DSA Revision Guide

# Stack

## Next greater element:

* Hame next greater element to the right side of current element find karna hai (next greater hai largest to right nhi)
* Ex: 1 2 3 4 5 6 ans = 2 3 4 5 6 0
* Approach Using: stack in O(n) SC & TC
* Iska exp nahi hai dry run karo smjh jayega kaise kaam karti hai
* Practice tc : **[1,7,5,1,9,2,5,1] ans = [7,9,9,9,0,5,0,0]**
* Working:
  + Prepare an empty stack which will store pair > (arr[i],i)
  + For incoming elements :
  + For i ->(0, N)
    - While ((stack and stack[-1][0]<arr[i])):
      * Pair = Pop the last pair
      * Change the value index of popped pair with arr[i]
        + arr[pair[1]] = arr[i]
    - Append the (arr[i],i) pair
    - And arr[i] = 0
  + Return arr

# Binary Search

## Find single element in sorted array of duplicates :

Logic : har element ka ek pair hoga hence pair ka starting ele even idx pe hoga and finising ele odd idx pe hoga

Ab agar koi ek single element hai array mein toh uske baad ke saare element ke yeh pair – idx relationship kharaab hojayenge kyunki uske wajah se aage wale elements ko ek idx aage shift karna hoga

While(left<=right)

Find mid ->

Check = mid-1 != mid != mid+1

Return mid

if mid is odd

check mid-1 == mid ?

Yes -> search in mid+1 to end

No -> Search in start to mid-1

If mid is even

Check mid+1 == mid ?

Yes -> search in mid+1 to end

No -> search in mid-1 to end

Return None

# Linked Lists :

## Linked List Cycle detection and Loop deletion

Loop detection:

* Concept yeh hai two slow fast pointer rakho ek ko kouble speed se bhagao agar loop hai toh dono kahi meeti karenge aur then we can say there is a loop in LL
* Note: Iska code likha thora tough hai (asaan lage dikhne)

Starting point of a loop

* + Check head is null or it is a single node in LL :
    - Return no loop
  + Fast = head
  + Slow = head
  + Entry = head
  + While(fast.next and fast.next.next):
    - Slow = slow.next
    - Fast = fast.next.next
    - If slow == fast: //loop hai
      * While(entry != slow):
        + Increment entry and slow by one node
      * Return loop found at entry
  + Return no loop
* Ek aur ptr lo “entry” jo head pe hoga
* Pehla step wohi hai slow fast pointer wala jaha meet honge waha pe ruk jao
* Then slow and entry ko eke k karke increment karo … jaha meet honge waha pe loop ka
* Starting point hai
* Proof:
* Text, letter

  Description automatically generated
* Now …. slow has travelled k distance and sitting at collided node and entry is at head ….. by reverse logic if they are meeting then they have to travel same distance (if we increment 1 by 1)
* Now that distance constitutes to be m
  + because we know m+ k is start of the loop and we have travelled k distance there m distance remains
  + if we travel m distance we reach at start of loop and coincidently distance between start of the loop and head node is also m therefore we can say when both nodes collide (start , entry) that would be start of the loop

## First non repeating element in Stream of Inpt :

Question : [Find the first non-repeating character from a stream of characters](https://practice.geeksforgeeks.org/problems/first-non-repeating-character-in-a-stream/0)

Solution:

Practise case : ftvbwnimpvvbfvtot ans : ffffffffffffttwww

* Since hame bola gya hai har incoming element ke liye abhi tak ka pehela non repeating character chahiye
* toh hame incoming elements ko track karna hoga aur usme se possible candidates ko kahi store karna hoga
* Store isliye kyunki agar current first non repating character incoming element se match khata hai toh phir who non repating nhi rha
* tab hame next non repeating element ko dekhna hoga -> isse hame intuition milta hai ki Queue ki jarurat ho skti hai jo possible non repeating ele ko store karte rkhega.
* Isme par ek issue hai sirf queue se kaam nhi chalega kyunki
* Agar hamara q hai [a,b,c,d] aur incoming element “a” aata hai toh ham “a” ko pop kar karwa denge hence q = [b,c,d] and non repating element b ho jayega
* But issue hai agar next bar “a” aata hai toh who abtak ki stream mein hai but who queue mein nhi hai toh isse confusion hoga ki isse queue mein dalna hai ki nhi for possible candidate ke liye.
* hence ham ek dict ki bhi jarurat padegi just to keep track of element which are already in string and do we have to put them in queue or not.
* Next issue yeh ata hai suppose:
* Q = [b,c,d,e,f] aur incoming element is “d”
* So pehele ham dic mein check karenge aur dekhenge ki yeh dic mein toh hai …hence q se delete karna padega since its not a valid candidate anymore.
* But as “d” is in middle(inbtween) of queue hence the TC increases aur har element kar agar same case hua toh TC would reach to O(N) per deletion
* Hence to for O(1) deletion we implement queue as DLL.
* Therefore the flow will be like this
  + While i 0<n:
    - Incoming element – val
    - Check val in dic ?
      * No ?
        + Possible candidate
        + Create a node of DLL append in queue
        + And add its ref to dic as dic[val] = node
      * Yes ?
        + Pop that element from q
        + Remove that node refrence from dic (but entry would still be there to cross check if it comes again)
    - First node val in LL is first non reap occurring till now append it to result string.
  + Return res string

