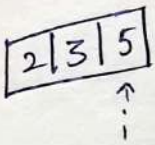
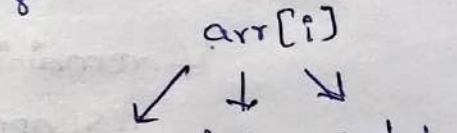
4/12/25

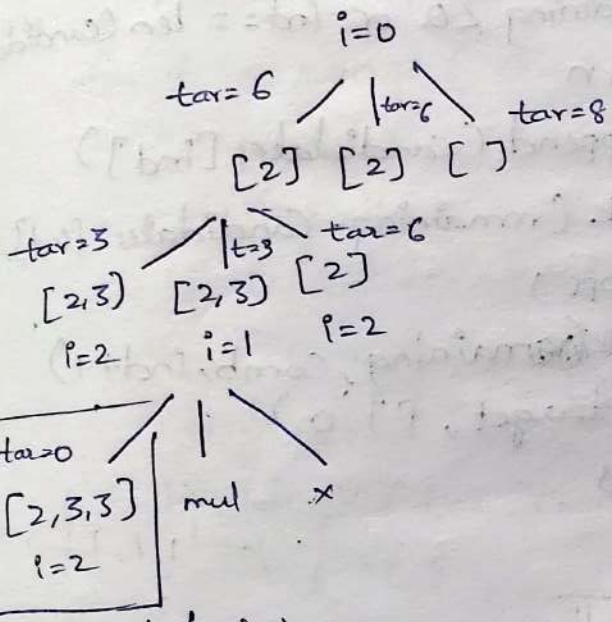
Recursion & Backtracking

Combination sum

1. arr[] = {2,3,5} target = 8







pseudocode/logic:

```
void combSum (arr[], i, combin, ans, tar)
    if (i==n)
        return;
    if (tar==0)
        ans.PB(combin)        } Ans
        return
    combin.PB (arr[i])
Single→ CS(arr, i+1, combin, ans, tar-arr[i])
mul→ CS (arr, i, combin, ans, tar-arr[i])
    combin.popback() → Backtracking
    CS (arr, i+1, combin, ans, tar)
```

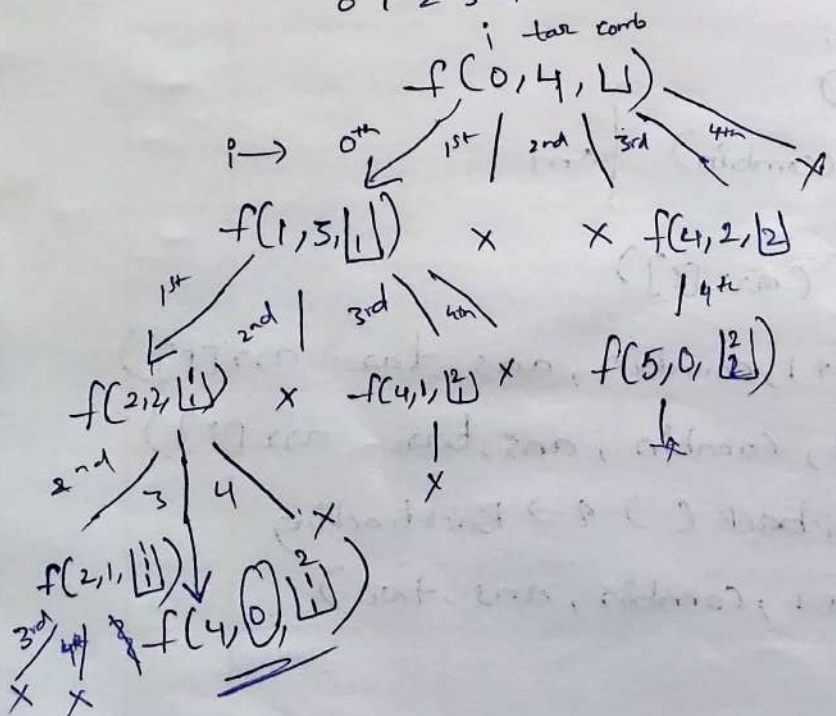
Code:

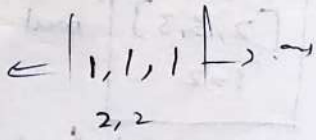
```
def combination(self, candidates, target):
    res = []
    def backtrack(remaining, comb, ind):
        if remaining == 0:
            res.append(list(comb))
            return
        if remaining < 0 or ind == len(candidates):
            return
        comb.append(candidates[ind])
        backtrack(remaining - candidates[ind], comb, ind)
        comb.pop()
        backtrack(remaining, comb, ind+1)
    backtrack(target, [], 0)
    return res
```

2. combination sum - II

arr[] = [1,1,1,2,2] target = 4

0 1 2 3 4





T.C → $2^n \times k$

S.C → $k \times x$

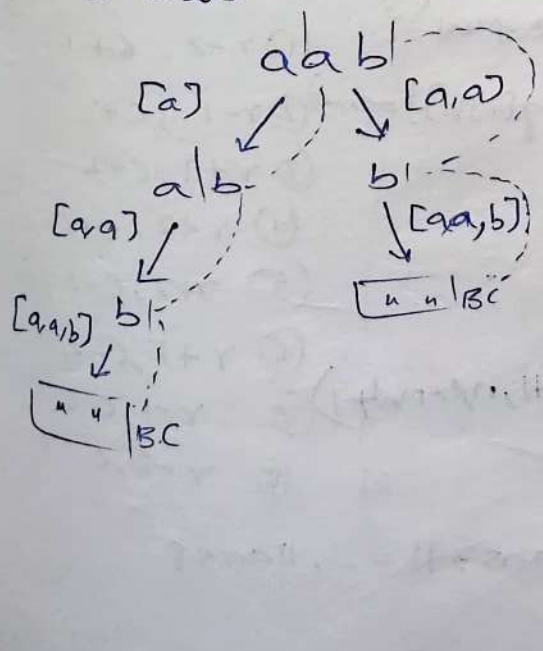
```
if (tar == 0)
    ans.add(comb)
if (ind, tar, c)
    loop(ind->n-1)->(i)
    ls.add(arr[i])
    f(i+1, tar-arr[i], c)
    c.remove(_)
```

Code:

```
candidates.sort()
res = []
def backtrack(tar, comb, start):
    if tar == 0:
        res.append(comb[:])
        return
    for i in range(start, len(candidates)):
        if i > start and candidates[i] == candidat[i-1]:
            continue
        if candidates[i] > target:
            continue
        comb.append(candidates[i])
        backtrack(tar - candidate[i], comb, i+1)
        comb.pop()
    backtrack(tar, [], 0)
```

3. 131. Palindrome Partitioning $O(n \times 2^n)$ T.C

s = "aab"




```
void getAllparts(s, partition, ans) {
    if (s.size() == 0) {
        ans.pb(partition)
        return;
    }
    for(i=0; i<n; i++) {
        String part = S.substr(0, i+1);
        if(isPalin(part)) {
            partitions.pb(part);
            part s.substr(i+1),
                partitions, ans)
            partitions.popback();
        }
    }
```

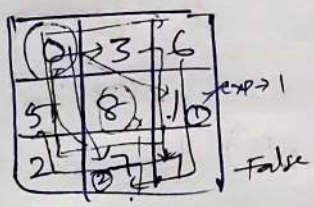
B.C

Rec

```
ans = []
def getAllParts(s, partitions):
    if not s:
        ans.append(listCpartitions))
        return
    for i in range(0, len(s)):
        part = s[0:i+1]
        if part == part[::-1]:  partitio
            partitions.append(part)
            getAllParts(s[i+1:], partitions)
            partitions.pop()
    getAllParts(s, [])
    return ans.
```

4. (25-96) Check Knight Tour Configuration

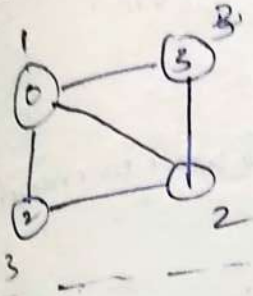
0 to n^2-1




```
bool isvalid(grid[], r, c, n, expval){
if(rcollccollr>=n || c>=n || q[r][c] != expval)
    return false;
if(expval == n^2-1)
    return true;
ans1 = isvalid(grid, r-2, c+1, n, expval+1)
ans2 = ||
       (8)
return ans1 || ans2 || ans3 || ... || ans8
```

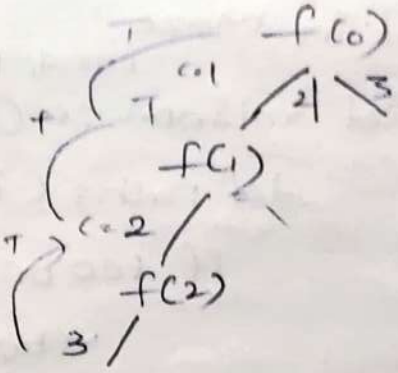
(1) r-2, c+1
(2) r-1, c+2
(3) r+1, c+2
(4) r+2, c+1
(5) r+2, c-1
(6) r+1, c-2
(7) r-1, c-2
(8) r-2, c-1

15. M-Coloring (gfg)



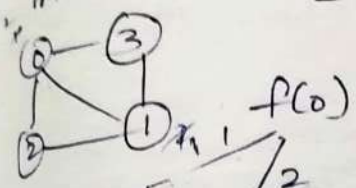
m = 3
V = 4
edge = [[x,y], [u,v], ...]

True



False

m=2




```
def graphColoring(V, edges, m):
    adj=[[] for _ in range(V)]
    for x,y in edges:
        adj[x].append(y)
        adj[y].append(x)
    colors = [0]* V
    def isSafe(node, c):
        for nei in adj[node]:
            if colors[nei]==c:
                return False
        return True
    def Solve(node):
        if node==V:
            return True
        for c in range(1,m+1):
            if isSafe(node,c):
                colours[node]= c
                if Solve(node+1):
                    return True
                colors[node]=0
        return False
    return Solve(0)
```

code

6. Rat in Maze: (no. of poss paths)   T - $O(m^{v} - v)$ S - O(v+e)

```
def ratInMaze(maze):
    def paths(i,j,p,n):
        if i<0 or j<0 or i>n-1 or j>n-1 or maze[i][j]!=1:
            return False
        if i==n-1 and j==n-1:
            res.append(p)
            return
        maze[i][j]=-1
        paths(i+1,j,p+'D',n)
        paths(i,j-1,p+'L',n)
        paths(i,j+1,p+'R',n)
        paths(i-1,j,p+'U',n)
        maze[i][j]=1
    n=len(maze)
    res=[]
    if maze[0][0]==1:
        paths(0,0,"",n)
    return res
```

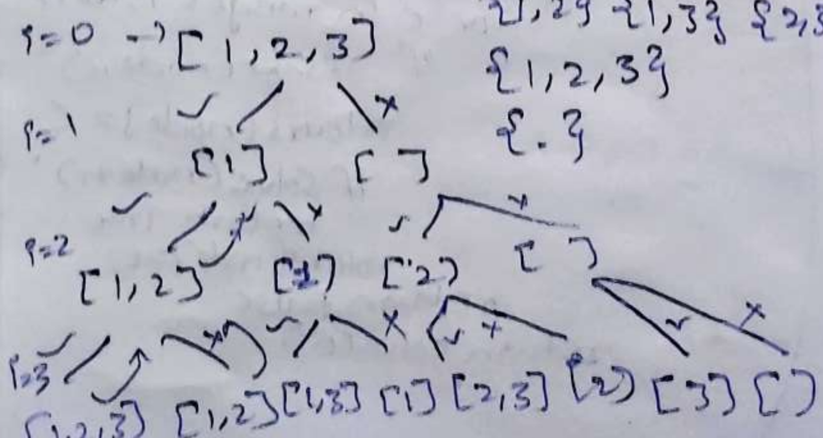
7 Subset-II (no duplicates)

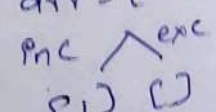
(print all subset)

arr = {1,2,3}   n = 3   n elements → 2^n

{1} {2} {3}
{1,2} {1,3} {2,3}
{1,2,3}
{.}

} $2^3 = 8$





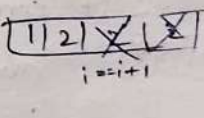
T.C → total calls ✗ work done in each call

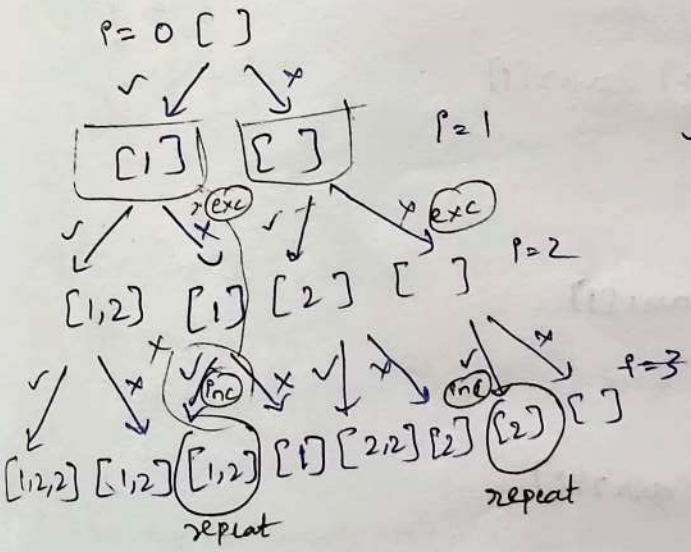
```
void ps(arr[], ans[], i){
    if(i==arr.size()) => print ans
                         return;
    ans.push_back(arr[i])
    ps(arr,ans,i+1) } ✓ inc
    ans.pop_back() ——→ (Backtracking)
    ps(arr,ans,i+1) } exc ✗
}
```

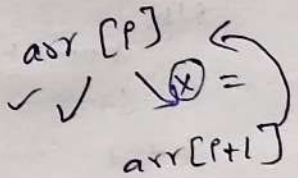
$2^n \times n \times O(1)$
(subsets) (call elements)
$= O(2^n \times n)$

// duplicates exists SS-II

arr = {1,2,2}







① Sort the arr
② calc subsets

```
nums.sort()
allSubsets=[]
def getAllSubsets(idx, ss):
    if i==len(nums):
        allsubsets.append(ss[:])
        return
    #Include
    ss.append(nums[i])
    getAllSubsets(i+1,ss)
    ss.pop()
    idx=idx
    idx=i+1
    while idx<len(nums) and nums[idx]==nums[idx-1]:
```

```
#exclude
getAllSubsets(i,ss)
getAllSubsets(0,[])
return(allSubsets)
print
```

8. (912) Sort ~~An Array~~ : nlogn { T.C → N logN, S.C → O(N)+O(N) }

```
def merge(nums, s, m, e):
    n1 = m+s+1
    n2 = e-m
    arr1 = nums [s: m+1]
    arr2 = nums [m+1: e+1]
    i=0
    j=0
    k=s
    while i<n1 and j<n2:
        if arr1[i] <= arr2[j]:
            nums(k) = arr1[i]
            i+=1
        else:
            nums[k] = arr2[j]
            j+=1
        k+=1
    while i<n1:
        nums[k] = arr1[i]
        i+=1
        k+=1
    while j<n2:
        nums[k] = arr2[j]
        j+=1
        k+=1
def mergesort (nums, s, e):
    if s<e:
        m = s+(e-s)//2
        mergesort (nums, s, m)
        mergeSort (nums, m+1, e)
        merge( nums, s, m, e)
mergeSort (nums, 0, len(nums-1))
print (nums)
```

nums = [5, 2, 3, 1]

[2,5] [5, 2] [3, 1] [1,3]

[5] [2] [3] [1]

s<e ✓ s==e ✗ s>e ✗

mid = s+(e-s)/2

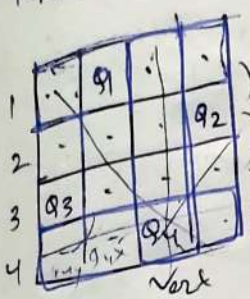
0+(3-0)/2

= 1

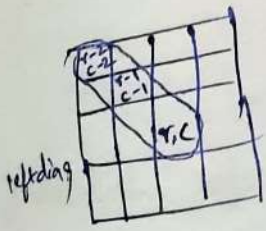
9. (51) NQueens

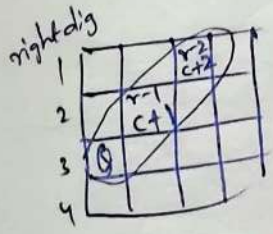
n=4

nxn → board



Q(n) n diff rows
Q(n-1)
Q(n-2)
Q(n-3)
hor
O(n!)






```
n = 4
def isSafe(board, row, col, n):
    # Hor
    for j in range(n):
        if board[i][col] == 'Q':
            return F
    # ver
    for i in range(n):
        if board[i][col] == 'Q':
            return F
    # Left diag
    i, j = row, col
    while i>=0 and j>=0:
        if board[i][j] == 'Q':
            return F
        i -= 1
        j -= 1
    i = row, j = col
    while i>=0 and j<n:
        if board[i][i] == 'Q':
            return false
        i -= 1
        j += 1
    return True
```

```
def nQueens(board, row, n):
    if row == n:
        ans.append(["".join(r) for r in board])
        return
    for col in range(n):
        if isSafe(board, row, col, n):
            board[row][col] = 'Q'
            nQueens(board, row+1, n)
            board[row][col] = '.'
ans = []
board = [['.']*n for _ in range(n)]
nQueens(board, 0, n)
print(ans)
```

10. (37) Sudoku Solver

TLE (9 empty) ← TLE

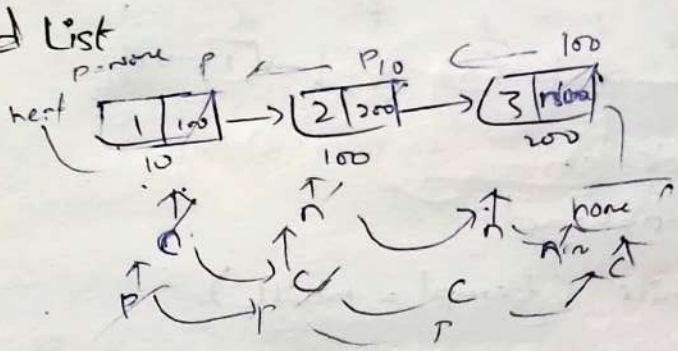
```
def isSafe(board, row, col, dig):
    for j in range(9):
        if board[row][j] == dig:
            ret F
    for i in range(9):
        if b[i][col] == dig:
            ret F
    sr = (row//3)*3
    sc = (col//3)*3
    for i in range(sr, sr+3):
        for j in range(sc, sc+3):
            if board[i][j] == dig:
                ret false
    ret True

def helper(board, row, col):
    if row == 9:
        ret T
    if col == 8:
        nextRow = row+1
        nextCol = 0
    else:
        nextRow = row
        nextCol = col+1
    if board[row][col] != '.':
        ret helper(board, nextRow, nextCol)
    for dig in "123456789":
        if isSafe(board, row, col, dig):
            board[row][col] = dig
            if helper(b, nextR, nextC):
                ret T
            board[row][col] = '.'
    ret F

helper(board, 0, 0)
```

rows = [set() for _ in range(9)]
cols = [set()]
boxes = [set()]

Linked List

7/12

1) Reversed Linked List

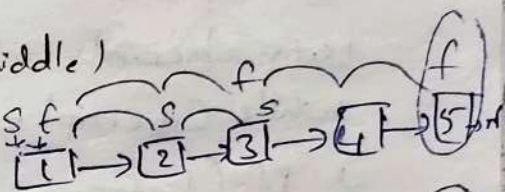



```
p = None
c = head
while c:
    n = c.next
    c.next = p
    p = c
    c = n
return p
```

2) Middle of a LL

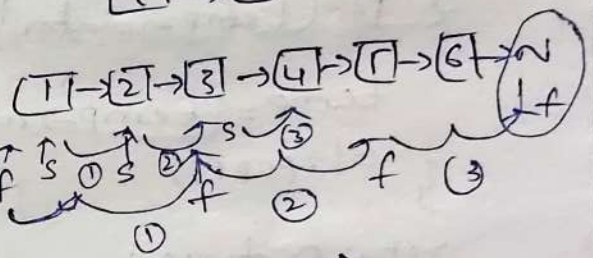
(1) Brute force
(2) slow-fast ✓

even - (2 middle)



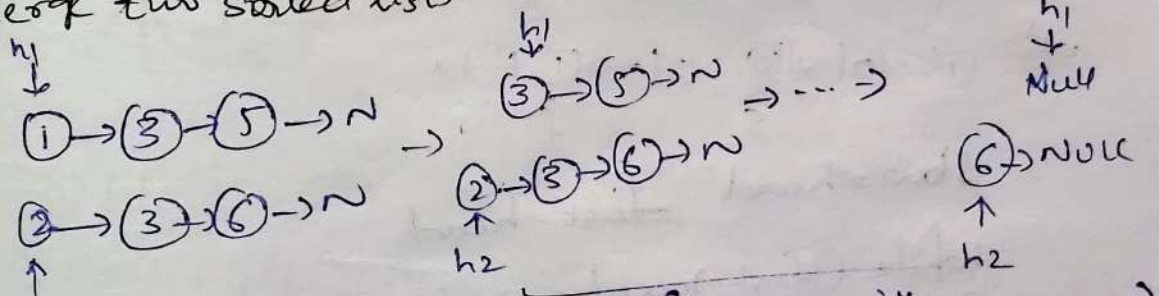

```
slow = head
fast = head
while(fast != NULL && fast->next != NULL)
{ slow = slow->next
  fast = fast->next->next
}
return slow;
```

odd



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

3) (21) Merge two sorted lists



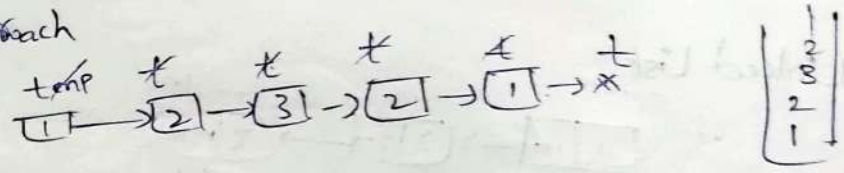
1 2 3 3 5 6

T.C : O(n+m)

```
if (h1 == NULL || h2 == NULL)
    return h1 ? h1 : h2;
if (h1->val <= h2->val)        // case1
    h1->next = merge(h1->next, h2);
    return h1;
else                           // case2
    h2->next = merge(h1, h2->next);
```

10 (1) (234) palindrome linked list

(1) approach




```
Stack st
temp=head
while (temp!=NULL)
{   st.push(temp->data)
    temp=temp->next;
}
```

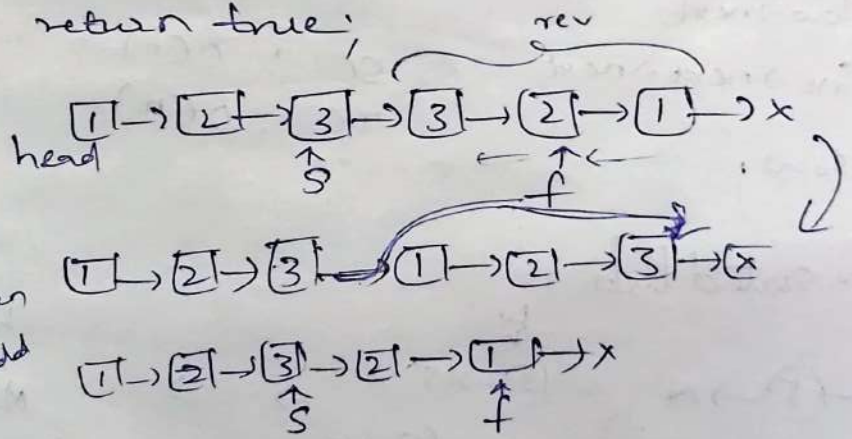
step 1: O(N)

```
temp=head
while (temp!=NULL)
{ if (temp->data!=st.top()) return false;
  temp=temp->next
  st.pop();
}
return true;
```

