half

↓ step 1 → first half queue

q2: 10 20 30 ← first half

ans → 10 40 20 50 30 60

Q. Sliding window Pattern → find subarray / find substring / window sliding

Q. First negative in every window of size K.

Kuch element remove hoga, Kuch element add hoga

→ first window handled separately ✓

eg: -8 2 3 -6 10 K=2

0 1 2 3 4

-8 0 -6 -6

3 step process → ① process first window

② process remaining window

③ store last window ans.

- ans
- removal
- addition

eg: -10 -20 -30 -40 -50 -60 -70 80 K=4

0 1 2 3 4 5 6 7

ans -10

dq: 0 1 2 3

i-0 i-1 i-2 i-3

4 1 2 3

if (i - q.front() >= K) → pop_front

TC: O(n)
SC: O(K)

current window → (i-K+1) → i

Queue Class-3 (LIVE) 8 aug 2024

leetcode 239.

Q. sliding window maximum

[3 7 5] → q.Keandar < arr[i]
pop

K=3

1	3	1	2	0	5

arr 0 1 2 3 4 5

(left wale ko compare krna bas)
(right wale ko compare Kroge toh woh toh next window me bhi ayega)

(index)
dq | x | 1 | 2 | 3 | 4 |

Ist window → 3 3 3 → 2 ana chiyetha

Remaining windows IInd 1 < 3-3+1 → 1 < 1 → F
1 < 4-3+1 → 1 < 2 → T ✓

→ front ki jgh back se pop krna hai.

TC: O(n)
SC: O(K)

eg: | 1 | -3 | -1 | -3 | 5 | 3 | 6 | 7 | K=3

o/p → {3 3 5 5 6 7}

dq | x | 1 | 2 | 3 | 4 | x | 6 | 7 | |

window: I → 3 ; II → 3 ; III → 5 ; IV → 5 ; V → 6 ; VI → 7

gfg
Q. First non repeating character in a stream

(M-1) → map → O(n²) → TLE
(M-2) stack X queue ✓ → FIFO TC: O(n)

eg: aacdde ans → a##cccc

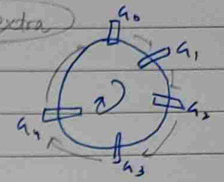
freq

char	int
a	1 2
c	1
d	1
e	1

q | a | a | c | d | d | e |

leetcode 134

Q. Gas Station



ans = 3

G0 G1 G2 G3 G4 → 3l extra → 6l extra

gas → | 1 | 2 | 3 | 4 | 5 |

cost → | 3 | 4 | 5 | 1 | 2 |

deficit 2 2 2 → 3 4

(M-1) Brute force → O(n²)
→ Total 6l deficit

(M-2) deficit = 0 2 4 6
balance = 2 3 6
start = 0 1 2 3

gas | 1 | 2 | 3 | 4 | 5 |
Km or cost | 3 | 4 | 5 | 1 | 2 |
(gas < km deficit = 2) (gas > km balance = 3)

if (balance - deficit >= 0)
→ circle complete ho jayega → return start

else → circle can't be completed → return -1

TC: O(n)