## Flatten a Linked List :

* 2d LL diya hoga har node ka ek next aur bottom pointer hoga LL bottom wise sorted hogi
* 5 -> 10 -> 19 -> 28

| |

7 22

| |

8 50

|

30

* Ans : 5-> 7-> 8- > 10 -> 19-> 20->22-> 28-> 30-> 35-> 40-> 45-> 50.
* Use ek single LL mein convert karna hai such that all element are sorted
* Approach
* Kuch nhi hai pehele 2 bottom LL ko pkdo merge karo phir continuously aage bdhte jao aur incoming bottom LL ko pichele resultant sorted LL ke saath merge karte rho.
* Merge two Sorted LL se merge hoga.

## Find Kth Node from end

* Brute force : Calculate length n aur phir n-k+1 node is answer
* Single Pass:
  + Keep two pointers start and end create a window of size k by expanding end pointer
  + At max window size(K) slide the window by moving both pointers ahead if until end ptr reachs last node
  + Return start node

## Merge two sorted LL

* Intuition toh yehi aayega ki merge sorted array jaise kaam karo and within linked list hi wiring change kardo aaps mein during single pass
  + But isme issue aata hai agar dono pointers same element ko point karte hai toh in both list and internal wiring fails on case 2,3,4,6 & 1,2,4,5,8,10
* Correct tareeka yeh hai ki sui dhage jaise socho ke
* empty pointer banao aur dono list se traverse karo (wohi 2 pointers merge sortd array wala tareeka)
* jaise jaise new elements/nodes milte hai us empty pointer ke age daalte jao.
* Baad mein yeh newly formed list ko return kardo
* It takes O(1) space kyunki ham wohi nodes ko append karhe new order mein

## Sort012

* Yeh wala question thora differ karta hai arrays wale se

* 2 approach
* 1: count 0s 1s and 2s and overrite data in LL
* 2: Sort by changing link
  + Make 3 separate LL(of 0,1,2) via inter changing links in O(n) w/o extra space
  + Then connect them
  + Idea yeh hai ki ham har element (0,1,2) ke liye two pointer rakh rhe hai(total 6) ek startP and endP
  + Initially EK full LL pass marenge jisme startP and endP har element ke first occ ko point karenge
  + Fir ek 2nd time loop chalega
    - Isme ham dekhenge agar curr element firstP wala nhi hai toh hum
    - endP.next = this newly curr element
    - phir endP yeh latest element ko point karega
  + Is loop khatam hone ke baad hame 3 alag alag LL milenge containing only 0s , 1s, 2s
  + Baaki phir hame 0s ke ending -> 1 ke start
  + 1 ka ending -> 2 start
  + And return 0 ka start (edge cases ka dhyaan incase 0,1,2 element is fully missing from list

# Array :

## Sort012

* Idea -> Seggregation
  + Zero ko left side pheko
  + Two ko right side pheko
  + One apne aap middle mien aajayenge
* Mark two **start** = 0 and **end** = n-1 pointers
* Take a **mid** pointer from 0
* Iterate via **mid** pointer jab tak **end** se kam hai
  + One aaya toh ignore karo aur **mid** ko aage badhao
  + Zero aaya toh **start** se swap maro & **Mid** aur **start** ko ek aage badhao
  + Two aaya toh **end** se swap maro aur end ko ek kam karo (Isme mid ko aage nhi badhana kyunki agar mid pe swap ke baad zero bhi aaskta hai )
* Return arr

## RangeQuery Questions and difference array concept

Yeh trick tab kaam aati hai agar ham eek array ke particular range [L,R] ke sare element ko ek given value se update karni ho by default o(n^2) per query hoti h

but yeh o(n) mein karti hai by using prefix sum

PDF Detailed Working : <https://bit.ly/3EdcRqY>

Rest of Notes on GDRIVE : [https://docs.google.com/document/d/10GyHdmhVaNeBTZODxfbxjApuSFcpta8R/edit#](https://docs.google.com/document/d/10GyHdmhVaNeBTZODxfbxjApuSFcpta8R/edit)

1094 Car Pooling

Heaps (there is also a solution based on range query faster than this)

Idea yeh hai ki

Sort on the basis of endpoints

Prepare a running capacity

Prepare a empty pq

* Now for every trip:
  + while pq not empty and top element endpoint < current element start
    - remove previous trips from heap
    - also remove running capacity
  + push current trip into heap
  + add current trip capacity into running capacity
  + if running capacity > total capacity of taxi:
    - return false
* return true

def carPooling(self, trips: List[List[int]], capacity: int) -> bool:

trips.sort(key=lambda x:(x[1],x[2]))

pq = [] #(end,num)

total = 0

for num,s,e in trips:

while pq and pq[0][0] <= s:

total -= heapq.heappop(pq)[1]

heapq.heappush(pq,(e,num))

total += num

if total > capacity:

return False

return True