O(N)

TC → O(2N)
SC → O(N)

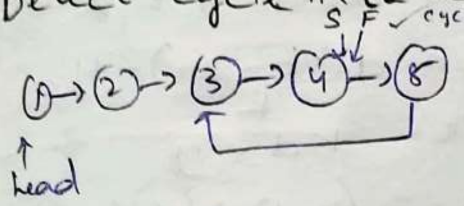
(2)

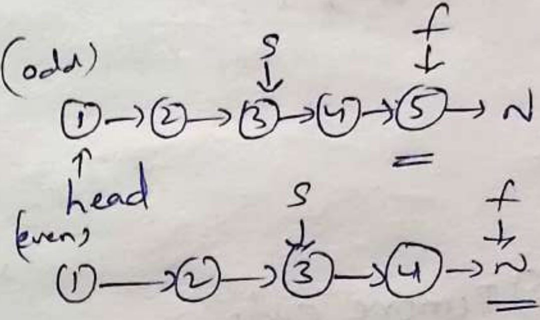



```
slow=head  fast=head
while (fast->next!=NULL && fast->next->next!=NULL)
{ slow=slow->next
  fast=fast->next->next
node * newnode=reverse(slow->next)
first=head  second=newhead
while (second!=NULL)
{ if (first->data!=second->data)
  {
```

~~reverse (newhead)~~
return false;
}
}
~~reverse (newHead)~~ first = first -> next
second = second -> next
return true

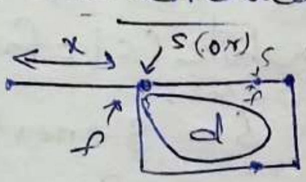
5) Detect cycle in LL (141) (odd)

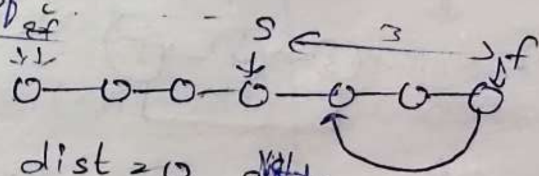




s = head
f = head
while (f != NULL && f -> next != NULL)
{
s = s -> next
f = f -> next -> next
if (s == f) return true
}
~~ret~~
return false

Mathematical proof

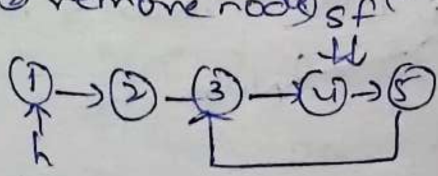




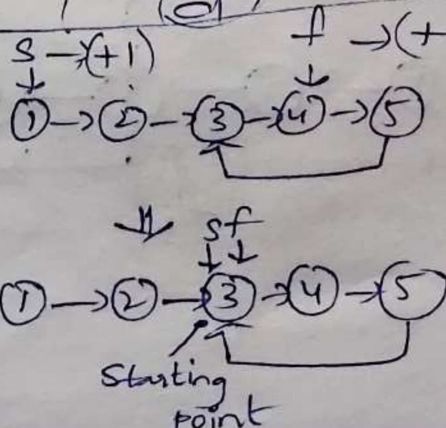
dist = 0 ~~dist~~ k
= 0+1 k+1
= 2 k+2
= 3 k+3
= 4
d

6) (142) LL-II
① start node
② remove node } (+1) (+2) sf

①

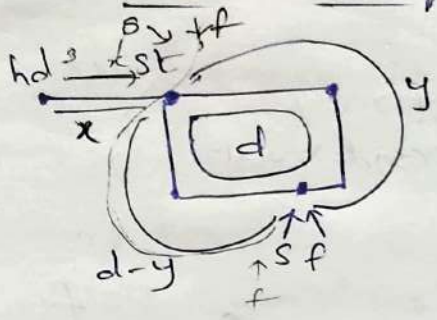


=>



(i) find a cycle exists or not
(ii) s = h L if [s = f]:
while (s != f) {
s = s -> next
f = f -> next
}
return

Mathematical proof



Slow dist = $x + a \times d + y$ (a rounds)

fast dist = $x + b \times d + y$

fast = 2 × slow

$x + bd + y = 2x(x + ad + y)$

$x + bd + y = 2x + 2ad + 2y$

$bd - 2ad - y = x$

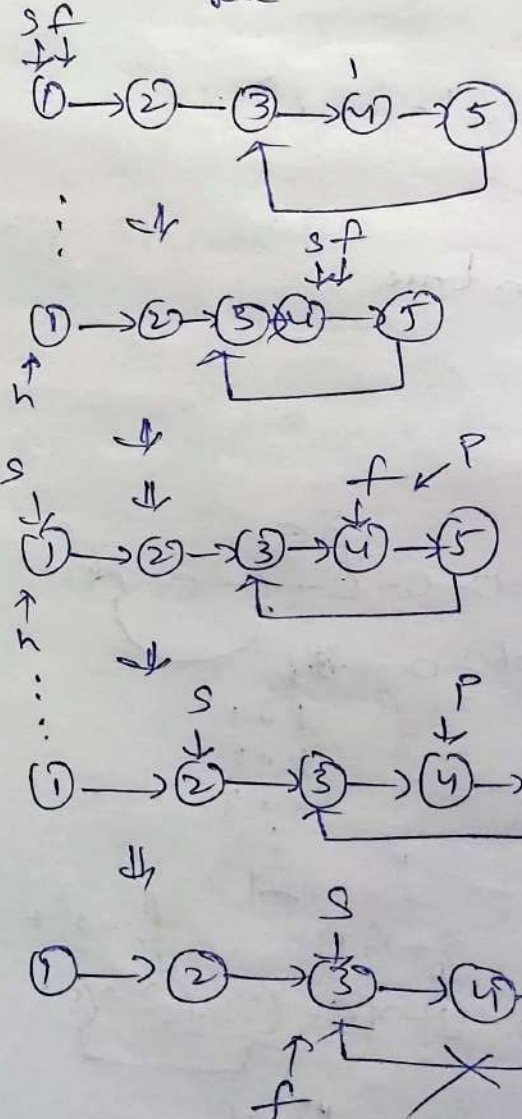
$x = d(b - 2a) - y$

$x = kd - y$

$x = (k-1)d + (d-y)$

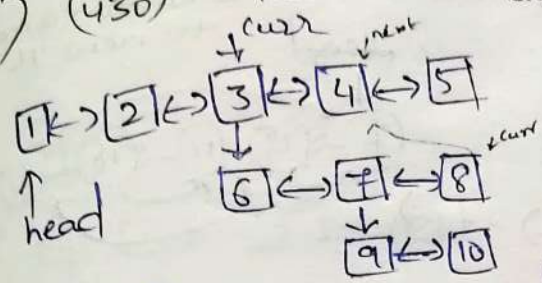
s = head; s = x; f = $(k-1)d + (d-y)$

(2) Remove cycle

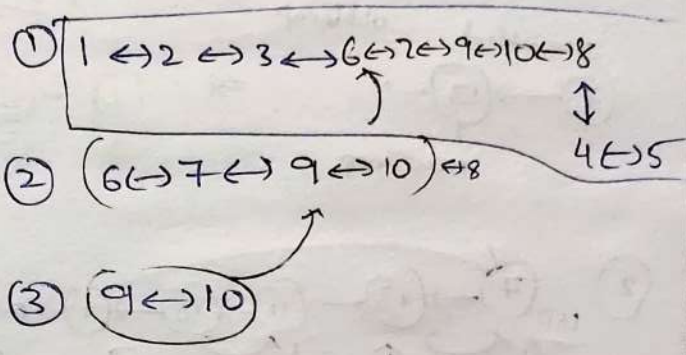


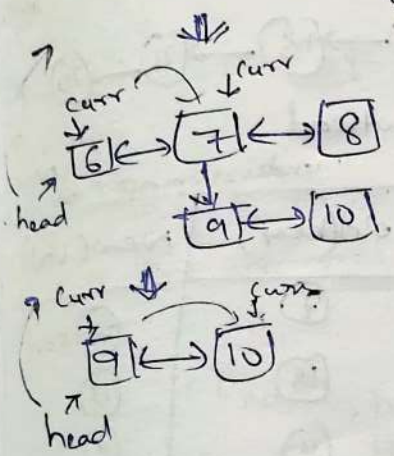

```
if (!isCycle){
    return NULL
}
s = head
P = NULL
while (s != fast)
{
    s = s->next
    P = f
}   f = f->next
P->next = NULL  // remove cycle
return s;
```

1) (430) — Flatten a multilevel Doubly LL



node
{ val
next
prev
child
}






```
Node* flatten (Node* head){
    head = Null → return head
    Node* curr = head
    while (curr != NULL)
    { if (curr→child != NULL)
      { Node * next = curr→next
        curr→next = flatten (curr→child)
        curr→next→prev = curr
        curr→child = NULL

        while (curr→next != NULL)
            curr = curr→next

        if (next != NULL)
        { curr→next = next
          next→prev = curr
        }
      }
      curr = curr→next
    }
    return head
```

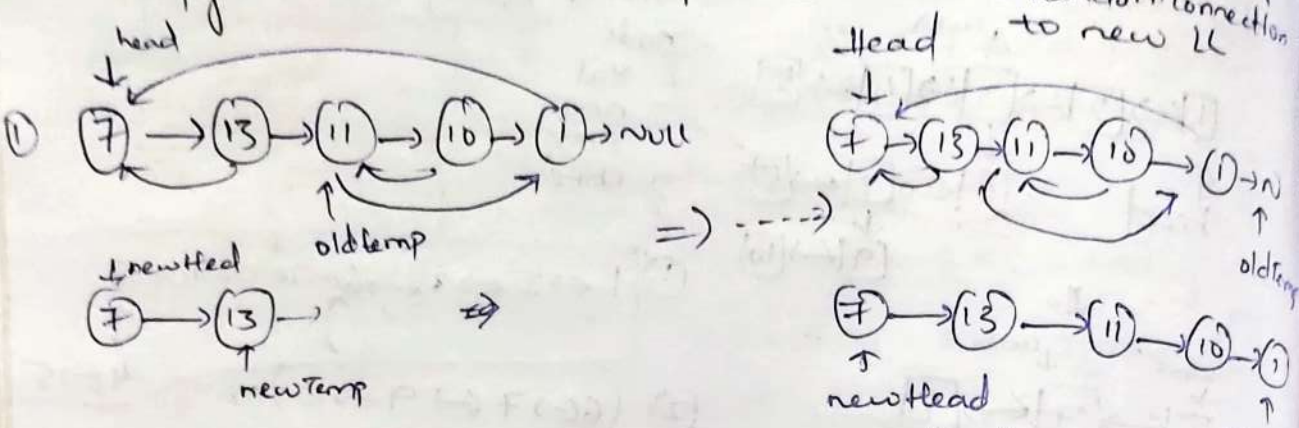
① flatten the child node

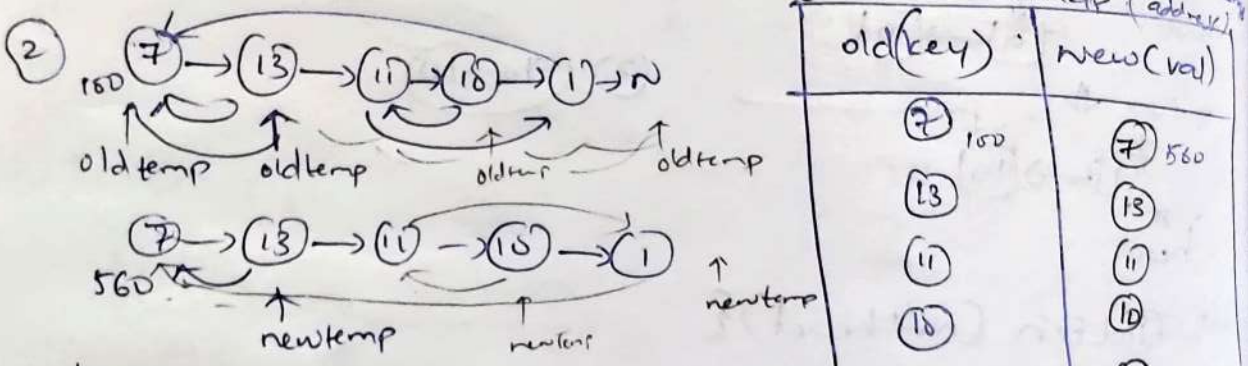
② find tail

③ attach tail with next ptr

8) Copy list with Random pointer

((Hard)) (1) Simple copy (2) Random connection to new LL





unordered map (address)

old(key)	new(val)
7 180	7 560
13	13
11	11
10	10
1	1

Logic

```
unordered_map<N,N>m;
Node* newHead = newNode(Head_val)
 oldTemp = head->next
 newTemp = newHead
 m[head] = newHead
 while(oldTemp != NULL)
 { Node* copyNode = newNode(oldTemp->val)
    m[oldTemp] = copyNode;
    newTemp->next = copyNode
    oldTemp = oldTemp->next
 } newTemp = newTemp->next
```

(1)

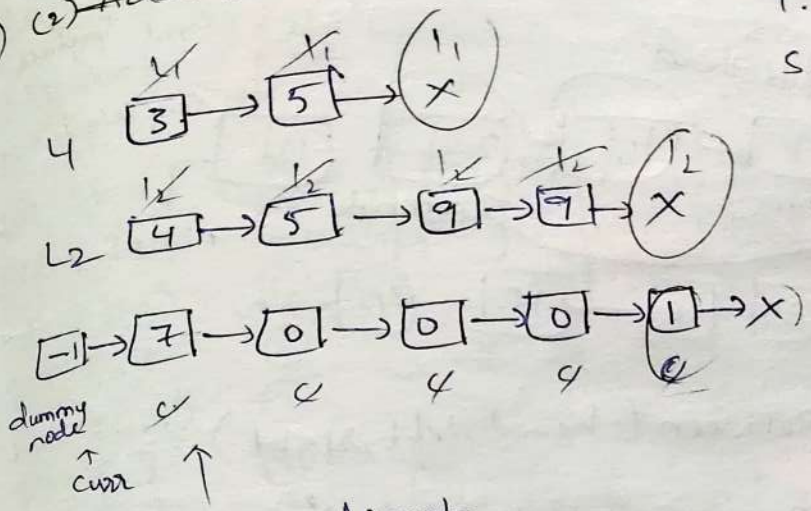
```
oldTemp = head; newTemp = newHead;
while(oldTemp != NULL)
{ newTemp->random = m[oldTemp->random]
   oldTemp = oldTemp->next
} newTemp = newTemp->next
return newHead
```

(2)

9) (2) Add 2 numbers

T.C → O(max(N1, N2))

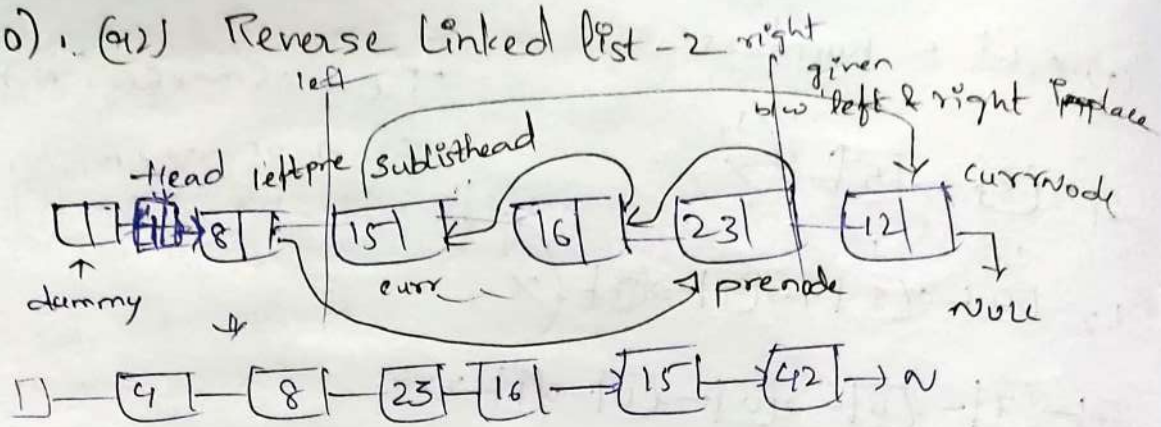
SC → O(max(N1, N2))



logic

```
func (head1, head2)
    t1 = head1, t2 = head2
    dummyNode = newNode(-1)
    curr = dummyNode
    while (t1 != NULL || t2 != NULL)
    { sum = carry
      if (t1) sum = sum + t1->data
      if (t2) sum = sum + t2->data
      newNode = newNode(sum%10)
      carry = sum/10
      cur->next = newNode;
      cur = cur->next

      if(t1) t1 = t1->next
      if (t2) t2 = t2->next
    }
    if(carry)
    { newNode = newNode(carry)
    } curr->next = newNode;
    return dummynode->next
```

10). (a) Reverse Linked list - 2

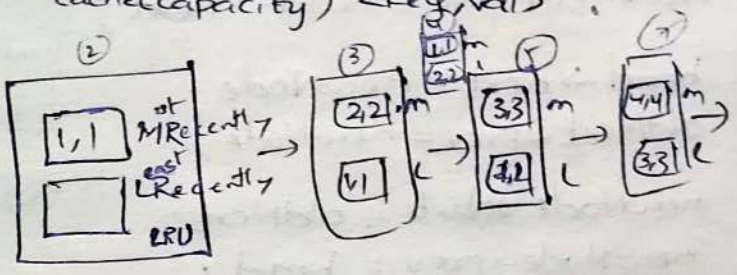



```
ListNode ReverseBetween (head, left, right)
{ ListNode dummy = new ListNode(0);
  dummy.next = head;
  ListNode leftpre = dummy.   (useful when left is head)
  ListNode currNode = head;
  for (int i=0; i<left-1; i++)
  { leftpre = leftpre.next;
  } currNode = currNode.next;
  ListNode SublistHead = currNode;   // marker - where we start reversing
  ListNode preNode = null;
  for (int i=0; i<=right-left; i++)   // r-l+1 times
  { ListNode nextNode = currNode.next;
    currNode.next = preNode;
    preNode = currNode;
    currNode = nextNode;
  }
    // Join the Piece
  leftpre.next = preNode;
  SublistHead.next = currNode;
  return dummy.next;
}
```

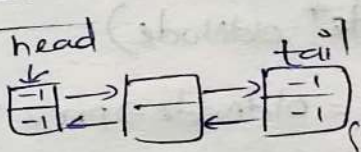
11) (146) Design an LRU Cache

- LRU cache (2) ①
- put (1,1) ②
- put (2,2) ③
- get [1] ④
- put (3,3) ⑤
- get [2] ⑥
- put (4,4) ⑦
- get [1] ⑧
- get [3] ⑨
- get [4] ⑩

cache(capacity) <key, val>



1 -1 -1 3 4

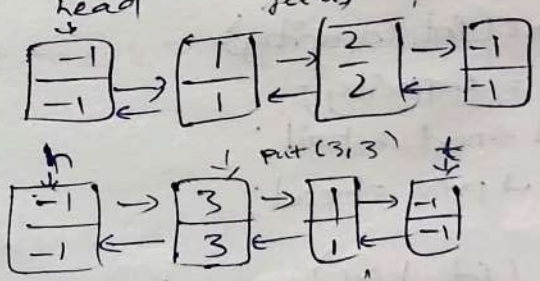


- DLL
- map
- helper funct (add, del)

{ LRU → end of DLL
MRU → start of DLL }

key	Node
1	(1,1)
~~2~~	~~(2,2)~~
3	(3,3)

```
Class Node {
    Node* next, prev
} int key, val
```



T.C -> O(1)

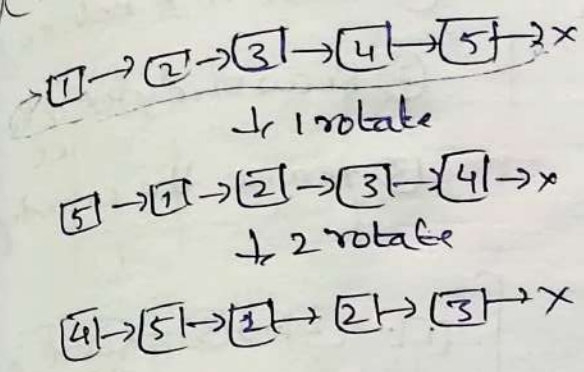
Code:

```
class Node{
    Node* next, prev;
    int key, val;
    Node (self, k, v):
    {   key = k
        val = v
        prev = next = NULL;
};  }

Node *head = new Node(-1, -1);
Node *tail = new Node(-1, -1);

Unordered_map<int, Node*> m;
int limit;
```

↳

```
void addNode (Node* newNode)                    O(1)
{ Node* oldNext = head->next;

  head->next = newNode;
  oldNext->prev = newNode;

  newNode->next = oldNext;
} newNode->prev = head;

void delNode (Node* oldNode)
{
  Node* oldPrev = oldNode->prev;
  Node* oldNext = oldNode->next;

  oldPrev->next = oldNext;
} oldNext->prev = oldPrev;

LRUCache(int capacity)
{ limit = capacity;
  head->next = tail;
} tail->prev = head;

int get (int key)                               O(1)
{ if (m.find(key) == m.end()) // if key doesn't exist
  { return -1
  }
  Node* ansNode = m[key];        // (key, node) (1, (1,1))
  int ans = ansNode->val;

  m.erase(key);
  delNode (ansNode);
  addNode (ansNode);             // update to MRU
} m[key] = ansNode;
  return ans;

void Put (int key, int val)                     O(1)
{ if (m.find(key) != m.end()) // exist
  { Node* oldNode = m[key];
    delNode (oldNode);
  } m.erase (key);
  if (m.size() == limit) // delete LRU data
  { m.erase (tail->prev->key);
  } delNode (tail->prev);

  Node *newNode = new Node(key, val);
  addNode (newNode);
  m[key] = newNode;
  }
};
```

12) (61) Rotate a LL



k = (len) multiple – no change

5+5+5 = 15

5+5+4 = 14

k % len

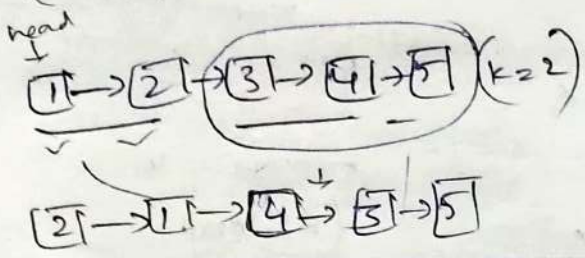
15 % 5 = 0

14 % 5 = 4

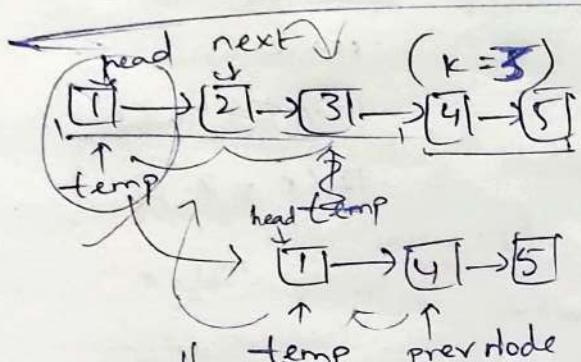
12 % 5 = 2

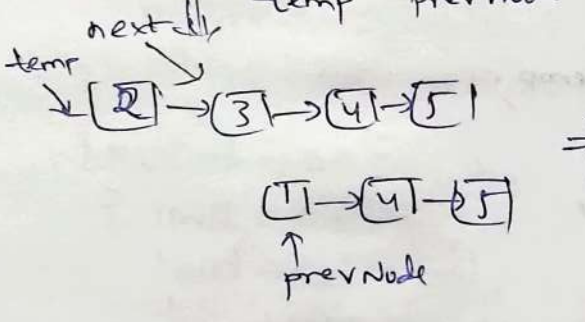
```
Node* findNthNode (node* temp, int k)
{ int cnt = 1;
  while (temp != NULL)
  { if (cnt == k) return temp;
    cnt++;
    temp = temp->next;
  }
  return temp;
}

Node* rotate (Node* head, int k)
{ if (head == NULL || k == 0) return head;
  Node* tail = head;
  int len = 1;
  while (tail->next != NULL)
  { tail = tail->next;
  } len += 1;

  if (k % len == 0) return head;

  k = k % len
  // attach tail to head
  tail->next = head
  Node* newLastNode = findNthNode (head, len-k)

  head = newLastNode->next;
} newLastNode->next = NULL;
```

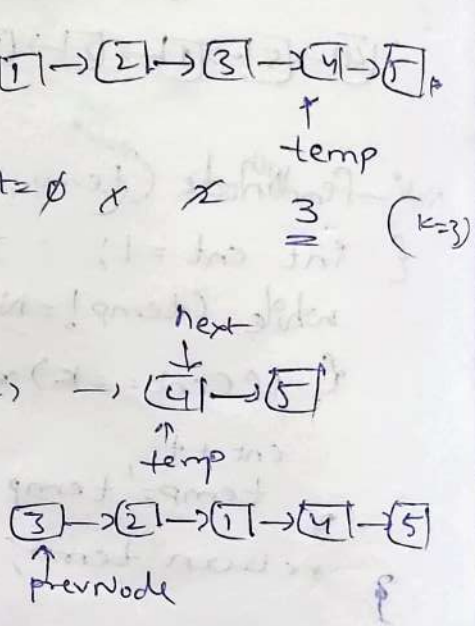
13. (25) Reverse k-Group LL



(1) check if k nodes exist
(2) Recursively rest of the LL
(3) reverse the current group








```
Node* reverse (head, k)
{ Node* temp = head
  cnt = 0
  while (cnt < k){
      if (temp == NULL)
          return head;
      temp = temp->next
  }
prevNode = reverse (temp, k)

temp = head ; cnt = 0
while (cnt < k)
{ Node* next = temp->next
  temp->next = prevNode
  prevNode = temp
  temp = next
  cnt++ ;
}
```

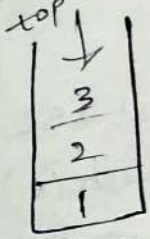
new head of LL
↓
return prevNode

9/12/25

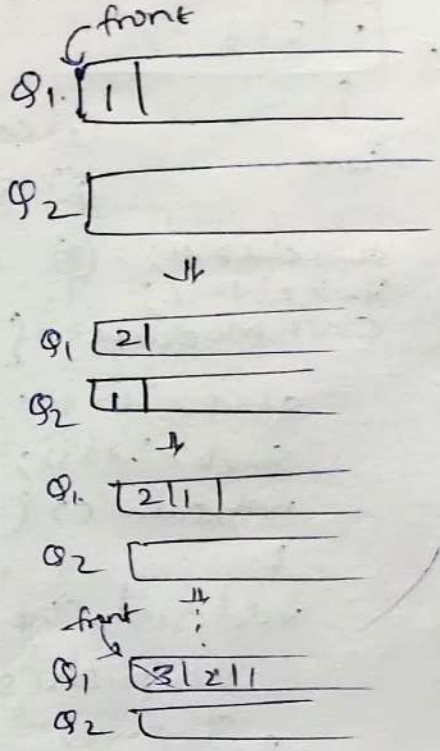
Stack & Queue

1. (225) Stack using Queues

push
pop
top
empty



stack
LIFO



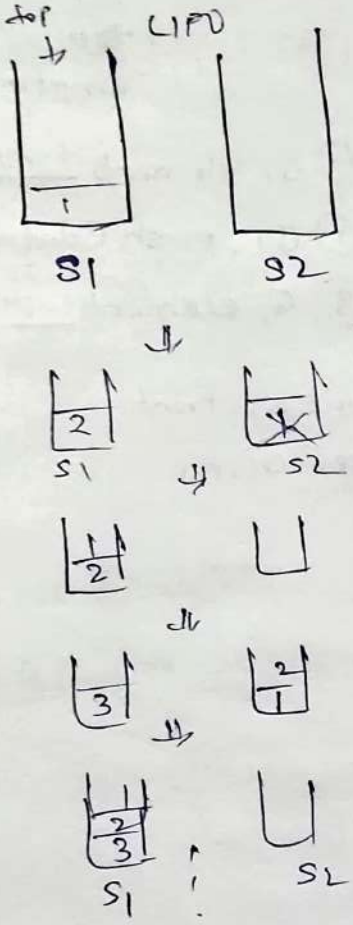
① Q1 elements copy → Q2
② Q1.push(data)
③ Q2 elements copy back → Q1

top = Q1.front
pop = Q1.pop

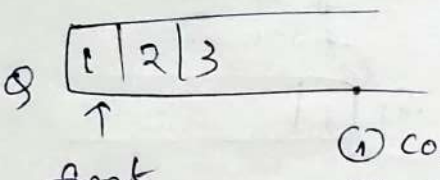
```
class MyStack {
    queue<int> q1;
    queue<int> q2;
MyStack(){
}
void push(int x){
    while(!q1.empty()){
        q2.push(q1.front());
    } q1.pop();
    q1.push(x);
    while(!q2.empty()){
        q1.push(q2.front());
    } q2.pop();
}
int pop(){
    int ans = q1.front();
    q1.pop();
    return ans;
}
```

```
int top(){
    return q1.front();
}
bool empty(){
    return q1.empty();
}
```

2. (232) Implement Queue using Stacks



FIFO



① copy S1 → S2
② S1.push(x)
③ copy S2 → S1

~~Stack<int> s1;~~
~~Stack<int> s2;~~

```
Class MyQueue {
    stack<int> s1;
    stack<int> s2;
    MyQueue() {
    }
    void push(int x) {
        s2.push(s1.top());
        s1.pop();
        s1.push(x);
        while(!s2.empty()){
            s1.push(s2.top());
            s2.pop();
    }}
    int pop() {
        int ans = s1.top();
        s1.pop();
    } return ans;

    int peek() {
    } return s1.top();

    bool empty() {
        return s1.empty();
    }
```

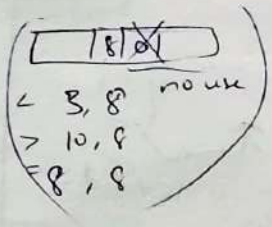
3. Next Greater Element

(arr[i] → immediately greater
i → right)

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

arr = [6, 8, 0, 1, 3]

8, -1, 1, 3, -1



i) right elements
ii) greater than curr ele
iii) immediately

```
stack<int> s;
vector<int> NG(arr.size(),0);   (ans arr)
for(i=n-1;i>=0;i--)
{ while (s.size()>0 && s.top()<=arr[i])
    { s.pop();                    # removing invalids
    }
    if(s.empty()) → -1
    else → NG[i]= s.top
    s.push(arr[i])
}
```

(496)

nums1 = [4,1,2] nums2 = [1,3,4,2]

ans = [-1 | 3 | -1]

① nums2 → Next Greater

map —

nums2[i]	NG
1	3
3	4
4	-1
2	-1

```
def nextGreaterElement(nums1, nums2):
    stack = []
(NG) nextGreater = {}
    for i in reversed(nums2):
        while stack and stack[-1]<i:
            stack.pop()
        if not stack:
            NG[i] = -1
        else:
            NG[i] = stack[-1]
        stack.append(i)
    return [NG[x] for x in nums1]
```

② nums1 → search

map[nums1[i]]

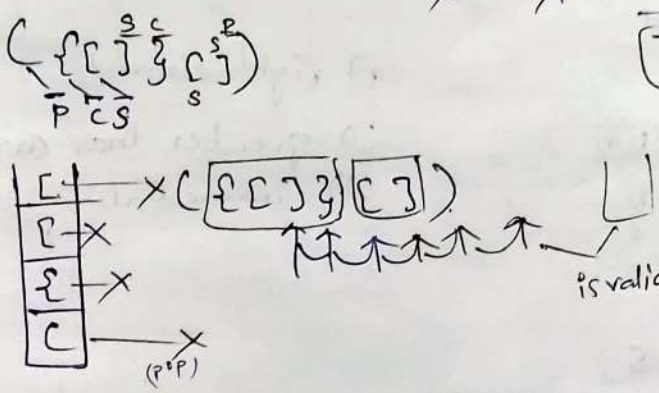
NG

4. (20) valid parenthesis

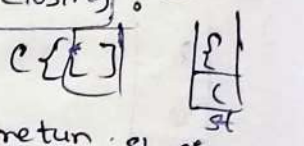
[Last opening bracket ⇒ first closing bracket]

({ [ ] }) { } ✓ {] × ({ } ×

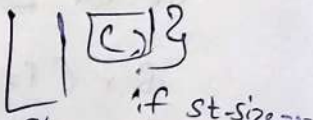
Stack - LIFO



open > closing :-



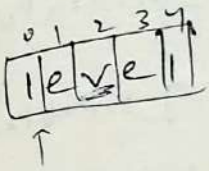
closing > opening;

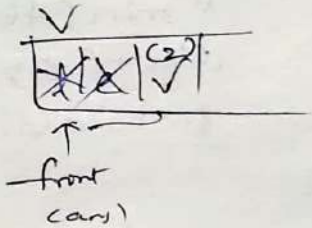



```
bool isvalid(str)
{ stack<char> st;
  for(i=0 to n)
  { if(str[i] => opening)
    { st.push(ch)
    }
    else:
    { if(st.size()==0)
        return false } close opening
      if(str[i] match with st.top())
        st.pop()
      else
        return false;
    }
  return st.size()==0; } oper > close
}
```

5. (387) First unique character in a string

str = "level"



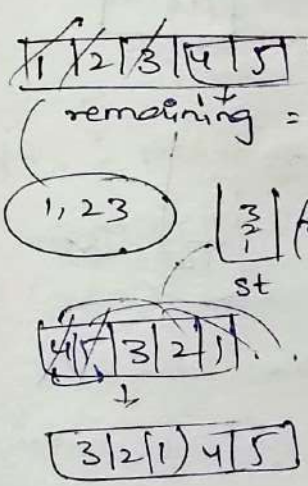


char	int
l	2
e	2
v	1
*	
*	

```
unordered map<char,int> m;
queue<int> q;
for (i=0 to n)
{ if (m.find[s[i]] == m.end()) (if not in dict)
      q.push(i);
  m[s[i]]++;
while(q.size()>0 &&
      m[s[q.front()]]>1)
{ q.pop();
}
return q.empty() → -1
else → q.front()
```

6. Reverse first k elements of queue (gfg)

k=3, q = |1|2|3|4|5|



① Container
② reverse container
③ push them in queue
④ Remaining pop

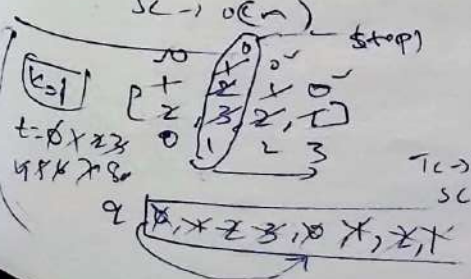
```
def revFirstK(q,k):
    if k>len(q) or k<=0:
        return q
    stack = []
    for i in range(k):
        stack.append(q.popleft())
    while stack:
        q.append(stack.pop())
    rem = len(q) - k
    for _ in range(rem):
        q.append(q.popleft())
    return q
```

*. (2073) Time Needed to buy Ticket

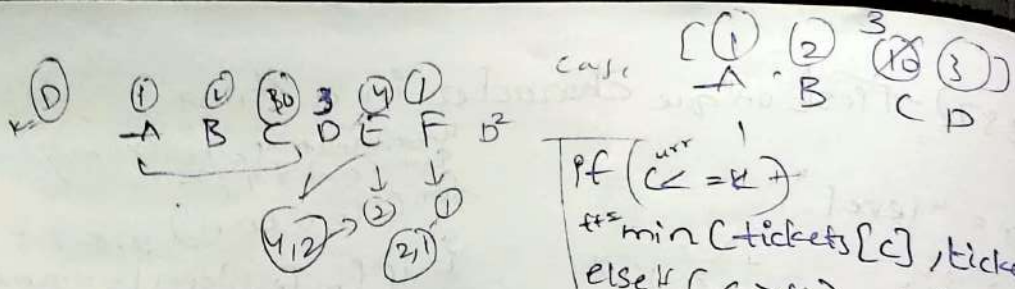
k=1, [2, 3, 2, 1] ← no. of tickets to buy
0 1 2 3 ← person

T = 0 2+3+2+1 = 8

TC → O(n)
SC → O(n)

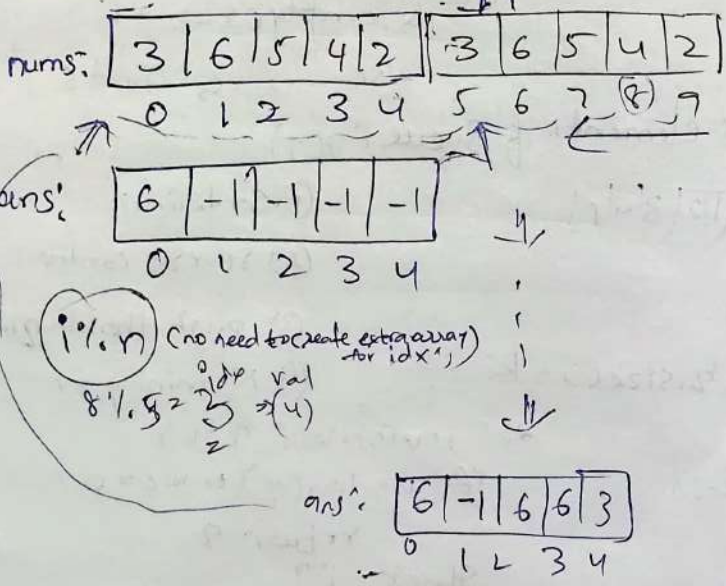


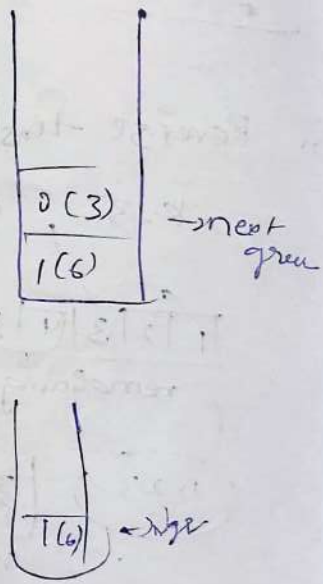

```
func(tickets, k)
{ int time=0;
  for(int c=0; c<tickets.length; c++)
  { if(c<=k)
    { time=time+min(tickets[c],
                    tickets[k]);
    }
    else
    { time=time+min(tickets[c],
                    tickets[k]-1);
    }
  }
  return time;
```



8. Next Greater Element - II (circular)

nums = [1,2,3,4,3] → [3,6,5,4,2]






```
vector<int> ans(n, -1)
Stack<int> s
for(2n-1 to 0)
{
  while(s.size()>0 && nums[s.top()] <= nums[i%n])
  {
    s.pop()
  }
  ans[i%n] = s.empty()? -1 : nums[s.top()];
  s.push(i%n); // valid id
}
return ans
```

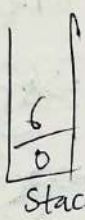
9. previous Smaller Element

TC - O(n)
SC - O(n)

arr = [3,1,0,8,6]

arr[i] → immediately smaller & ts on left

-1 -1 -1 0 0




```
vector<int> ans
stack<int>S;
for (i=0; i<n ;i++)
{ while (S.size()>0 && S.top()>=arr[i])
  { S.pop()
  }
  if(S.empty()) → -1
   else → S.top()
} S.push(arr[i])
```

(1856) Maximum Subarray Min-product:

nums
(Integer array)
↳ subarray

minproducts = [sum(subarray_i) × min(subarray_i) ∀]
max(min products)

| 1 | 2 | 3 | 2 |

(sum * min)

[1] => 1.1 = 1 [2] => 2.2 = 4 [3] => 3.3 = 9 [2] => 2.2 = 4

[1,2] => 3.1 = 3 [2,3] => 5.2 = 10 [3,2] => 5.2 = 10

[1,2,3] => 6.1 = 6 [2,3,2] => 7.2 = 14

[1,2,3,2] => 8.1 = 8

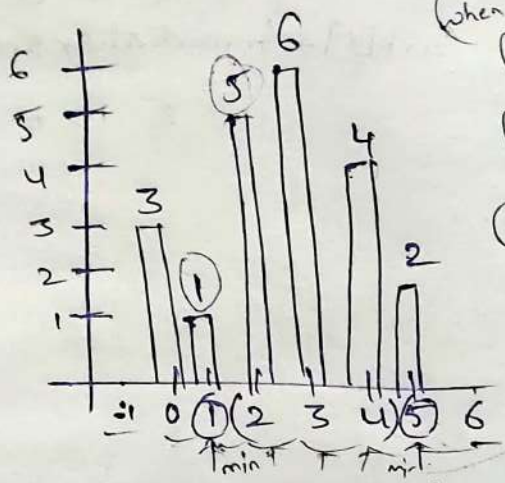
max

A B C D

4 + 3 + 2 + 1

n + (n-1) + (n-2) ... + 1 → n(n+1)/2 ⇒ O(n²)

Time - O(n)



(when min is assumed at)

③ [3] => 3 . 3 => (9)

① [3, 1, 5, 6, 4, 2] => (1 . 21) => (21)

⑤ [5, 6] => 5 . 11 => (55)

⑥ [6] => 6 . 6 => (36)

④ [5, 6, 4] => 4 . 15 => (60)

② [5, 6, 4, 2] => 2 . 17 => (34)

[3

(ele → +ve
more ele → sum greater)

nums = | 3 | 1 | 5 | 6 | 4 | 2 |
0 1 2 3 4 5

prefSum = | 0 | 3 | 4 | 9 | 15 | 19 | 21 |
0 1 2 3 4 5 6

Sum => PSum[rr] - PreSum[ll]
19 - 4 => 15

SmallerOnLeft = | -1 | -1 | 1 | 2 | 1 | 1 |

SmallerOnRight = | 1 | 6 | 4 | 4 | 5 | 6 |

(boundary)

class sol {

```
public int maxSumMinProduct(int nums[])
{ int n = num.length;
  long PSum[] = new long[n+1];
  for(int i=0; i<n; i++)
  { PSum[i+1] = PSum[i] + nums[i];
  }
  int smallerOnLeft[] = NSEL(nums);
  int SmallerOnRight[] = NSER(nums);
  int max = long.MIN_VALUE;
  for(int i=0; i<n; i++)
  { int min = nums[i];
    int lr = smallerOnLeft[i];
    int rr = smallerOnRight[i];
```

① ② ③

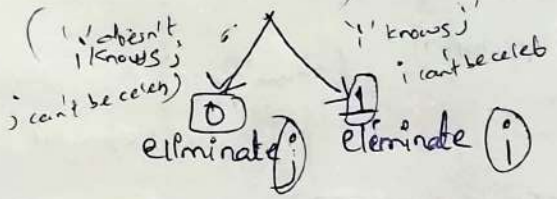
```
long sum = psum[rr] - psum[lr+1];
long minProduct = min x sum;
max = Math.max(minProduct, max);
}
return (int)(max % 1000000007);      (return in 10^9+7)
}                                     -fit in 64 bit signed int

public static int[] NSEL(int nums[])
{  Stack<Integer> st = new Stack<>();
   int res[] = new int[nums.length];
   for(int i=0; i<nums.length; i++)
   { while(st.size()>0 && nums[st.peek()] >= nums[i])
      { st.pop();
      }
      if(st.size()==0)
      { res[i] = (-1)
      }
      else
      { res[i] = st.peek();
      }
      st.push(i)
   }
   return res;
 }
}

public static int[] NSER (int nums[])
{   "      "
   for(int i=nums.length-1; i>=0; i--)
   {  while (st.size()>0 && nums[st.peek()] >= nums[i])
      { st.pop();
      }
      ,
    , "
       res[i] = st[-1] if st else (n)        (7)
 } "   "   st.append(i)
```

10. Find Celebrity problem

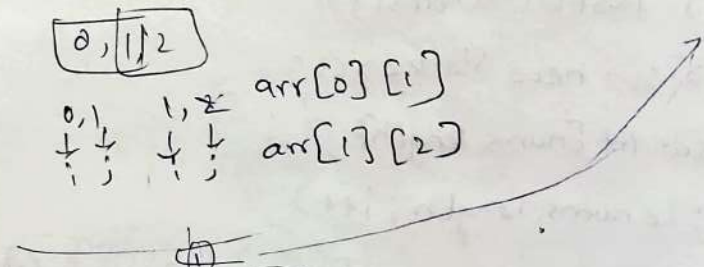
arr = [[0,1,0],[0,0,0],[0,1,0]]

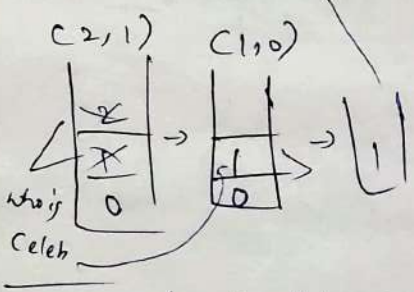
	0	1	2
0	0	1	0
1	0	0	0
2	0	1	0

arr[i][j] =



everyone should know celeb
(1) arr[i][celeb] = 1 ∀ i
(2) arr[celeb][i] = 0 ∀ i
(celeb should not know anyone)





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

```
stack<int>S;
for (i=0 to n-1)
    S.push(i)
while (S.size()>1)
{   i = S.top(); S.pop();
    j = S.top(); S.pop()
    if (arr[i][j]==0)
        S.push(i)
    else
        S.push(j)
}
```

```
int celeb = S.top()
for (i=0; i<n; i++)
{ if (i!=celeb) &&
    (arr[i][celeb]==0 ||
     arr[celeb][i]==1)
  { return -1;
  }
}
return celeb;
}
```

11. (155) min stack

- push -2
- push 0
- push -3
- getMin
- pop
- top
- getmin

minstack: | 0 | -2 | → -3, 0, -2

minstack: | -3,-3 | 0,-2 | -2,-2 | (-3, 0, -2)

(1) stack<pair<val,minval>> s;

```
minstack() { }
void pust(int val)
{ if (s.empty())
  { s.push({val, val}); }
  else
  { int minval = min(val,
                 s.top().second);
    s.push(val, minval);
  } }
void pop()
{ s.pop(); }
void top()
{ return s.top.first; }
void getMin()
{ return s.top().second; }
```

(for push) SC: O(2×n)

⇓

SC: O(n)

(2)

min val = -2 -3 (old, new)

push val < minval;

(formula) $val' = 2 \times val - minval$ (new, old)

minval = val

$= 2(-3) - (-2) = -6 + 2 = -4$

Stack: | -3 | -4 | 0 | -2 | (-4 < -3, (-3) min)

pop

$\text{old minval} = 2 \times \text{new minval} - val'$

$= 2 \times (-3) - (-4)$

$= -2$

```
int getMin()   // O(1)
{ return minval;
}
void push(val)
{
  if (s.empty())
    → s.push(val)
    → minval = val
  elif (val < minval)
  { s.push(2×val - minval)
    minval = val
  }
  else
  { s.push(val)
} }

void pop()
{ if (s.top() < minval)
  { minval = 2×minval - s.top();
  }
  s.pop()
}

int top()
{ s.top() < minval
    → return minval
  else
    → return s.top();
}
```

12. (134) Gas Station

gas = [1, 2, 4, 5, 9]
cost = [3, 4, 1, 10, 1]

→ Station idx
Surplus amt of Gas

① total Gas < total Cost ⇒ -1
21 < 19 ✗ (Ans possible)

② start=0; currGas=0

```
for(i=0 to n)
{ currGas += (gas[i] - cost[i])
  if(currGas < 0)
  { start = i+1
    currGas = 0
  }
}
return start;
```

(i+1) ← start = ?
currGas = ?

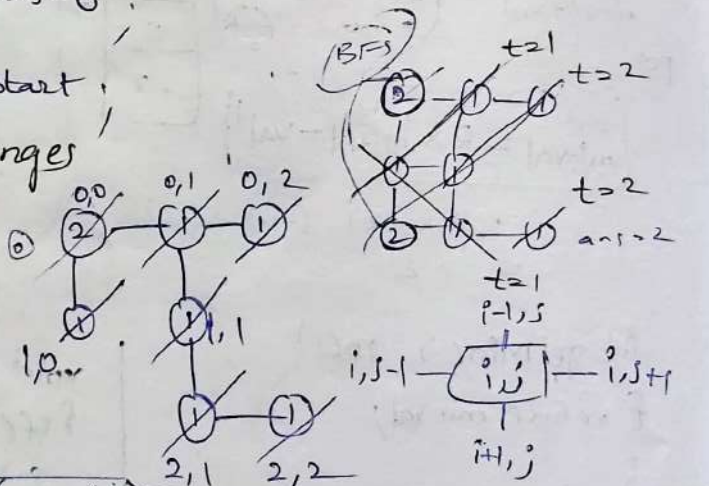
TC: O(n)
SC: O(1)

currGas = currGas + gas[i] - cost[i]

g = [2,3,4], c = [3,4,3]
9 < 10 ✓ (not possible)

13. - Rotting Oranges

rotten	2	1	1
fresh	1	1	0
empty	0	0	1



q: (0,0,0) (0,1,1) (1,0,1) (0,2,2) (1,1,2) (2,1,3) (2,2,4)
(i,j,time)

ans = 0 1 2 3 4

```
q<pair<pair<int,int>,int>>
for(i=0 to n)
  for(j=0 to m)
    if(grid[i][j]==2)
      q.push({{i,j},0})
      vis[i][j] = true

while(q.size()>0)
{ i,j,time = q.pop()
  ans = max(ans, time)
  if(i-1>=0 && !vis[i-1][j] && grid[i-1][j]==1)
  { q.push({{i-1,j},t+1})
    vis[i-1][j] = true
  }
  if(j+1<m) right
  if(i+1<n) bottom
  if(j-1>=0) left
}

//check for fresh oranges
for(i=0 to n)
 {for(j=0 to m)
  { if(grid[i][j]==1
    && !vis[i][j])
    { return -1
  }}}
return ans;
}
```

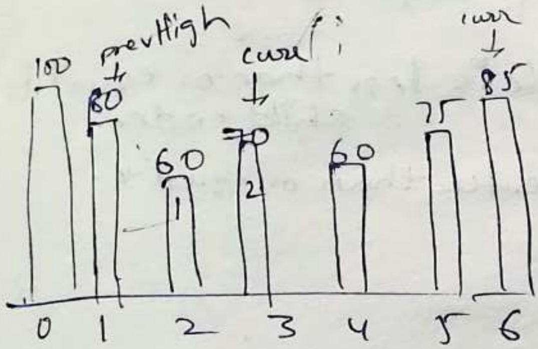
14) Stock Span (hackerEarth)

Span = max no. of consecutive days from today

price <= today's price

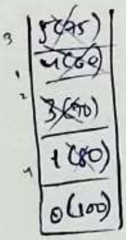
include all day price

price = [100, 80, 60, 70, 60, 75, 85]



i=0 1
i=1 1
i=2 1
i=3 2
i=4 1
i=5 4
i=6 6

prevHigh → Immediate greater value for any day



stack → prev values greater values for today c(idx)

$Sp_{an} = i - pH$

$= 4 - 3 = 1$

$= 5 - 1 = 4$

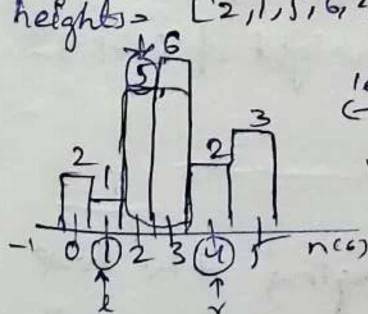
$= 6 - 0 = 6$

[1, 1, 1, 2, 1, 4, 6]

```
vector<int> ans;
stack<int> s;
for (i=0; i<n; i++)
{ while (s.size>0 && price[s.top()] <= price[i])
  { s.pop();
  }
  if (s.size()==0) => ans[i] = i+1
  else => i - s.top   (PH)
} s.push(i)
return ans
```

15) Largest Rectangle in Histogram

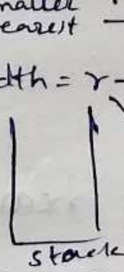
heights = [2, 1, 5, 6, 2, 3]



(w = 4 - 1 - 1 = 2
h = 5
area = 5 × 2 = 10)

left smaller ← nearest   right sm →

width = r - l - 1



right[i] left[i]

stack: smaller vals from ht[i]

① right smaller nearest → next Greater Smaller
② left " " → prev small

right = | 1 | -1 | 4 | 4 | -1 | -1 | (idx)

left smaller = | -1 | -1 | 1 | 2 | 1 | 4 |

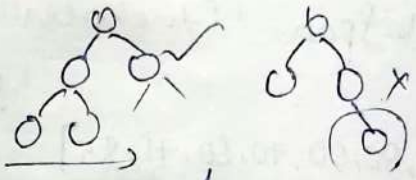
```
ans = 0
for (i=0 to n)
{ currArea = height[i] × (r-l-1)
  ans = max(currArea, ans)
}
return ans
```

right smaller

```
for (i=n-1 to 0)
{ while s && h[s.top()] >= h[i])
  { s.pop();
  }
  r[i] = s.empty() ? n : s.top();
} s.push(i);
while (!s.empty()){s.pop()}   // left smaller  for(i to n)
```

13/12/25

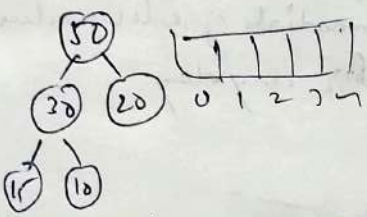
Heaps



Heaps
+
Complete Binary Tree + Heap property

(1) min Heap → value of Node is less than or equal to of child nodes.

(2) max heap → " is greater than or equal "



Convert array into tree nodes:

A) If parent Node is at i^{th} index
   - ↳ left child would be at $2*i+1$;
   - ↳ right " " " " $2*i+2$;

B) If child node is at i^{th} index:
   - ↳ parent will be at $\left[\frac{N}{2}\right]-1$; (ceil)

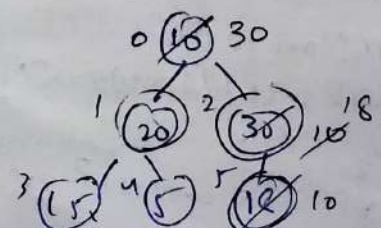
C) If height of a tree is 'h', max nodes in a complete binary tree will be $(2^{h+1})-1$ [h always start from 0]

D) If there are N nodes, then height of Comp. B.T is $\lfloor Log_2 N \rfloor$

Heap operations

→ Heapify → The process of rearranging the heap by recursively comparing a parent node with its child node.

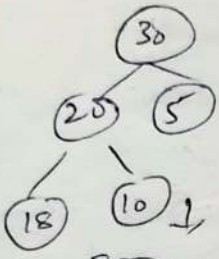
↳ Heapify given tree for the index 0 (maxheap)

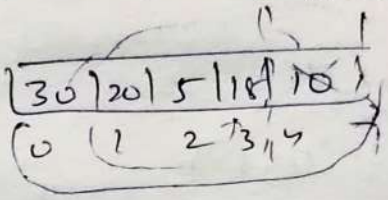


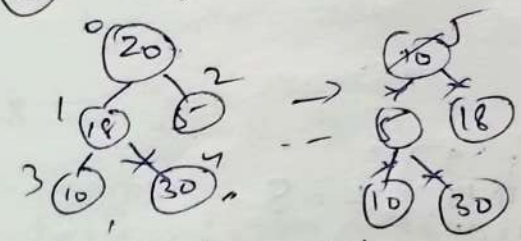
→ 10 [5 [20, 6, 11]]
0 1 2 3 4
5/2 = 2

1 Heaps
2 Heapify
3 HeapSort
4 min Heap
5 max Heap
6 Increase key
7 Decrease key
8 Insert in Heap
9 Delete from heap
10 complete Tree

→ HeapSort → use heap property to sort an array
↳ Build heap
↳ use extract max (store at the end)

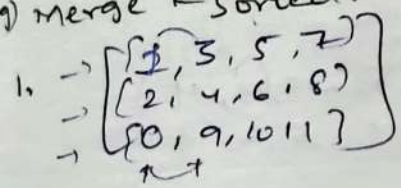


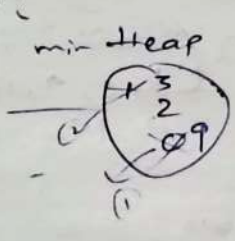




5, 10, 18, 20, 30

(gfg) Merge K sorted Arrays:

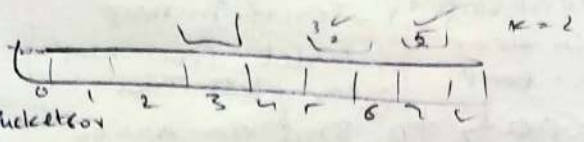





```
import heapq
def mergeArrays(mat):
    n = len(mat)
    heap = []
    res = []
    for i in range(n):
        if mat[i]:
            heapq.heappush(heap, (mat[i][0], i, 0))
    while heap:
        val, ri, ci = heapq.heappop(heap)
        res.append(val)
        if ci+1 < len(mat[ri]):
            heapq.heappush(heap, (mat[ri][ci+1], ri, ci+1))
    return res
```

(347)

2. k most frequent element:



- heap
- max /
- min /
- bucket

3. Heap Sort

- Heapify nums = [5, 2, 3, 1]

while num:
pop()

/ mergesort

4) Kth Smallest element

① min heap /

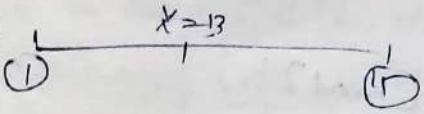
② Binary search

$$mat = \begin{bmatrix} 1 & 5 & 9 \\ 10 & 11 & 13 \\ 12 & 13 & 15 \end{bmatrix} \quad k = 8$$

min for k: [1, 5, 9, 10, 11, 12, 13, 13, 15]
0 1 2 3 4 5 6 7 8

$$\begin{bmatrix} x < k-1 \\ >> k-1 \end{bmatrix} \quad \begin{matrix} m \text{ R} \uparrow & s = m+1 \\ m \text{ L} \downarrow & E = m-1 \end{matrix}$$

8 12 $8 + \frac{(15-8)}{2} = 11$



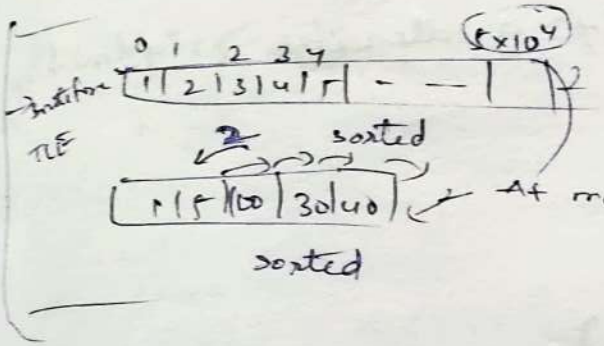
[x < k
x => k]

	T	S
min heap	O(k log n)	O(n)
Binary search	O(n log range)	O(1)

5) (295) Find Median from Data Stream

~~arr = [2, 3, 4]~~ n

1, 3, 25 ← addnum & findMedian (ordered 11, 1)



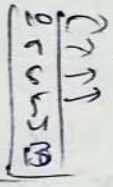
even = $(n/2 + (n/2 - 1))/2$

odd = arr $[n/2]$ ⇒ arr

At most 5×10^4 calls will be made to addnum & findmedi

→ Heap ← (Need DS that can keep your data sorted, you don't need to sort fi...)

3 4 5 8 9 10 n=6



(8+5)/2 = 6.5

maxheap
min heap

(n log n)

⇒ optimal approach

3 4 5 | 8 9 10 , n = 6

3, 4, 5 | 8, 9, nor

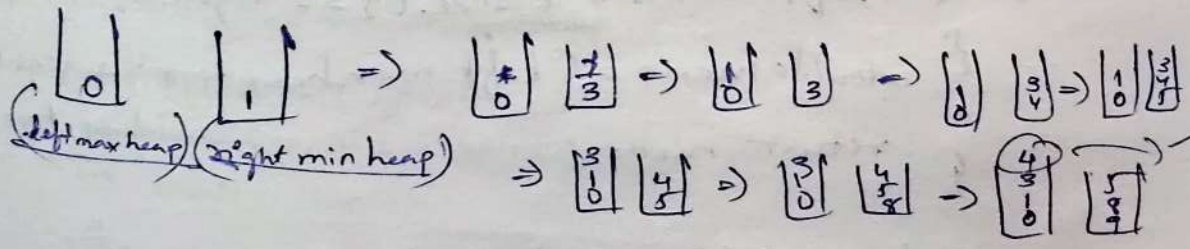
(maxheap left max heap)

(min heap right max heap)

(5+8)/2 = 6.5

any num

0 1 3 4 5 8 9 , n=7



```
class MedianFinder {
public:
    priority_queue<int> left_max_heap;
    priority_queue, vector<int>, greater<int>> right_min_heap;
    MedianFinder()
    {
    }
    void addNum(int num)
    { if (left_max_heap.empty() || num < left_max_heap.top())
      { left_max_heap.push(num);
      }
      else
      { right_min_heap.push(num);
      }
      if(abs(int(left_max_heap.size()) - int(right_max_heap.size())) > 1)
      { right_min_heap.push(left_max_heap.top());
        left_max_heap.pop();
      }
      else if(left_max_heap.size() < right_min_heap.size())
      { left_max_heap.push(right_min_heap.top());
        right_min_heap.pop();
    }
    double findMedian()
    { if (left_max_heap.size() == right_min_heap.size())
      { double mean = (left_max_heap.top() + right_min_heap.top())/2;
        return mean;
      }
      return left_max_heap.top();
    }
};
```

6) (632) Smallest Range covering Elements from k lists

(Naive approach)

[{4,10,15,24,26}, {0,9,12,20}, {5,18,22,30}]

[0,4,5,9,10,12,15,18,20,22,24,26,30]

(0,4), (0,5)

(4,5) x (4,9)

(a,b)

(20,22) → 20 21 22 x, (20,24) → 20,21,22,23,24

24-20=4

(20,24) → 4

Improving –

(The lists are sorted → we didn't use any benefit out of it)

min max

[4 0 5]

(a,b) ⇒ (0,5)

{4 10 15 24 26},
{0 9 12 20},
{5 18 22 30}

0 1 2

| i | j | k |

5-0=5

[10, 15]

[15, 18]

[18, 20]

[20, 24]

if '5' is decreased 3rd arr we can't find any element next) so increase min

T.C = O(n×k) S.C = O(k)

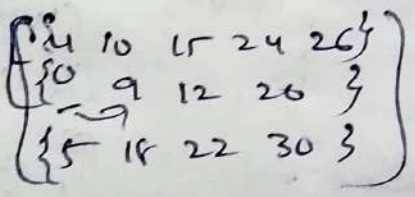
minEle =
minEleIdx =
maxEle =

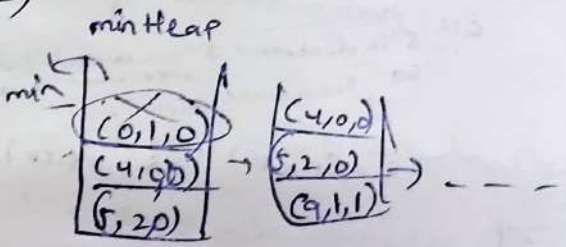
SmallestRange (nums)

{
int k = nums.size();
vector<int> vec(k,0); // {0,0,0...}
vector<int> resultRange = {-10^5, 10^5}; // [a,b]
int minEle = INT_MAX;
int maxEl = INT_MIN;
int minEleListIdx = 0;
for(int i=0; i<k; i++)
{
int listIdx = i;
int elIdx = vec[i];
int element = nums[listIdx][elIdx];

```
if (element < minEL)
{ minEL = element
  minELListIdx = lstIdx;
}
maxEl = max (maxEL, element);
}
// range
if ((maxEl - minEl) < (resultRange[1] - resultRange[0]))
{ resultRange[0] = minEL;
  resultRange[1] = maxEL;
  // shorten this range by moving minEl idx
  int nextIdx = vec[minListIndex] + 1;
  if (nextIndex >= nums[minEl[ListIndex].size())
  { break;
  }
  vec[minEl[ListIndex] = nextIndex;
}
return resultRange;
}
```

Heap - T.C = O(n x log k)






```
maxEl = 5
minEl = 0
listIdx = 1
idx = 0
((idx+1) < nums[lstIdx].size())
{ nextEL = nums[lstIdx][idx++];
  PQ.push (nextEL, lstIdx, idx+1);
  maxEl = max(maxEL, nextEl);
}
```

14/04/2025

Tries

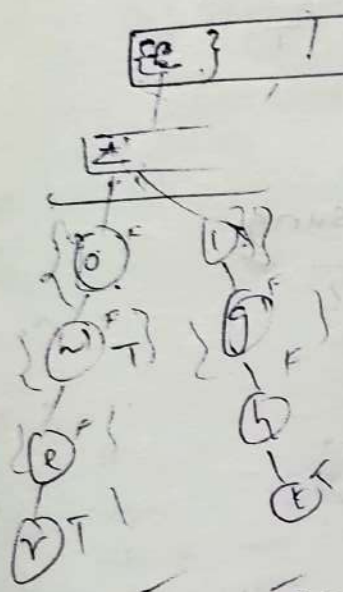
1. (14) Longest common prefix

strs = ["flower", "flow", "flight"]

- strs.sort()
- compareStrs[0] [-1]
- strs[0] strs[-1]
- flight flower — flower

ans = fl

Trade {
self.data = {}
& self.eow = False
}

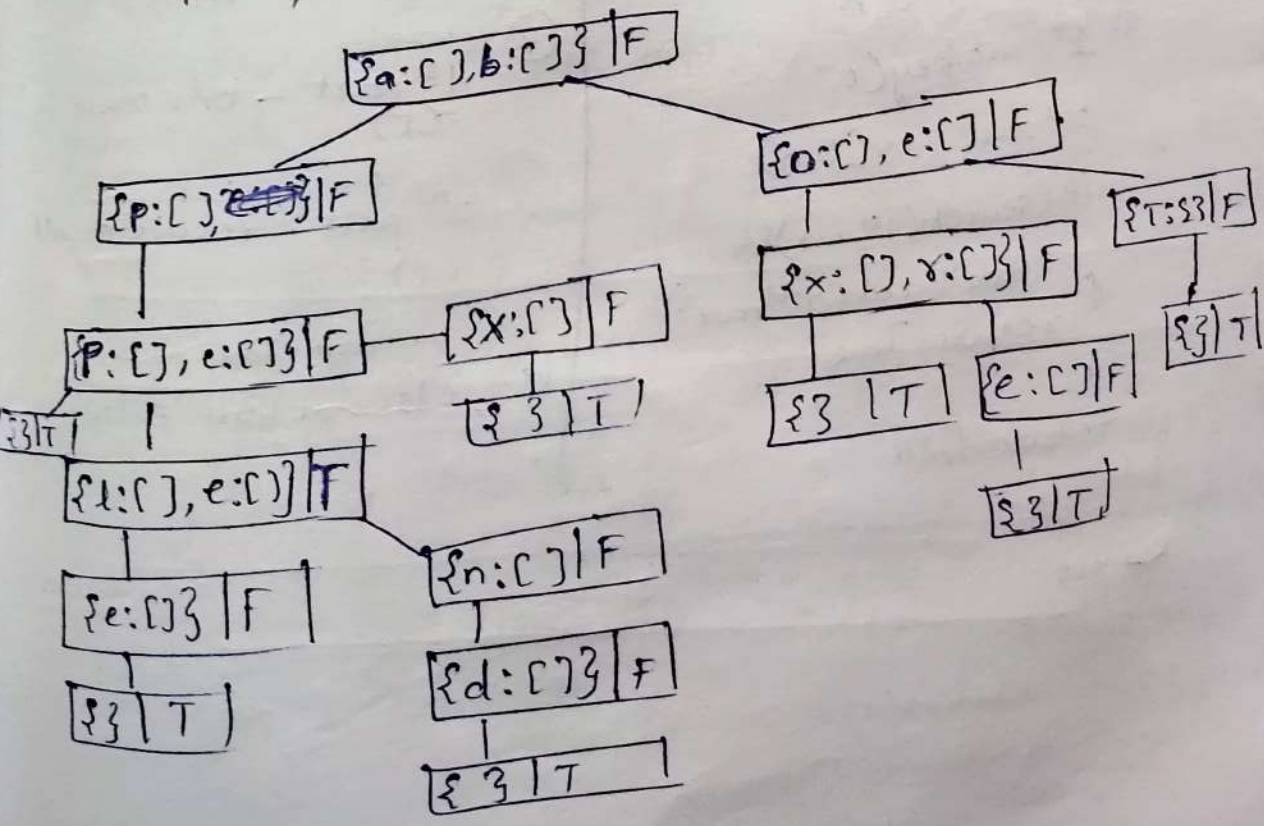


(dict) self.data

(boolean) self.next

{ch: {}} | F

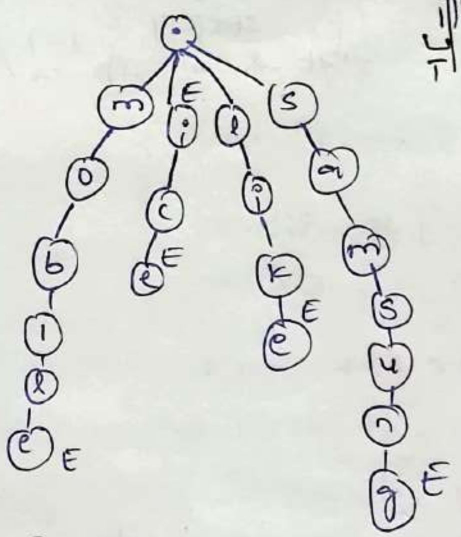
apple, app, apex, bet, append, ape, box, bore



2. word Break problem

dict' word = {i, like, sam, samsung, mobile, ice} (All possible substring pieces)

key = "ilikeSamsung"



i|likeSamsung → (T)

T

like|Samsung

T

Sam|sung   samsung

T   (T)

sung

F

i|likeSamsung

exists - Trie   Trie ? (true)

(Logic)

```
bool wordBreak(string key)   " key="appl". " "
                                  1st part  2nd
{ if(key.length == 0)
  { return True;
  }
  for(int i=1 to n)
  { 1st part=key(0,i)
    2nd part key(i,n)
    if(search(1st part)&&
         wordBreak(2nd part))
    { return True
  } }
  return false
}
```

↳ TLE

use memo={} also

trick: 1)
memo={}

if(memo[s])
{ if s in memo
return memo[s] }

1. all possible parts string
2. loop
   1st part - trie search (T)
   +
   2nd part - recursive call (T)
   +
   return True
3. else return False.

3. phone directory

Insert all contacts into Trie
  For each char in contact
    move/create child node
  mark end of word

For each character in search-string's
  Extend prefix
  If prefix exists in Trie
    Traverse to that node
      collect all words below it using DFS
  ELSE
    store ["0"] (no match)

Return all prefix results.

---

4 Implement a Trie:
- Insert
- search
- startsWith (prefix)

root.child[i]
↙ !null
↘ EOW = True

(1) temp = char
(2) compare temp & ans
(3) recursive child

(char) i + 'a'
i + 'a' → 'b'

---

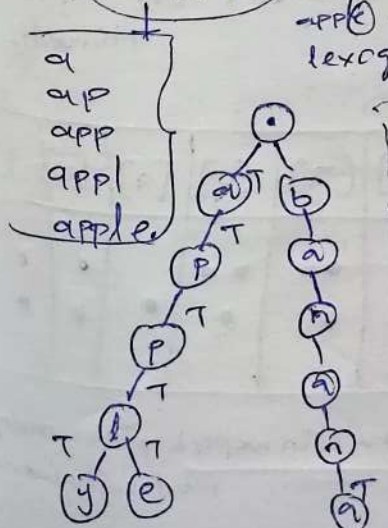
5. Longest word with all prefixes.
(node end = True)

words = ["a", "banana", "app", "appl", "ap", "apply", "apple"]

ans = "apple"

a
ap
app
appl
apple

app(3) appl(4)
lexographic (ascending)




```
longestWord (root, stringBuilder temp)
static String ans = "";
if (root == null)
{ return; }
{ for (int i=0 to 26)
  { if (root.children[i] != null &&
      root.child[i].eow = T)
    { temp.append (cchar)(i+'a'))
      if (temp.length() > ans.length())
      { ans = temp
      }
      longestWord (root.child[i], temp)
      temp.deleteCharAt (temp.length()-1)
  }
}
```

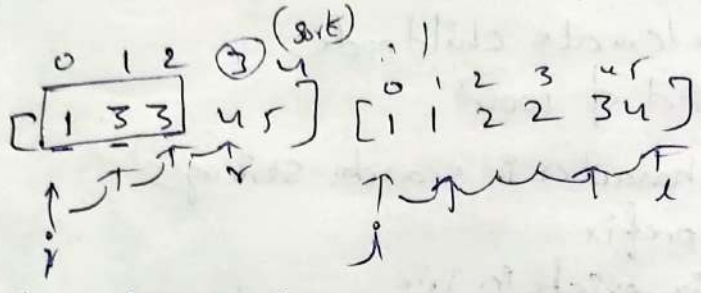
2012

Greedy

1. (455) Assign cookies

greed = [1 5 3 3 4] s = [4 2 1 2 1 3]

O(nlogn + mlogm + m)



return i g[i] <= s[j] 1 <= 1 ✓ ans = 3

3 <= 3 ✓ can satisfy 3 child

3 <= 4 ✓

2. (322) Coin change

(Greedy) — (Always Pick the largest coin ≤ remaining amt. Repeat)

① ② ⑤

amt = 11

⑤ ⑤ ① — −5 ⇒ 6, −5 ⇒ 1, −1 ⇒ 0

ans = 3 coins ✓

① ⑤ ⑥ ⑨

amt = 11

⑨ −9 ⇒ 2, ① −1 ⇒ 1, ① −1 ⇒ 0 — 3 coins

−6 ⇒ 5, −5 ⇒ 0 — 2 coins ✓

This demands dp for optimality

DP

amt = 11

[① ⑤ ⑥ ⑨]

amt	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
Min coins needed		•	• •	• • •	• • •	•	•	• •	• • •	•	• •	• •

{ amt to make = 3, coin choice = ① , remainder = 2 } ⇒ ① + 2 = 3

① represents the coin we pick + "how many more coins we will need"

{ amount to make = 5, Coin Choice = ①, remainder = 4 } ⇒ ① + 4 = 5 (1 coin)

⑤ ⇒ 1 = coin, 0

5 ⇒ 1 ↳ (1) ✓

amt to make = 8
coin choice = ① + 2 = 3 | ⑤ + 3 = 4 | ⑥ + 2 = 3
remainder = 7 | 3 | 2

amt to make = 9
coin choice = ① + 3 = 4 | ⑤ + 4 = 5 | ⑥ + 3 = 4 | ⑨
remainder = 8 | 4 | 3 | 0

amt = 11
= ① + 2 = 3 | ⑤ + 1 = 2 | ⑥ + 1 = 2 | ⑨ + 2 = 3
6 | 5 | 2

coins = [1, 2, 5] amt = 11

coin, amount
O(m*n) - TC
O(n) - SC

0	1	2	3	4	5	6	7	8	9	10	11
[0]	[1]	[2]									[11]

```
int coinChange(int[] coins, int amt)
{ if(amt < 1) return 0;
  // create DP arr
  int[] minCoinsDP = new int[amt+1];
  for(int i=1; i<= amt; i++)
  { minCoinsDP[i] = Integer.MAX_VALUE;
    // Try each coin
    for(int coin : coins)
    { if(coin <= i && minCoinsDP[i-coin] != Integer.MAX_VALUE)
      { minCoins[i] = Math.min(CoinsDP[i], 1 + minCoinsDP[i-coin]);
      }
    }
  }
  if(minCoinsDP[amt] == Integer.MAX_VALUE)
      return -1;
  return minCoinsDP[amt];
}
```

3. Fractional Knapsack

w = 15
n = 7

problem

objects — 1 2 3 4 5 6 7
profit — 5 10 15 7 8 9 4
weight — 1 3 5 4 1 3 2
P/w — 5 3.3 3 1.75 8 3 2 (Max prof/wt ratio)

maxprof order (I) 47.25
(II) min weight order 46
(III) profit/weight order 51

Objects	profit (p)	weight (w)	Remaining wt
5	8	1	15 − 1 = 14
1	5	1	14 − 1 = 13
2	10	3	13 − 3 = 10
3	15	5	10 − 5 = 5
6	9	3	5 − 3 = 2
7	4	2	2 − 2 = 0
	51		

Greedy method

(0/1 → DP knapsack)

gcd = 15, 4
gcd(9, 6)
gcd(6, 9 % 6)

pairs.sort(key = lambda x: x[1]) (sort by second value)

4. max length of pair chain (same as Job scheduling without overlapping)

5. Activity selection (gfg)

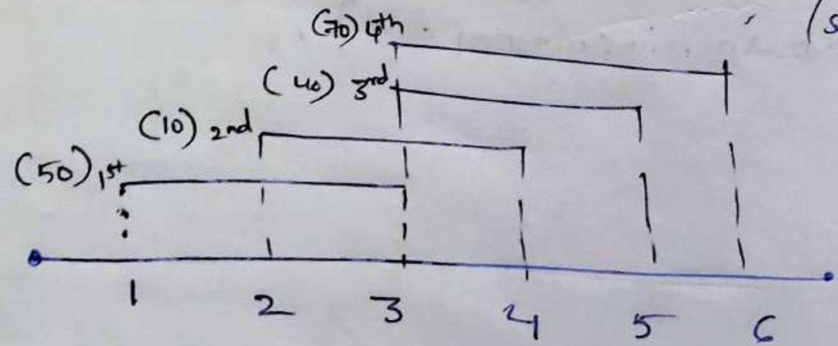
(1 2) (3 5) (6, 8)
(2 < 3)

activities = sorted(zip(start, finish), key = lambda x: x[1])
start, finish = zip(*activities)

6. (1235) Maximum Profit in Job Scheduling

recursion memo
dp + binarysearch

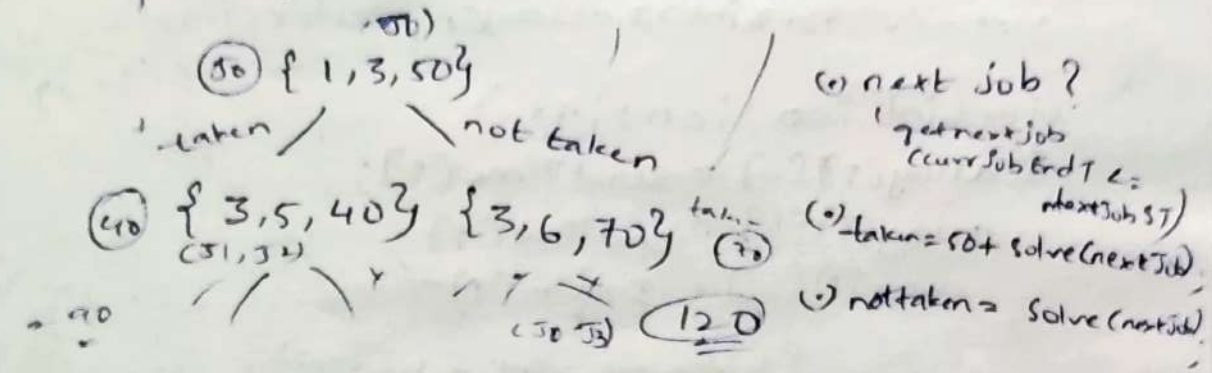
(subset shouldn't overlap)



50 + 40 = 90 | 50 + 70 = 120 | 10 | 40 | 70

{s, e, p} -> // sort based on 's'

{{1, 30, 50}, {2, 4, 10}, {3, 5, 40}, {5, 6, 70}}
0 1 2 3




```
class Solution
{public:                    Recursion || memorization || BS
   int n; int memo[50001];
   // find 1st job - starting point >= current job end point
   int getNextIndex (vector<vector<int>>& array, int l,
                                    int currentJobEnd)
   { int r = n-1;
     int result = n+1;
     while (l <= r)
     { int mid = l+(r-1)/2;
       if (array[mid][0] >= currentJobEnd) // we can make
                                           this's task
       { result = mid;
         r = mid-1;
       }else {
         l = mid+1
     } }
     return result;
}
int solve(vector<vector<int>> &array, int i)
{  if (i>=n)
      return 0;
   if (memo[i] != -1)
      return memo[i];
   int next = getNextIndex(array, next, array[i][1]);
   int taken = array[i][2] + solve(array, next);
                (profit)
   int notTaken = solve(array, i+1);
   return memo[i] = max(taken, notTaken);
}
```

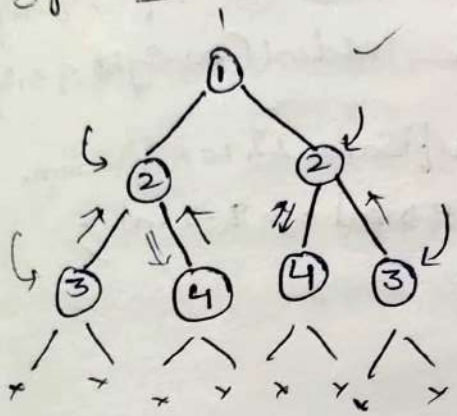
```
int jobScheduling(vector<int& startTime, vector<int>& endTime,
                  vector<int>& profit)
{   n = startTime.size()
    vector<vector<int>> array(n, vector<int>(3,0));
                                        // [s,e,p]
    for(int i=0; i<n; i++)
    { array[i][0] = startTime[i];
      array[i][1] = endTime[i];
    } array[i][2] = profit[i];
    auto comp = [&](auto& vec1, auto& vec2)
    { return vec1[0] < vec2[0];
    }                    (sort according to start time)
    sort(begin(array), end(array), comp)
     return solve(array,0);
}
};
```

25/12/25

Binary Trees

1. } Inorder, preorder, postorder
3. }

4. Symmetric Tree

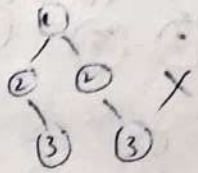


Mirror

Left ⇄ Right

(Root → Left)	(Root → Right)
Root Left Right	Root Right Left

T.C



```
bool isSymmetric (TreeNode* root)
{ return root==NULL || isSymmetricHelp(root->left, root->right);
}

bool isSymmetricHelp(TreeNode* left, TreeNode right)
{ if (left == null || right== null)
    return left==right;
  if (left.val != right.val) return false
  return (isSymmetricHelp(left.left, right.right)
&& (isSymmetricHelp (left.right, right.left)
}
```

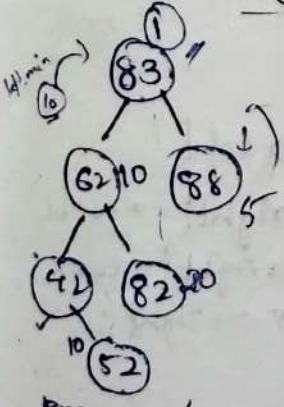
5. Min Distance b/w BST nodes

BST : sorted → Inorder

left subtree

root ->prev = root

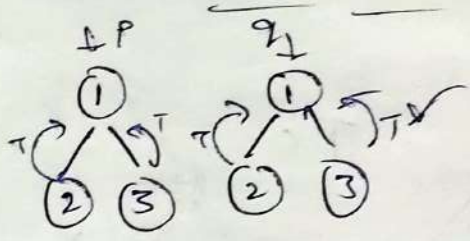
right subtree

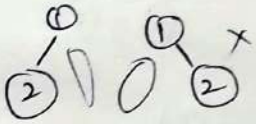



```
TreeNode* prev = NULL
int minDist (root)
{ if (root== NULL)
    return INT_MAX;
  ans= INT_MAX;
  if (root->left)
  { left_min = minDist(root->left)   -①
  } ans = min(ans, left_min);
  if (prev != NULL)
    ans = min(ans, root->prev)       -②
  prev = root;
  if (root->right)
  { right_min = minDist (root->right)  -③
    ans = min(ans, right_min);
  }
```

42
52
62
82
83
88
ret ans;

6. (100) Same Tree (Identical Tree)



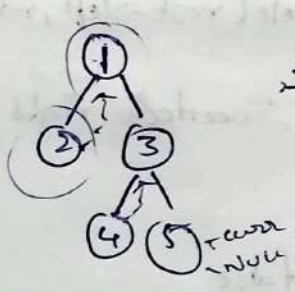


p	q
NULL	① → false
①	NULL → false
NULL	NULL → True

```
bool IsIdentical(P,Q)
{ if(P==NULL || Q==NULL)
    return P==Q
  IsLeftSame = IsIdent(P->left, Q->left)
  IsRightSame = IsIdent(P->right, Q->right)
  return IsLeftSame && IsRightSame
         && P->val == Q->val;
}
```

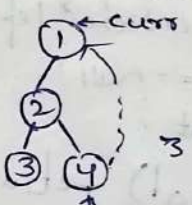
7. Morris Inorder Traversal

recursive ×
iteration ✓



2, 1, 4, 3, 5

InorderPredecessor: (IP) (storing)
↳ right most node in left subtree



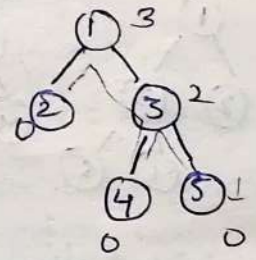

```
While (curr != NULL)
  if (curr->left == NULL)
    // Print curr
    curr = curr->right
  else
    find InorderPredecessor(IP)
    if (IP->right == NULL)
      IP->right = curr // create thread
      curr = curr->left
    else
      IP->right = NULL // delete thread
      // Print curr // res.append(curr.val)
      curr = curr->right
```

```
IP = curr->left
while(IP->right != NULL
      && IP->right != curr)
{ IP = IP->right;
}
```

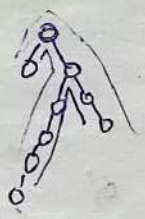
8. Diameter of Binary Tree

ans = 0, 3;

```
int height(root)
{ if (root == null) return 0;

  leftHt = ht (root->left) ;
  rightHt = ht (root->right)

  ans = max(lH+rH, ans)   // extra int func for diameter
  return (max(LH, RH)+1 ;
}
```

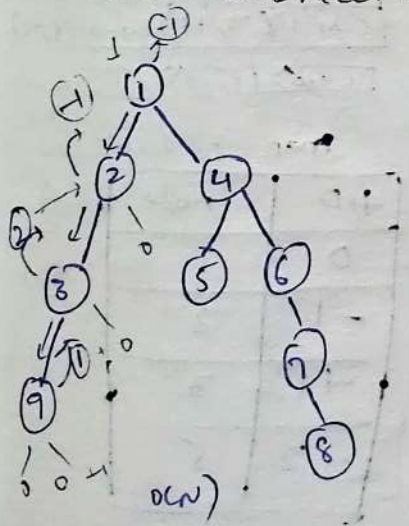


height (root)
return ans



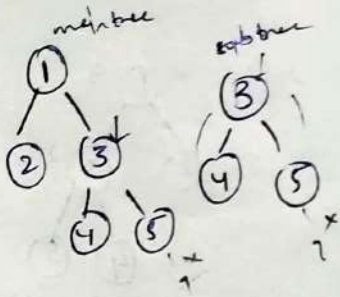
node
① diam through root
len = lh+rh
② rightDiam
③ leftDiam

9. (110) Balanced Binary Tree




```
int check(node)
{ if (node == null)
      return 0

  lh = check (node->left)
  rh = check (node->right)
  if(lh==-1 or rh==-1) ret -1;

  if(abs(lh-rh)>1) ret -1;
  return max(lh, rh)+1;
}
```

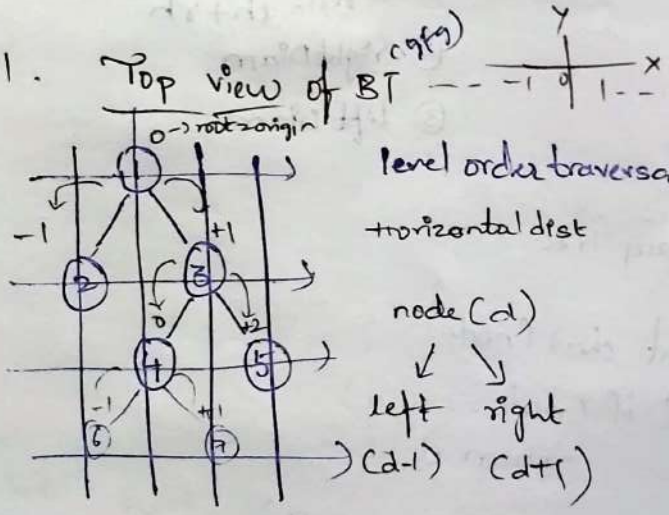
10) (572) Subtree Tree of another Tree



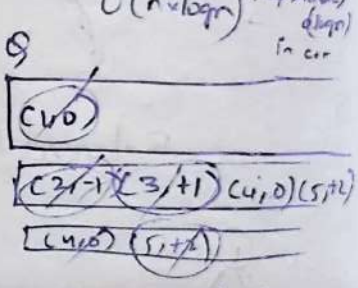

```
bool isSubTree(root, subRoot)
{ if(root==NULL || subRoot==NULL)
      return root==subRoot
  if(root->val==subRoot->val &&
      isIdentical(root, subRoot))
      return true;
  return isSubtree(root->left, subRoot)
      || isSubtree(root->right, subRoot);
}
```

```
bool isIdentical(TreeNode* p, TreeNode* q)
{ if(p==NULL || q==NULL)
  { return p==q; }
  return p->val==q->val
  && isIdentical(p->left, q->left)
  && isIdentical(p->right, q->right)
}
```

11. Top view of BT



O(n log n) map→find() O(log n) in cur



map (ordered)

HD	node
0	1
-1	2
+1	3
+2	5

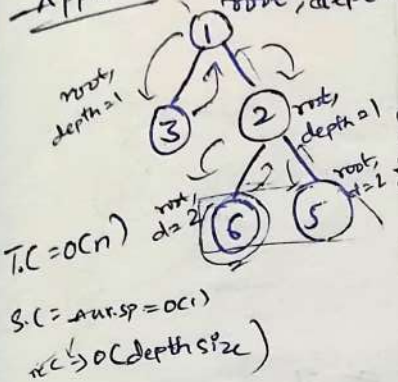
(-1 0 +1 +2)
2, 1, 3, 5

```
void topView(Node* root)
{ queue<pair<Node*, int>> q;
  map<int, int> m;   // HD, node val
  while(q.size()>0){
      Node* curr = q.front().first;
      int currHD = q.front().second;
      q.pop();
      if(m.find(currHD)==m.end()){
      }  m[currHD] = curr->data;
      if(curr->left!=NULL){
      }  q.push({curr->left, currHD-1});
       if(curr->right!=NULL){
  } } }  q.push({curr->right, currHD+1});
```

```
for(auto it:m)
{ cout<<it.second<<" ";
}
```

12) (513) Find bottom-left Tree value

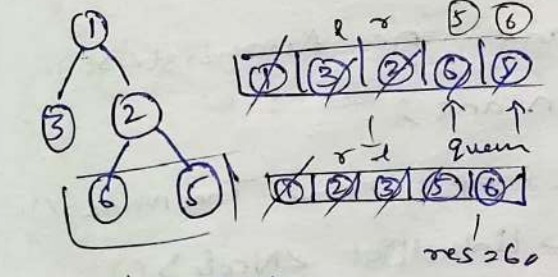
Approach 1: (DFS)




```
int maxDepth;
int bottomLeft;
void dfs (TreeNode* root, int curDepth)
{ if (!root) {return;}
  if(curDepth > maxDepth)
  { maxDepth = currDepth;
    bottomleft = root->val;
  }
  dfs (root->left, curDepth+1);
  dfs (root->right, currDepth+1);
}
dfs(root, 0);
return bottomleft;
```

maxDepth = -1 ✓ 2
if (depth > maxdepth)
{ maxdepth = depth;
result = 7;
3
6
=
}

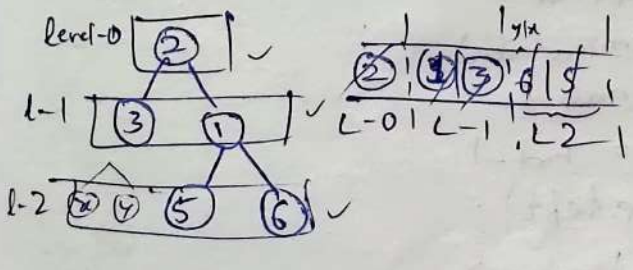
Approach-2: (BFS) normal



① root → right
② root → left

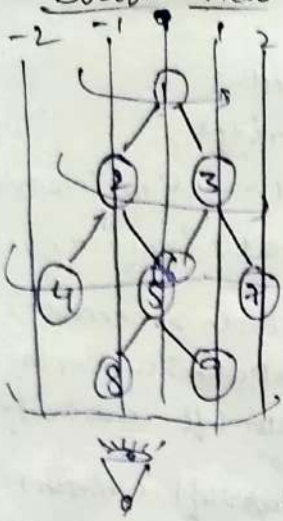
```
while (!que.empty())
{ node = que.front(); que.pop();
  res = node->val;
  if (node->right)
    que.push (node->right);
  if (node->left)
    que.push (node->left);
```

Approach-3: (Level By Level processing)



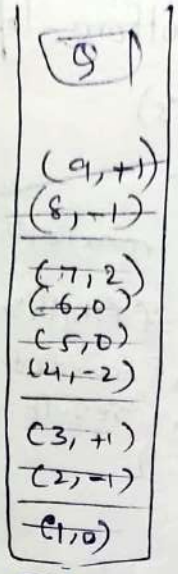

```
{ int bottomLeft;
  que.push(root);
  while (!que.empty())
  { int n = que.size();
    while (n--)
    { TreeNode* node = que.front();
      que.pop();
      bottomLeft = node->val;
      if(node->right)
        que.push(node->right)
      if (node->left)
        que.push(node->left)
    }
  }
}
```

Bottom view of B.T



4, 8, 6, 9, 7

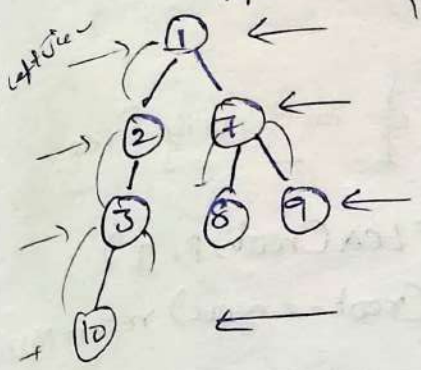
(dist, node) map
2 → 7
-2 → 4
1 → 3 9
-1 → 2 8
0 → 1 6



node = 1 2 3 4 5 6 7 8 9

```
{ bottomview(node root)
  { ArrayList<Integer> ans = new ArrayList<>();
    If (root == NULL) return ans;
    Map<Integer, Integer> map = new TreeMap<>();
    Queue<Node> q = new LinkedList<Node>();
    root.hd = 0;
    q.add(root);
    while (!q.isEmpty())
    { Node temp = q.remove();
      int hd = temp.hd
      map.put(hd, temp.data);
      if(temp.left != null)
      { temp.left.hd = hd-1;
      } q.add(temp.left);
      if(temp.right != null)
      { temp.right.hd = hd+1;
      } } q.add(temp.right);
    for(Map.Entry<Integer, Integers entry: map.entrySet())
    { ans.add(entry.getValue());
    }
    ret ans;
} }
```

Right view



{1 7 9 10}

```
List<Integer> rightsideview (TreeNode root)
{ List<Integer> res = new ArrayList<>();
  if(root == NULL)
  { return res;
  }
  Queue<TreeNode> queue = new LinkedList<>();
  queue.offer (root);
  //bfs
  while (!queue.isEmpty ())
  { int size = queue.size();
    while (size > 0)
    { TreeNode node = queue.poll ();
      if (node.left != null)
      { queue.offer (node.left);
      }
      if (node.right != null)
      { queue.offer (node.right);
      }
      size--;
      if (size == 0)
      { res.add (node.val) }
    }
  }
  return res;
```

Left view

```
if (size > 0) {
  Node node = queue.peek();   (will not delete)
  res.add (node.data);
}
```

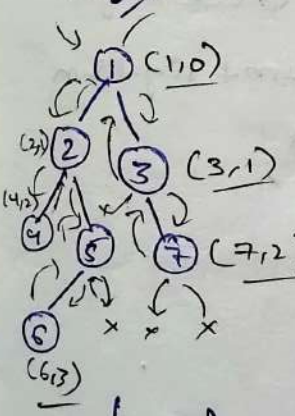
TC → O(n)

SC → O(m) worst case

Left view

```
if (size > 0) {
  Node node = queue.poll();   (will delete)
  res.add (node.data);
}
```

Right view / left




```
f (node, level, ds)
{ if (node == null)
    return;
  if (level == ds.size)
    ds.add (node)
  if (node.right, level+1);
  if (node.left, level+1);
}
```

root, 0, ds

RLR → RRL

T.C → O(n)

SC → O(H)

rec stack

for left side view → left

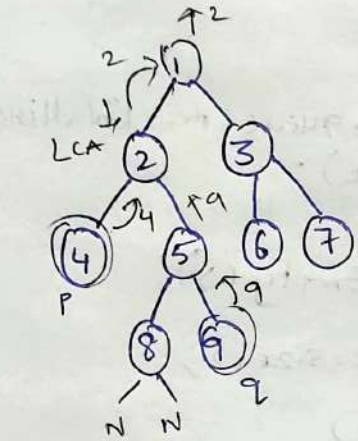
for right → right

ds: 6, 7, 3, 1 (indices 3, 2, 1, 0)

14) (102) level order traversal

15) (1611) Maximum Level Sum of a BT

16) (236). Lowest Common Ancestor of a Binary Tree

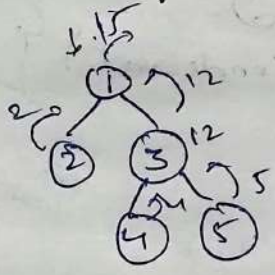


LCA → 1st node P,Q → Subtrees exist

```
Node* LCA(root, P, Q)
{ if(root == NULL) return NULL;
  if(root == P || root == Q)
      return root; // LCA
  leftLCA = LCA(root->left, P, Q)
  rightLCA = LCA(root->right, P, Q)
  if(leftLCA && rightLCA)
      return root;
  else if (leftLCA != NULL)
      return leftLCA
  else
      return rightLCA
```

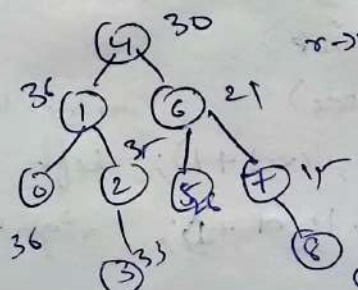
L=N R=N

17) Transform to SumTree



```
int SumTree(root)
{ if (root == NULL) return 0;
  leftSum = SumTree(root->left)
  rightSum = SumTree(root->right)
  root->data += leftSum + rightSum
  return root->data;
}
```

(1038) BST to GST (greater Sum Tree)



```
r->rt->l  bstToGst(root)
{ self.total = 0
  def dfs(node):
      if not node: return
      dfs(node.right)
      self.total += node.val
      dfs(node.left)
  dfs(root)
```

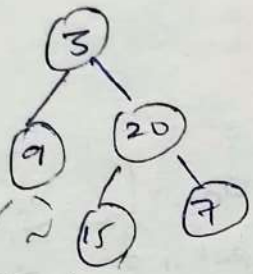
MB. (105) Build Tree from preorder & Inorder

preorder = [3, 9, 20, 15, 7] inorder = [9, 3, 15, 20, 7]

{ root, left, right } preIdx { l, rt, r } idx

left : l to inIdx − 1

right : inIdx + 1 to r



buildTree (& preorder [], & inorder [], & preIdx, left, right)

{ Node* root = preorder [preIdx]; if (left > right) return NULL;

preIdx ++

inIdx = search (inorder, left, right, preorder [preIdx])

root->left = BT (preorder, inorder, preIdx, left, inIdx − 1)

root->right = BT (preorder, inorder, preIdx, inIdx + 1, right)

return root;

}

return BT (preorder, inorder, preIdx, 0, inorder.size() − 1);

int search (vector<int>& inorder, int left, int right, int val)

{ for (int i = left; i <= right; i++)

{ if (inorder[i] == val)

{ return i;

} } }

(preIdx, left, right)

(0, 0, 4)

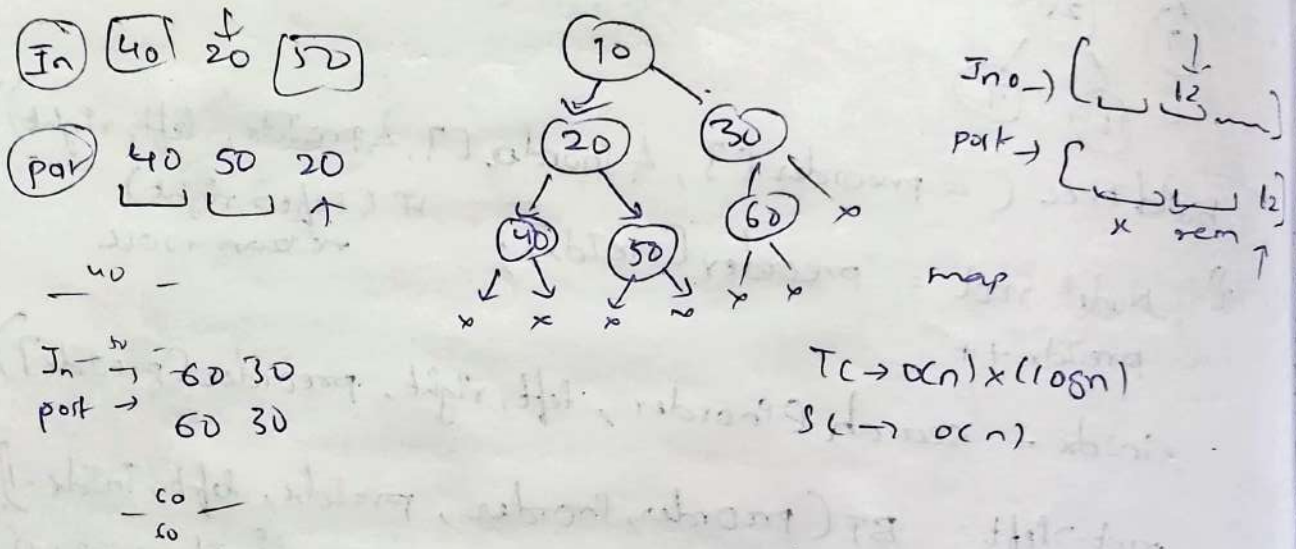
(1, 0, 0) (2, 2, 4)

NULL NULL (3, 2, 2) (4, 4, 4)

(19) (106) Build tree from Inorder & Postorder

Inorder → [40 20 50 10 60 30]
(Left Root Right) 0 1 2 3 4 5

Postorder → [40 50 20 60 30 10]
(Left Right Root)



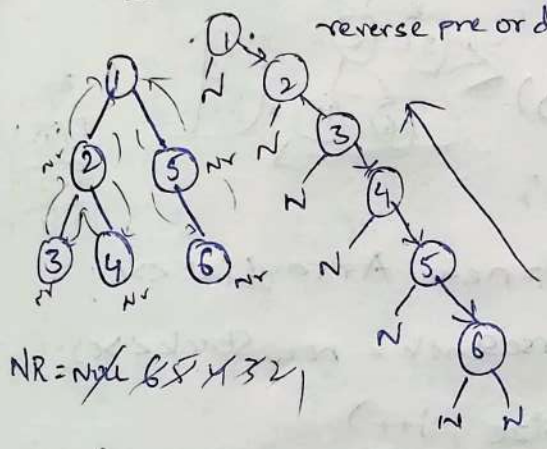

```
buildTree_Node (inorder, postorder)
{ if (inorder.size() != postorder.size())
      return NULL
  map<int,int> hm;
  for (i=0 → inorder.size())
      hm[inorder[i]] = i;
  return buildTreePostIn (inorder, 0, inorder.size()-1,
                          postorder, 0, postorder.size()-1, hm);
}
buildTreePostIn (inorder, is, ie, postorder, ps, pe, hm)
{ if (ps > pe || is > ie) return NULL;
  root = TreeNode (postorder[pe]);
  int inRoot = hm[postorder[pe]];
  int numLeft = inRoot - is;
  root->left = buildTreePostIn (inorder, is, inRoot-1,
                                postorder, ps, ps+numsLeft-1, hm);
```

root->right = buildTreeFromPostIn(preorder, preRoot+1, ie, postorder, ps+numsleft, pe-1, hm);

return root;

};

20) Flatten Binary Tree to LinkedList



reverse pre order Rt l r → r l Rt

Node* nextRight // last visited node (NR)

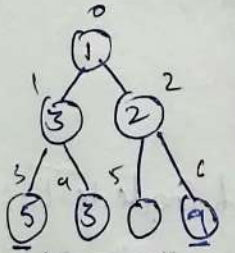
```
void flatten(root)
{ flatten(root->right)
  flatten(root->left)
  root->left = NULL
  root->right = NR
  NR = root
}
```

Tc: O(n)

NR = Null 65 432 1

21) (562) Maximum width of Binary Tree

(longest length blw the end-nodes where the null nodes blw the end-nodes that would be present in a complete BT (CBT))



CBT index

1	3	2	5	6		9
0	1	2	3	4	5	6

(root)

Par, i

left: 2*i+1 right: 2*i+2

level = (endIdx - stIdx) + 1

= width

(1,0)

f (3,1)(2,2) r

(5,3) (3,4) (9,6)

6 — 3 —

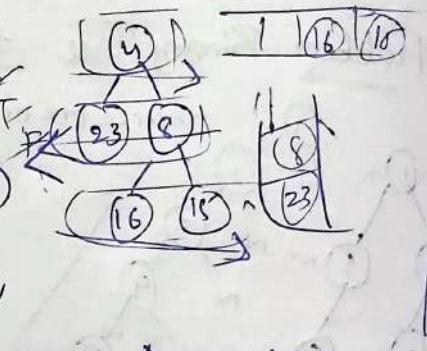
```
int maxWid(root)
{ Queue<pair<Node*, int>> q
  q.push(root, 0);
  maxWid = 0
  while(q.size()>0)
  { stIdx = q.front().second;
    endIdx = q.back().second;
    maxWid = max(maxWid, endIdx-stIdx+1);
    for(i=0 to curr.levSize)
    { currNode = q.pop() → idx
      if(left) → (node->left, 2*idx+1)
} } { if(right) → (node->right, 2*idx+2)
```

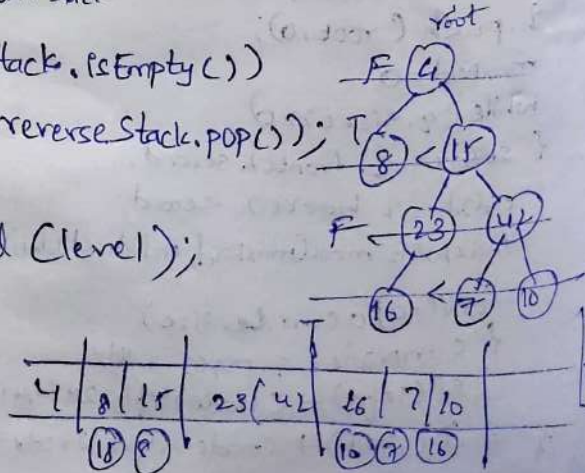
maxWidth = 4

O(n)

103. Zigzag Traversal O(n)

```
List<List<Integer>> zigzag = new ArrayList<>();
if (root == null) return zigzag;
Queue<TreeNode> queue = new LinkedList<>();
queue.add(root);
boolean flag = false;
while (!queue.isEmpty())
{ int size = queue.size();
  List<Integer> level = new ArrayList<>();
  Stack<Integer> reverseStack = new Stack<>();
  for (int i=0 ; i< size ; i++)
  { TreeNode node = queue.poll();
    if(flag)
      reverseStack.add (node.val);
    else
      level.add (node.val);
    if (node.left != null ) queue.add (node.left);
  } if (node.right != null) queue.add (node.right);
  flag = !flag;
  // pop all ele from stack
  while (!reverseStack.isEmpty())
    level.add (reverseStack.pop());
  ~~flag~~
  zigzag.add (level);
return zigzag;
}
```





23rd (124) Maximum Path Sum in Binary Tree

15+7+20 = 42

9+(-10)+35 = 34

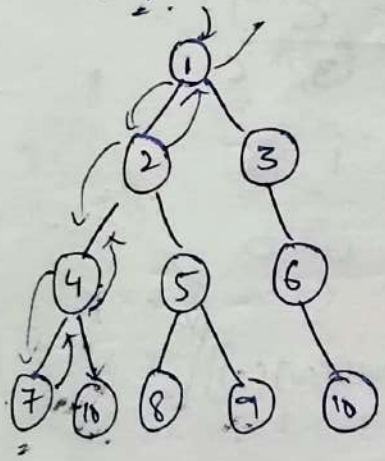
```
Max = -∞ or 42
int maxPath (node, maxi)
{ if (node == null)
    return 0;
  leftSum = maxPath (node->left)
  rightSum = maxPath (node->right,.)
  maxi = max(maxi, leftsum + rightsum
                + node->val)
  return (node->val) + max(leftSum,
                rightsum);
}
```

24) (1483) Kth Ancestor of a Tree Node

(kth ancestor of the given node in BT)

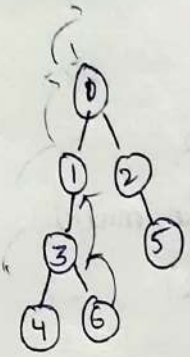
T k = 4 3 2 1 0

val = 7.



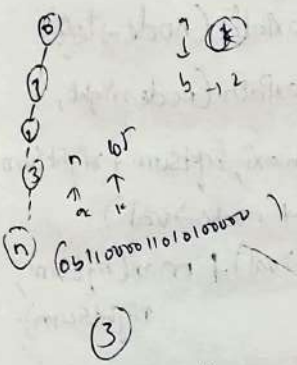

```
Node* helper (Node* root, int& val, int& k)
{ if(!root) return;
  Node* rv = NULL;
  if (root->data == val)
  or (rv = helper(root->left, val, k))
  or (rv = helper(root->right, val, k))
  { if (k==0)
    { cout << "ans:" << root->data << endl;
    } return NULL;
    k--;
  } return rv ? rv : root;
} return NULL;
void getKthAncestor(Node* root, int val, int k)
{ Node* parent = helper(root, val, k);
  if (parent)
  { cout << "-1" << endl;
  }
}
```

K^{th} Ancestor | Binary lifting



(1) Parent [0 0 0 1 3 2 3]
0 1 2 3 4 5 6

par of i = P[i]



(2)

	0	1	2	3	4	5	6
0	0	0	0	1	3	2	3
1	0	0	0	0	1	0	1
2	0	0	0	0	0	0	0
3							
10							

(3)

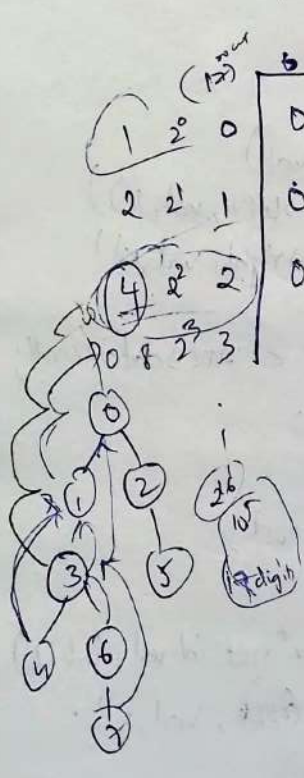
k = 1 0 0 1

2^3 2^2 2^1 2^0

8 4 2 1

11 → 1 0 1 1

8 + 2 + 1 → 11



	0	1	2	3	4	5	6	7
0	0	0	0	1	3	2	3	6
1	0	0	0	0	1	0	1	3
2	0	0	0	0	0	0	0	0

table[i][j] = table[i-1][table[i-1][j]]

(2nd parent)

(11) 3 2 1 0

1 0 1 1

2^3 2^1 2^0

3 1 0 ← pow

1<<1 → 10

8 → 2^3

$\log_2(n)+1$

a → 8 3 0

k → 3 → 1 1

```
query(a, k)
{ for(i=0; i<max; i++)
    mask = (1<<i);
    if(k & mask)>0,
    { a = table[i][a];
} } }
```

max span

log n - time

```
Class Main
{ static int MAX = 17;
  static int table [][];
  static void build (int P[])
  { int n = P.length;
    table = new int[MAX][n];
    for (int i=0; i<n; i++)
    { table[0][i] = P[i];
    }
    for (int i=1; i<MAX; i++)
    { table[i][j] = table[i-1][table[i-1][j]];
    }
  }
  Static int query (int a, int k)
  { for (int i=0; i< MAX; i++)
    { int mask = (1<<i)
      if (k & mask) > 0)
      { a = table[i][a];
  } } }

void main( args) —
{ ..
  n
  parent [ ?
  q
  build (parent);
  while (q-- > 0)
  { String inp[] = read.readline() : split(" ");
    int a = Integer parseInt (read. readline());
    int q = Integer.parseInt (read. readline());
} }
```

28/12

Binary Search Trees

1. (703) k^{th} Largest Element in a Stream.

[[3, [4,5,8,2]], [3], [5], [10], [9], [4]]

k=3 → [4,5,8,2]

[2,3,4,5,8] → 4

[2,3,4,5,5,8] → 5

[2,3,4,4,5,5,8,9,10] → 8

Brute force
{ [4,5,8,2,7] → Sort
arr.get(n-k) }

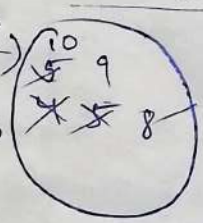
k>3

n, n+1, n+2 ...

opt → heap

add [3 5 10 9 4] → 4 5 5 8 8

Minheap



k=3

kvalue (insert)

int k)

s=0 1 2 3

2<4 ×

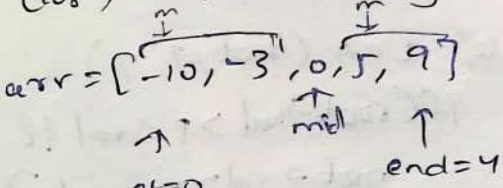
T.C → O(k) + 10^4 log k

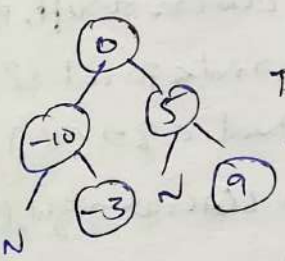
→ O(k) + 10^4 log k

S.C → O(k)

```
public kth largest(k, nums[]) //constructor
{ this.k=k
  pq = new priorityQueue<>();   //minheap
  for(int num: nums)            (3) peek
  { add(num);
} }
public int add(int val)
{ if(pq.size()<k || val> pq.peek())
  { pq.offer(val);
    if(pq.size()>k)
    { pq.poll();
} } }
```

2. (108) Sorted Array to Balanced BST



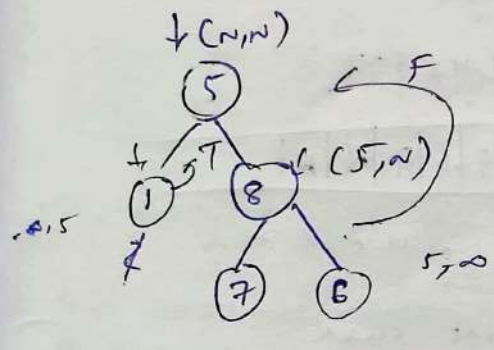


T.C O(n)
S.C logn (rec stack)

left < par < right

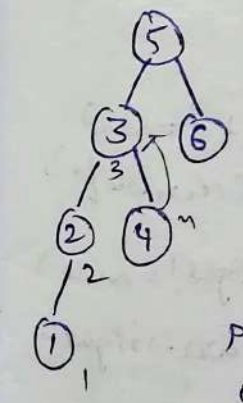
```
Node* helper(nums[], st, end)
{ mid = st + (end-st)/2
  root = newNode(nums[mid])
  root->left = helper(nums, st, mid-1)
  root->right = helper(nums, mid+1, end)
  return root;
}
```

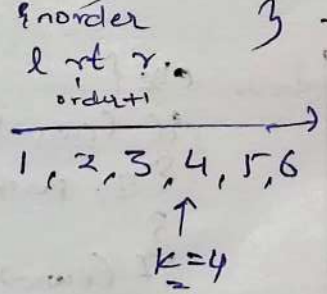
3. (148) Validate BST




```
bool isBST(root, Node* min, max)
{ if(root==NULL) return true;
  if(min!=NULL && root->val<=min->val)
      return false;
  if(max!=NULL && root->val>=max->val)
      return false
  return isBST(root->left, min, root)
  && isBST(root->right, root, max)
}
```

4. k^{th} Smallest in BST





prev

order = 0 1 2 3

1, 2, 3, 4, 5, 6

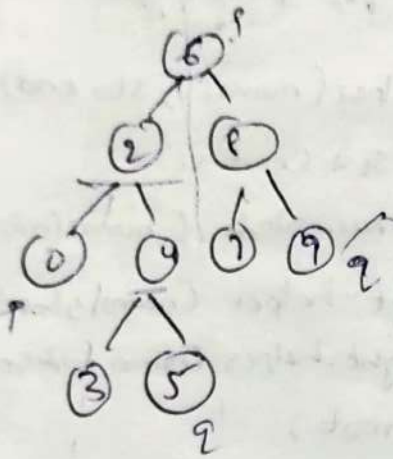
(left) (root) (right)

order = 3

order = k

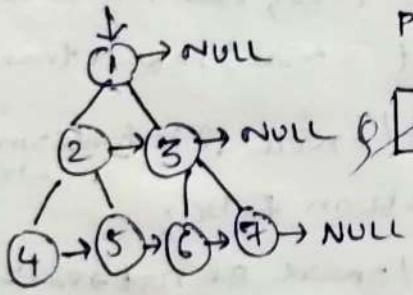
```
kthSmallest(root, k)
{ if(root==NULL)
    return -1
  left: if(root->left)
  { leftAns = kthSmallest(root->left, k)
    if(leftAns!=-1)
      return leftAns
  }
  root: if(preOrder+1==k)
      return root->val;
    preOrder = preOrder+1;
  right: if(root->right)
  { rightAns = kthSmallest(root->right, k)
    if(rightAns!=-1)
      return rightAns
  }
}
```

5. Lowest Common Ancestor (LCA) (235)




```
Node* LCA (root, p, q)
  if(root->val > p->val &&
     root->val > q->val)
    return LCA(root->left, p, q);
  elif (root->val < p->val &&
        r->val < q->val)
    return LCA(root->right, p, q)
  else -> return root
```

6. (116) populate Next Right Pointers



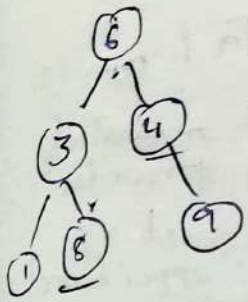
prev = N 1 N 2 3 N 4 5 6 7 N #

curr = N 1 2 3 4 5 6 7 N

```
{ if prev != N
    prev->next = curr
  prev = curr
```

```
Node* connect (Node* root)
{ if (root == NULL || root->left == NULL)
  { return root;
  }
  queue<Node*> q;
  q.push(root);
  q.push(NULL); // mark end of level
  Node* prev = NULL;
  while (q.size()>0)
  { Node* curr = q.front();
    q.pop();
    if (curr == NULL)
    { if (q.size() == 0){
        break;
      q.push(NULL);
    }
    else {
    if (curr->left != NULL)
    { q.push(curr->left);
    }
    if (curr->right != NULL)
    { q.push(curr->right);
    }
    if (prev != NULL)
    { prev->next = curr;
    }}
    prev = curr;
  }
  return root;
}
```

(99) Recover BST



(Sorted) 1, 3, 8, 6, 4, 9 — inorder { prev < next

(Compare) 4 8

8,6 6,4

F S

first = 8

sec = 6 4

prev = 1 3 8 6 4 9

TC - O(n)

SC - O(n)

```
prev = NULL; first = NULL; sec = NULL;
Inorder(root) {
  If (root == NULL) -> return.
  inorder(root->left)
  if (prev != NULL && root->val < prev->val)
  { if(!first){
      first = prev;
    }
    sec = root;
  }
  prev = root
  Inorder(root->right)
}
```

```
int tmp = f->val
f->val = s->val
s->val = f->tmp
```

Swap f & s

=> To optimize S.C to O(1) use [Morris Inorder Traversal (Iterative)]

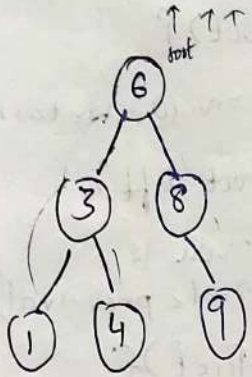
TC O(n)

else S.C O(1)

```
void recoverTree(TreeNode* root)
{ TreeNode* first = NULL;
  TreeNode* sec = NULL;
  TreeNode* prev = NULL;
while (root != NULL)
{ if (root->left == NULL)
  { if (prev != NULL && prev->val > root->val)
    { if (first == NULL)
      { first = prev;
      }
      second = root;
    }
    prev = root;
  } root = root->right;
else
{ TreeNode* IP = root->left;
  while (IP->right != NULL && IP->right != root)
  { IP = IP->right;
  }
  if (IP->right == NULL)
  { IP->right = root;
  } root = root->left;
  if (prev != NULL && prev->val > root->val)
  { if (first == NULL)
    { first = prev;
    }
    second = root;
  }
  prev = root;
  IP->right = NULL;
  root = root->right;
} }
if (first != NULL && second != NULL)
{ tmp = f->val;
  f->val = s->val;
  s->val = tmp;
}
```

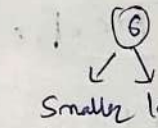
8. (1008) Construct BST from Preorder

preorder = [6, 3, 1, 4, 8, 9]

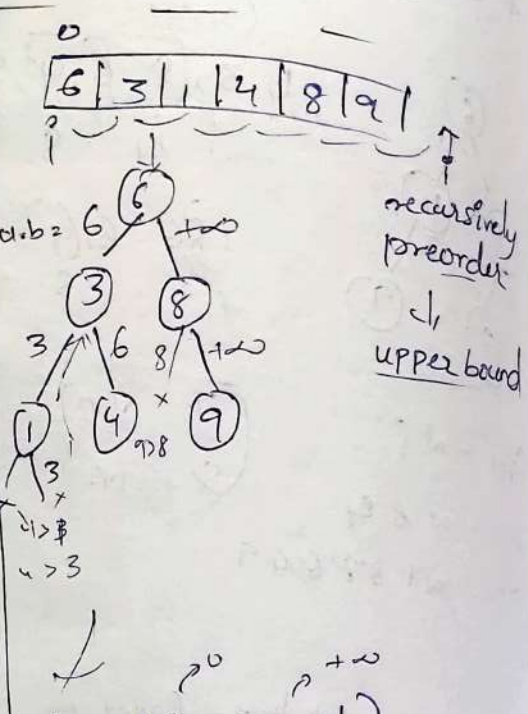


root
left
right

Brute force
O(n * n)

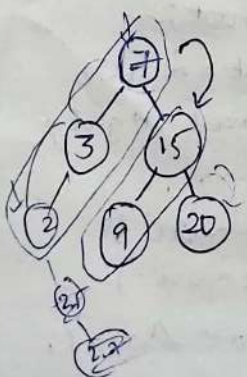


Placing 1 by 1 in tree according to BST




```
Node* buildBST (preorder, &i, bound)
{   if (i >= preSize() || preorder[i] > bound)
        return null;
    Node* root = new Node(preorder[i]);
    i++;
    rt->left = buildBST(preorder, i, root->val)
    rt->right = buildBST (preorder, i, bound)
}   return root;
```

9. (173) BST Iterator

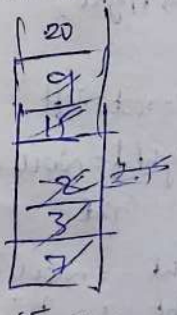


inorder → left, root, right

↳ 5, 7, 9, 15, 20

next() 5
next() 7
next() 9
hasNext() true
next() 15

TC: O(1)
S.C: O(h)



2, 3, 7, 9, 15, 20

① constructor
Store' LeftNodes()

② next()
ans = S.top()
S.pop()
if (ans->right)
storeleftNodes(ans->right)

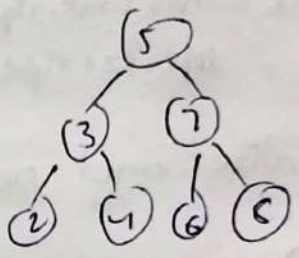
③ hasNext()
return S.size() > 0;

void storeleftNodes(root)
{ while(root != NULL)
{ S.push(root)
root = root->left
}}

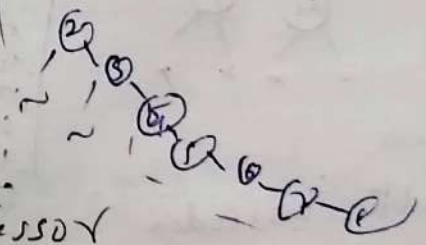
10: Flatten BST to Sorted list

prev = None
head = None
def Inorder(root):
if not root: return N
Inorder(root.left)
if prev:
prev.right = root
prev.left = None
else:
head = root
prev = root
Inorder(root.right)
if prev: prev.left = None
return head

i/p = [5,3,7,2,4,6,8] root

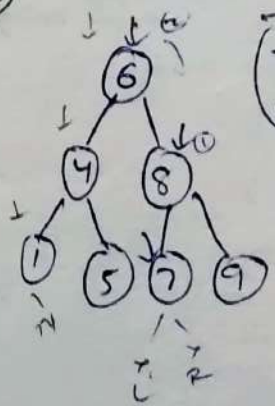


o/p: [2,N,3,N,4,N,5,N,6,N,7,N,8]



11) Inorder Successor & Inorder Predecessor

12)



TC: O(H)
SC: O(1)

key = 7
pred = 6
succ = 8

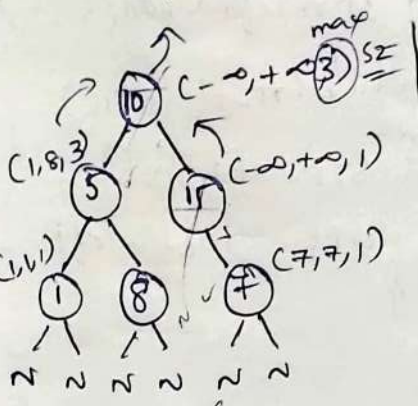
key = 3
pred = 1
succ = 4

① key < curr->data
succ = curr
curr = curr->left

② key > curr->data
pred = curr
curr = curr->right

③ key == curr->data
↳ Inpred = right most in leftsubtree
↳ Insucc = left most in right subtree
break;

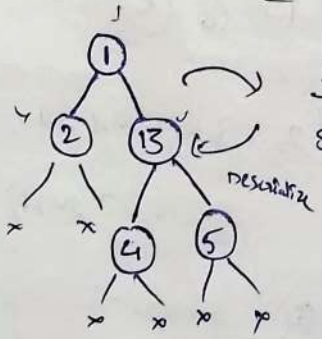
13. Largest BST in B.T




```
helper(root)
  BASECASE → return info(+∞, -∞, 0).
  info left = helper(root->left)
  info right = helper(root->right)
  if(root->data > left.max && root->data < right.min) // valid BST
      return info(CurrMin, CurrMax, CurrSz)
  else
      return (-∞, +∞, max(left.sz, right.sz))

CurrMin = min(root, left.min)
CurrMax = max(root, right.max)
CurrSz = left.sz + right.sz + 1
```

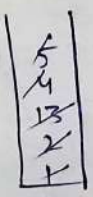
```
class Info
Info(mn, mx, sz)
  this.mn = mn
  this.mx = mx
  this.sz = sz
```

14. Serialize and Deserialize Binary Tree



Serialize → Level order.

String → 1, 2, 13, #, #, 4, 5, #, #, #, #,



9 → empty return root

T.C → O(n)
S.C → O(n)

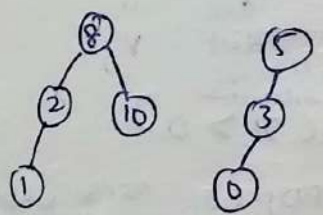
```
Public class Codec
{ public String serialize(TreeNode root).
  { if(root == NULL) return "#";
    Queue<TreeNode> q = new LinkedList<>();
    StringBuilder res = new StringBuilder();
    q.add(root);
    while(!q.isEmpty())
    { TreeNode node = q.poll();
      if(node == null)
      { res.append("#");
      } continue;
```

root → s
string ← d

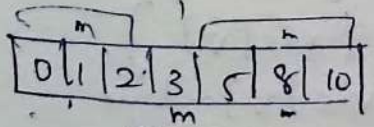
```
        res.append(node.val + " ");
        q.add(node.left);
        q.add(node.right);
    }
    return res.toString();
}
public TreeNode deserialize(String data)
{ if (data == "") return null;
  Queue<TreeNode> q = new LinkedList<>();
  String[] values = data.split(" ");
  TreeNode root = new TreeNode(Integer.parseInt(values[0]));
  q.add(root);
  for (int i = 1; i < values.length; i++)
  { TreeNode parent = q.poll();
    if (!values[i].equals("n"))
    { TreeNode left = new TreeNode(Integer.parseInt(values[i]));
      current.left = left;
      q.add(left);
    }
    if (!values[++i].equals("n"))
    { TreeNode right = new TreeNode(Integer.
      parseInt(values[i]));
      parent.right = right;
      q.add(right);
    } }
  } return root;
```

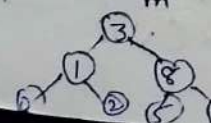
T.C - O(m+n)

4. Merge 2 BSTs



arr1 | 1 | 2 | 8 | 10 | arr2 | 0 | 3 | 5 |



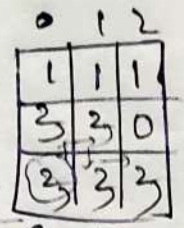



```
node* merge(root1, root2)
{ Inorder(root1, arr1)
  Inorder(root2, arr2)
  vector<int> temp;
  while(i<arr1.size() && j<arr2.size())
  { if(arr[i] < arr[i])
        temp.pb(arr1[i++])
    else
        temp.pb(arr2[j++])
  }
  while(i<arr1.size()) → temp.pb(arr1[i++])
  while(j<arr2.size()) → temp.pb(arr2[j++])
  BuildBST(temp)
  return
```

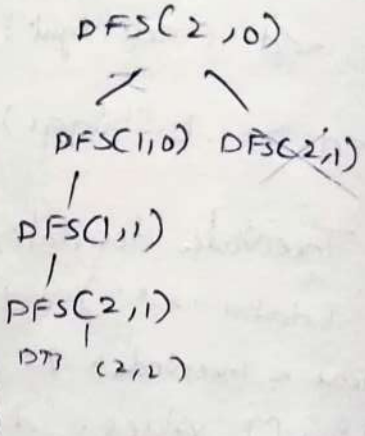
30/12/25

Graphs

1. (733) — Flood fill



newcolor = 3
sr = 2 sc = 0
Initial color = 2



(r-1, c+0)
(r+0, c-1) (row, col) -> (r+0, c+1)
(r+1, c)

Δrow = {-1, 0, +1, 0}
Δ col = {+0, +1, +0, -1}

up

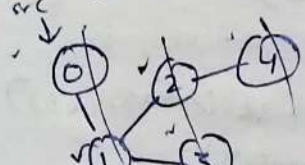
```
Void dfs (int row, int col, vector<vector<int>>& ans, vector<vector<int>>& image,
{   ans[r][c] = newColor;                                  int newColor, int delRow[]
    n = image.size();                                      delCol[]
    m = image[0].size;
    for (i = 0; i < 4; i++)
    {   int nrow = row + delRow[i];
        int ncol = col + delCol[i];
        if (nrow >= 0 && nrow < n && ncol >= 0 && ncol < m
            image[nrow][ncol] == iniColor && ans[nrow][ncol]
            dfs(nrow, ncol, ans, image, newColor, delRow, delCol);
    }
}
```

BFS

O(V+E)

1) travel to immediate neighbours first

V = 5
src → 0
dest → 4



vis

0	1	2	3	4
T/F	T/F	T/F	T/F	T/F

Q

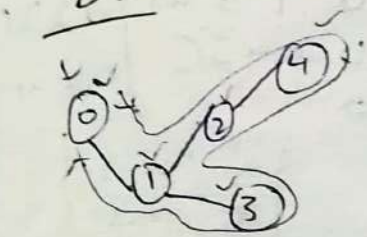
0	1	2	3	4

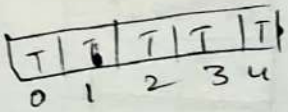
```
while (q.size() > 0)
{   int u = q.pop()   //src
    cout << u;
    for (v : neighbours of u) {
        if ( ! vis[v] )
            ① vis[v] = true
            ② q.push(v)
    }
}
```

u = 0 1 2 3 4 o/p = (0 1 2 3 4)

O(V+E)

DFS



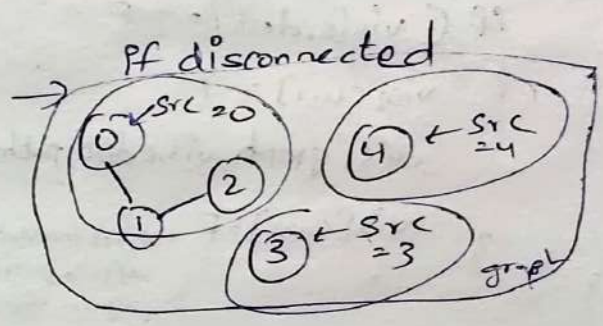


O/P – [0, 1, 2, 4, 3]

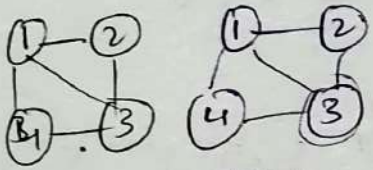
```
dfs(u, vis)
{ cout<<u;
  vis[u] = true
  for(v : l[u])
  { if(!vis[v])
    { dfs(v, vis)
  } } }
```

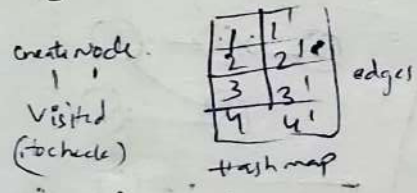
```
for(int i=0; i<v; i++)
{ if(!vis[i])
  { dfsHelper(i, vis);
  }
}
cout<<endl
```

If disconnected



2. (133) Clone Graph (Deep copy - connections same, memory diff)



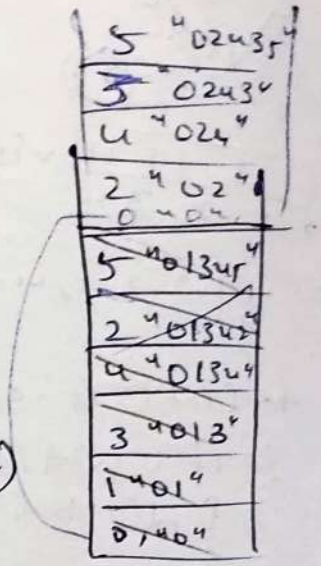



```
Node* cloneUtil(Node* node, mp)
{ Node* newNode = newNode(node->val);
  mp[node] = newNode
  for(auto neighbour : node->neighbours)
  { if(mp.find(neighbor) == mp.end())
    { // Create the neighbor's clone
      (newNode->neighbors).push_back
          (cloneUtil(neighbor, mp));
    }
    else {
      (newNode->neighbors).pushback
                (mp[neighbor]);
  } }
  return newNode;
}

Node* cloneGraph(Node* node)
{ if(!node) return NULL;
  unordered_map<Node*, Node*> mp;   // old-new
); } return cloneUtil(node, mp);
```

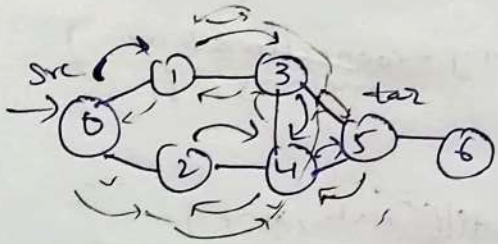
All paths from source to target

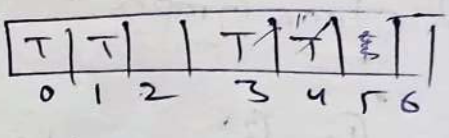
```
dfs(graph, vis[], curr, String path, tar)
{
    if (curr == tar)
        print(path)
        return
    for (int i=0 to graph[curr].size())   // neighbours
    {
        Edge e = graph[curr].get(i)
        if (vis[e.dest] == F
        {
            vis[curr] = T
            dfs(graph, vis, e.dist, path+e.dest, tar)
            vis[curr] = F   // (not mandatory to set like dfs as it can be traversed again for path)
    } } }
```

0 1 3 4 5 | 0 1 3 5
0 2 4 3 5 | 0 2 4 5
0



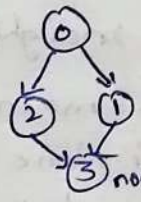
Rec Stack

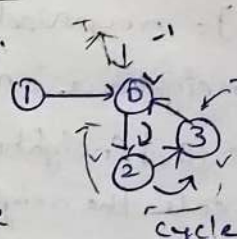


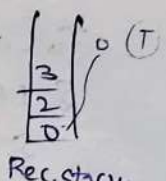


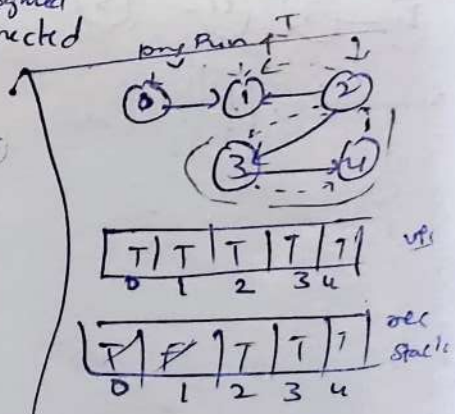
Cycle detection

weighted/unweighted
directed / undirected










```
dfs(graph, vis[], curr, rec[])
{
    vis[curr] = T
    rec[curr] = T
    for(int i=0 to graph[curr].size())
    {
        Edge e = graph[curr].get(i)
        if (rec[e.dest] == T)
            return T                    } cycle
        else if (!vis[e.dest])
            if(dfs(graph, vis, e.dest, rec()))
                return true
    }
    rec[curr] = F
}
```

(directed)

```
for (int i=0; i<V; i++)
{ if(!vis[i])
  { boolean cycle = isCycleDirected
      (graph, vis, curr:0, rec);
    S.o.p(isCycle)
  } break;
}
```

Disjoint set union (DSU) / Union-find

oper^{n} - (1) → Combine two given sets

oper^{n} - (2) → Tell me if two members (b, g) belong to same set or not

A ∩ B ∩ C = NULL

union(a,b), (c,d)

union(a,c); union(e,c)

find(c) → g

find(g) → g

DSU → find, union

parent

Rank

To make height minimum

Union by Rank & path compression

(compression)

(rank)

find(6)

```
find(int i, parent)
{  if (parent[i] == i)
       return i;
   return find(parent[i], parent);
}
```

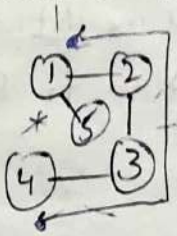
parent[i] = i

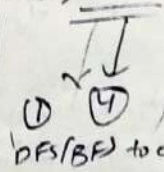
```
void union(int x, int y, vector<int>& parent, vector<int>& rank)
{ int x_parent = find(x, parent);
  int y_parent = find(y, parent);
  if (x_parent == y_parent)
        return
  if (rank[x_parent] > rank[y_parent])
  {
  }   parent[y_parent] = x_parent;
  else if (rank[x_parent] < rank[y_parent])
  {
  }   parent[x_parent] = y_parent
  else
  {   parent[x_parent] = y_parent;
      rank[y_parent] += 1;
  }
```

3. Redundant graph: (684)

edges: [[1,2],[2,3],[3,4],[1,4],[4,5]]

graph -





DFS/BFS to check component connected

(cycle)

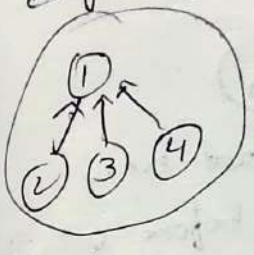
DFS

```
findRedundantConnection( edges)
{  int n = edges.size();
   // no. of nodes = n
   // no. of edges = n
   unordered_map<int, vector<int>> adj;
   // O(n^2)
   // space : O(n)
   for(int i=0; i<n ; i++) // O(n)
   {  int u = edges[i][0];
      int v = edges[i][1];
      vector<bool> visited (n, false);
      if(adj.find(u) != adj.end() && adj.find(v) != adj.end()
         && dfs(adj, u, v, visited)
      {  return edges[i];          // cycle exist
      }
      adj[u].push_back(v);
   } adj[v].push_back(u);
}; return {};
```

```
bool dfs( &adj , u, v, vis)
{  vis[u] = true;
   if(u == v)
   {  return true;
   }
   for(int nghbr : adj[u])
   {  if(vis[nghbr]) continue;
      if(dfs(adj, nghbr, v, vis))
      {  return true;
      }
   }
   return false;
}
```

edges = [(1,2) (2,3) (3,4) (1,4) (1,5)]

both Parents same



edge[i] = (u, v)

```
if (find(u) == find(v))
    return (u -> v);
```

DSU

```
findRedundantConnection(edges)
{
  n = edges.size();
  DSU dsu(n);
  // T.C : DSU = alpha(n)
  // T.C : O(n * alpha(n))
  for (auto &edge : edges) // O(n)
  {
    int u = edge[0]
    int v = edge[1]
    if (dsu.find(u) == dsu.find(v))
    { return edge;
    }
    dsu.Union(u,v);
  }
  return {};
}
```

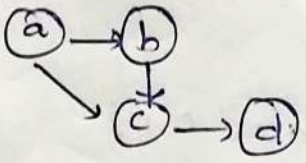
```
class DSU{
  parent;
  rank;
DSU(int n)
{ parent.resize(n+1); // 1,2,3,...n
  rank.resize(n+1);
  for(int i=1; i<=n; i++)
  { parent[i] = i;
    rank[i] = 0;
  }}
int find(int x)
{ if (x == parent[x])
  { return x;
  }
  return parent[x] = find(
                parent[x]);
} // path compression
```

```
void Union(int x, int y)
{ int x_parent = find(x);
  int y_parent = find(y);
  if (x_parent == y_parent)
  { return;
  }
  if (rank[x_parent] > rank[y_parent])
  { parent[y_parent] = x_parent;
  }
  else if (rank[y_parent] > rank[x_parent])
  { parent[x_parent] = y_parent;
  }
  else
  { parent[y_parent] = x_parent; // we need x as parent
  } rank[x_parent]++;
}
```

Topological Sorting

↳ works for DAG (Directed Acyclic Graphs)

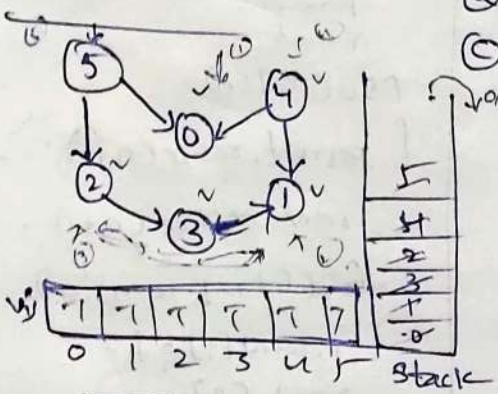
↳ It is a linear order of vertices such that every directed edge u→v, the vertex u comes before v in the order



a→b ①
b→c ②
a→c ③
c→d ④

DAG

u→v



o/p = [5 4 2 3 1 0]

```
topsort (graph, vis, curr, stack)
{ → vis[curr] = T
 for(int i=0 to graph[curr].size())
 { Edge e = graph[curr].get(i)
   if (!vis[e.dest])
   { topsort(graph, vis, e.dest, stack)
   }
 stack.push(curr)
}
```

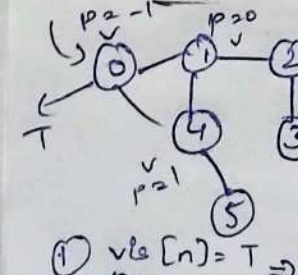
```
for (int i = 0 to v)
    if (vis[i] == F)
        topSort(, ==>
print → Stack.pop()
```

Cycles in Graph

Undirected graph → DFS
→ BFS
→ DSU (Disjoint Set Union)

Directed graph → DFS
→ BFS
→ Topological Sorting
→ graphColoring

Undirected Graph (modified DFS)




```
dfs(graph, vis[], curr, par)
{ vis[curr] = T
 for(int i=0 to graph[curr].size())
 { Edge e = graph[curr].get(i) //e.dest
 ① if (vis[e.dest] = T && par != e.dest)
      return true
 ③ if (!vis[e.dest])
   { if (dfs(graph, vis, e.dest, curr) == T)
        return true
   }
 }
```

① vis[n] = T ⇒ cycle(true) par ✗

② vis[n] = T ⇒ do nothing par ✓

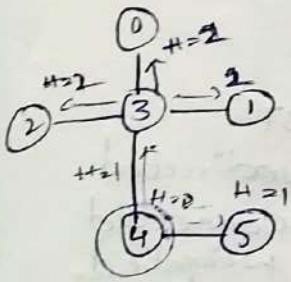
③ vis[n] = F ⇒ visit → true ✓ / false

Minimum height Trees

4.

n=6 edges = [[3,0],[3,1],[3,2],[3,4],[5,4]]

o/p - [3,4]



root = 0 → H = 3

root = 2 → H = 2

root = 4 → H = 2

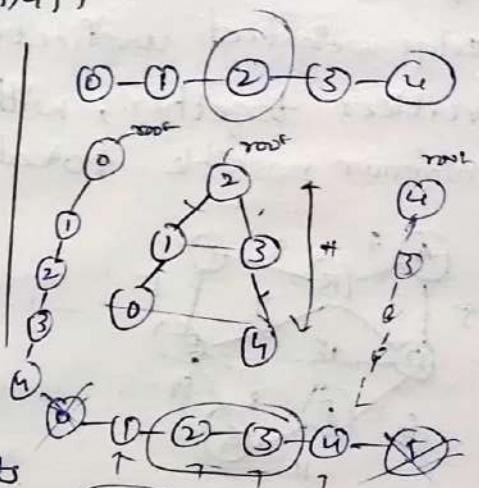
root = 5 → H = 3

BFS

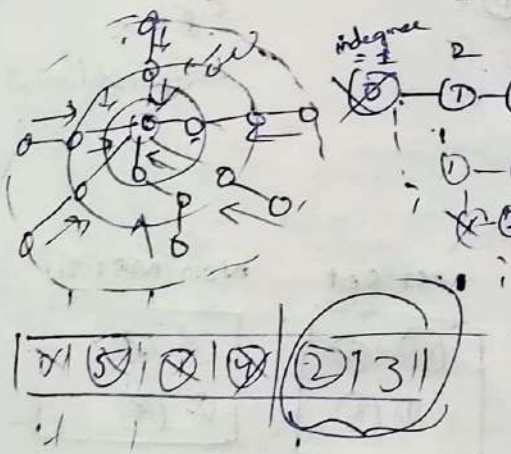
DFS

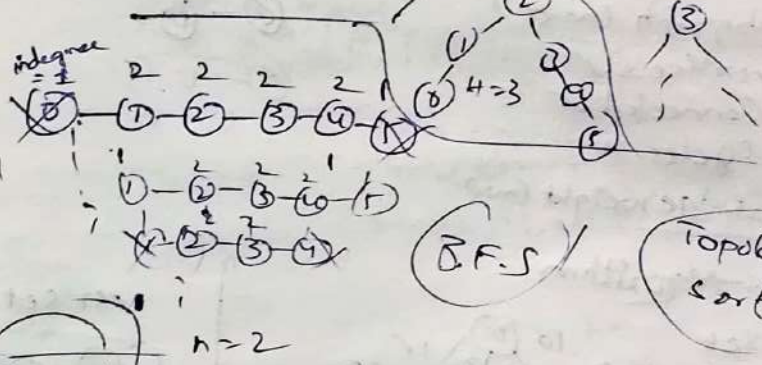
DFS

T.C. - E



- central nodes
- answers can be almost 2 roots





BFS

Topological sort

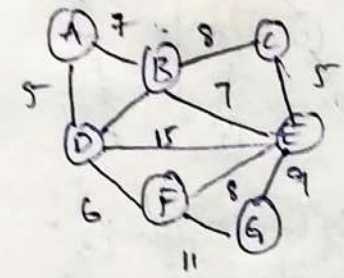
n = 2

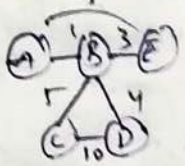
```
findMinHeightTrees(n, edges)
{ unordered_map<int, vector<int>> adj;
  vector<int> indegree(n);
  for (auto& edge : edges)
  { int u = edge[0];
    int v = edge[1];
    indegree[u]++;
    indegree[v]++;
    adj[u].push_back(v);
  } adj[v].push_back(u);
vector<int> result;
queue<int> que;
for (int i=0; i<n; i++)
{ if (indegree[i]==1)
  { que.push(i);
} }
```

```
if(n==1) { return 0; }
while (n > 2)
{ int size = que.size();
  n -= size;
  while(size--)
  { int u = que.front();
    que.pop();
    for (int &v : adj[u])
    { indegree[v]--;
      if(indegree[v]==1)
      { que.push(v); }
} } }
while(!que.empty())
{ result.push_back(que.front());
  que.pop();
}
return result;
}
```

Minimum Spanning Tree

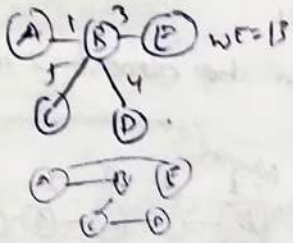
↳ (MST) is a subset of the edges of a connected, edge-weighted undirected graph that connects all the vertices together, without any cycles and with the minimum possible total edge weight.





Subgraph

MST
undirected
connected
weighted

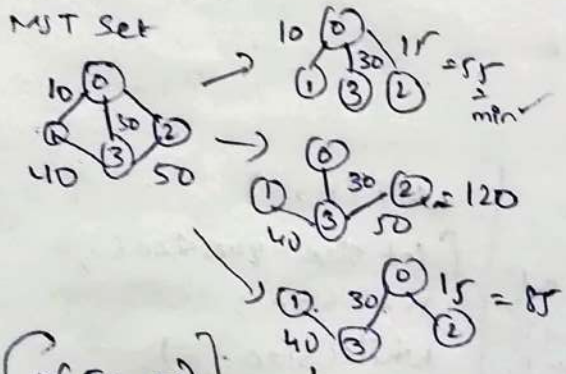


Subgraph (MST)
vertices ✓
Connected ✓
Cycles ✗
edge weight (min)

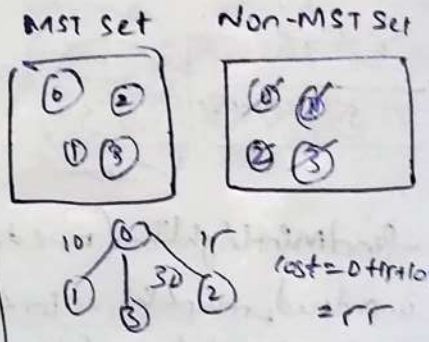
MST
nodes ✓ + connected
Cycle ✗
edge wt (min)

Prims Algorithm

MST Set



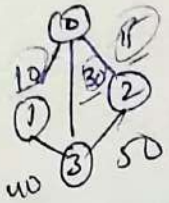
O(ElogE)



Pair (node, cost)

```
pq.add(pair(0,0)
boolean vis[]=F
while(!pq.isEmpty())
{ pair curr = pq.remove()
  if(!vis[curr.node])
  { vis[curr.node]=T
    cost += curr.cost
    for(int i=0; i<graph[curr.node].size;i++)
    { Edge e=graph[curr.node].get(i)
      if(!vis[e.dest])
        pq.add(e.dest, e.wt)
}}}
```

[mincost)
↓
DSU
pq (non MST)
vis[](MST)



Pair(n, cost)

(0, 0) => +0

(1, 10) => +10

(2, 15) => +15

(3, 30) => +30

already visited (3, 40) => ✗

(3, 50) => ✗

PQ (sort automatically)

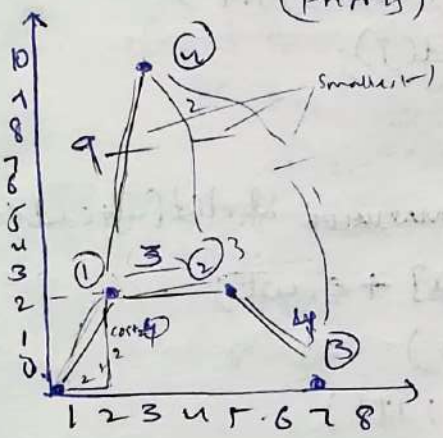
(1, 10) (2, 15) (3, 30) (3, 40) (3, 50)

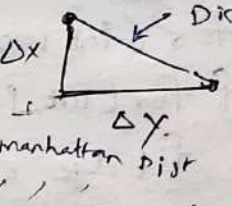
T	T	T	T	vis
0	1	2	3	

I.C : $O(n^2 \log n)$ (CB84)

S.C : $O(n^2)$

Min cost to connect all the points (prims)





H.S

vis [0, 1, 3, 4, 2] == len(no of nodes)

Heap : [dis, point]

[(0,0) (4,3) (7,3) (3,3) (4,4)

(7,0) (7,4) (13,2) (10,2) (9,2)]

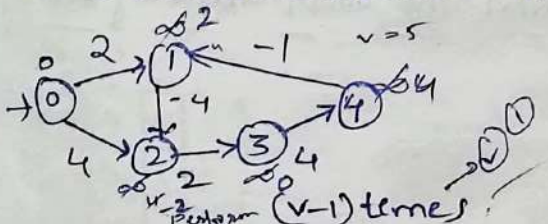
stop

cost : 0 + 4 + 3 + 4 + 9 =

	(dijkstra)			
	DJK	Bellman	DJK	BFA
works only for positive (+ve) wts	greedy	DP	$O(E+V\log V)$	$O(V \cdot E)$
	TC ↓↓			

Bellman Ford Algorithm

↳ shortest distance from the source to all vertices



$\theta(V \cdot E)$

0→1 (2) 0→2 (-2)

0→3 (0) 0→4 (4)

E (2)

for all edges (u, v)

{ if dist[u] + wt(u, v) < dist[v]

dist[v] = dis[u] + wt(u, v)

}

Perform (V-1) times (1)

dist[]

0	∞	∞	∞	∞
0	1	2	3	4

1st	0	2	-2	0	4
2nd	0	2	-2	0	4
3rd	0	2	-2	0	4
4th	0	2	-2	0	4

(maximum) run V-1 times

for (int k = 0 to v-1) — (1)

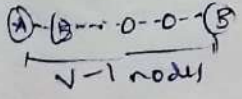
(2) for (int i = 0 to v)

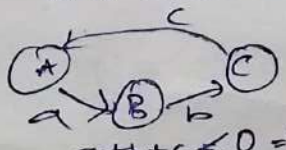
for (int j = 0 to graph[i].size())

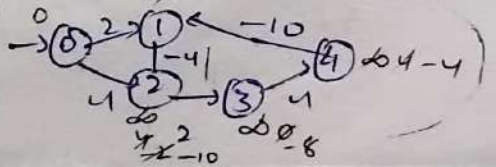
Edge e = graph[i].get(j)

→ BFA doesn't work for Negative cycles

(-ve ↑ getting on increasing wrong result)







```
void bellmanFord(graph[], int src, int V)
{ int dist[] = new int[V];
  for(int i=0; i<V; i++)
  { if(i != src)
    { dist[i] = Integer.MAX_VALUE;
    }
  }
  for(int k=0; k<V-1; k++) //O(V)
  { //O(E)
    for(int i=0; i<V; i++)
    { for(int j=0; j<graph[i].size(); j++)
      { Edge e = graph[i].get(j);
        int u = e.src;
        int v = e.dest;
        if(dist[u] != Integer.MAX_VALUE && dist[u]+e.wt<dist[v])
        { dist[v] = dist[u] + e.wt;
        }
      }
    }
  }
  //O(E) (// detect -ve wt cycles)
  for(int i=0; i<dist.length; i++)
  { S.op(dist[i] + " ");
  }
}
```

6. 1282. Group the people Given the Group Size they belong to

groupSizes = {3, 3, 3, 3, 3, 1, 3}
(0 1 2 3 4 5 6)

map

size	people
3	{0,1,2}
	{3,4,6}
1	{5}

Res = [0,1,2] [5] [3,4,6]

```
L = groupSizes[i]
if(mp[L].size() == L)   // mp[L] = {0,1,2}
    res.append(mp[L])
```

Floyd Warshall Algorithm

T.C - $O(V^3)$
S.C - $O(V^2)$
(DP over intermediate nodes)

↳ Multi-Source Shortest path

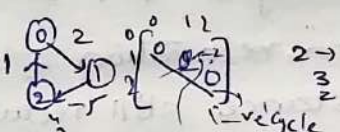
$$dist[i][j] = \min(dist[i][j], dist[i][k] + dist[k][j])$$

use → . All pairs shortest path
+ handles negative edges (× not neg cycles)
- Dense graphs, small-medium V

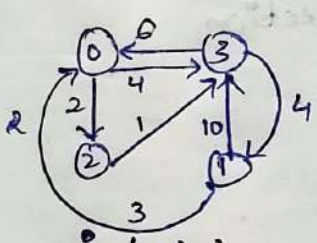
```
n = len(dist)   # matrix
for k in range(0,n)  // intermediate node
    for i in range(n):
        for j in range(n):
            if dist[i][k]!=INF and dist[k][j]!=INF:
                dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j])
```

Negative cycle (optional)

```
for i in range(n):
    if dist[i][i]<0:
        print("neg cycle exist")
```



transitive dependency
a → b b → C
a → C



mat	0	1	2	3
0	0	∞	2	4
1	3	0	∞	10
2	∞	∞	0	1
3	6	4	∞	0

via 0

	0	1	2	3
0	0	∞	2	4
1	3	0	5	7
2	∞	∞	0	1
3	6	4	8	0

i=1 j=2
1→0+0→3
1→3 = 7

via 1

	0	1	2	3
0	0	∞	2	4
1	3	0	5	7
2	∞	∞	0	1
3	6	4	8	10

via 2

	0	1	2	3
0	0	∞	2	3
1	3	0	5	6
2	∞	∞	0	1
3	6	4	8	0

via 3

	0	1	2	3
0	0	7	2	3
1	3	0	5	6
2	7	5	0	1
3	6	4	8	0

① 0 through
② Directly
③ Not connected

① ordering doesn't matter [A→B→D→C] = A→D→C

Dijkstra :

Positive

- Sparse: $E \log V \times V$ → $V^2 \log E$
- Dense: $O(V^2) \times V \Rightarrow O(V^3)$

Floyd Warshall
↳ Implementation easy
↳ Dense / Sparse (any)

Bellman Ford
$VE \cdot V = V^2E$
$= V^2 \times V^2$
$= V^4$

Dijkstra vs Bellman-Ford vs Floyd-Warshall

Algorithm	Negative edges	Detect Neg cycle	Works with Neg(cycle)
Dijkstra	✗	✗	✗ Breaks
Bellman-Ford	✓	✓	✗ reports cycle
Floyd-Warshall	✓	✓	✗ reports cycle

Why?

Dijkstra:

↳ Assumes once a node is finalized, it can't get shorter

↳ Negative edge violates this manipulation assumption.

↳ Fails silently → wrong answers

Bellman-Ford:

↳ Relaxes edges v-1 times.

↳ one extra relaxation

- if distance still reduces → negative cycle exists.

↳ Best for single-source + cycle detection.

Floyd-Warshall:

↳ Computes: all-pairs shortest paths.

↳ After completion:

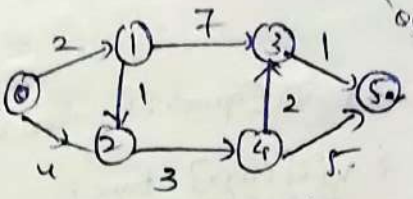
- if dist[i][i] < 0 → negative cycle reachable from i

↳ Best for all-pairs + dense graph.

- If $E > V^2 / \log V$ → treat as dense
- Else → Sparse
- Sparse → few roads, smart shortcuts
- Dense → everything connects, brute force wins

(shortest dist from src to all vertices) T.C $O((E+V)\log V)$

Dijkstra's Algorithm



pair(node, dist)

Priority queue (easy)

```
while(!pq.isEmpty())
{ pair curr = pq.remove();
  if(!vis[curr.node])
    vis[curr.node] = T
  for(int i=0 to ..) // neighbours
    Edge e = graph[curr.node].get(i)
    u → e.src
    v → e.dest
    if(dist[u]+e.wt < dist[v])
      dist[v] = dist[u]+e.wt
      pq.add(v, wt)
}
```

pq (some values added)

(0,0)	(1,2)	(2,3)	(2,4)	(4,6)	(3,8)	(5,9)	(4,9)	(5,11)

dist

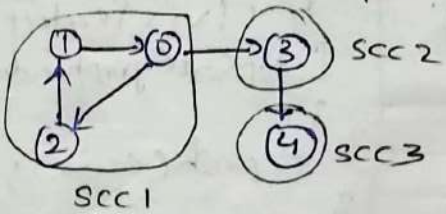
0	2	3	8	6	14
0	1	2	3	4	5

vis

T	T	T	T	T	T
0	1	2	3	4	5

Strongly Connected Component

↳ scc is a component in which we can reach every vertex of the component from every other vertex in that component.

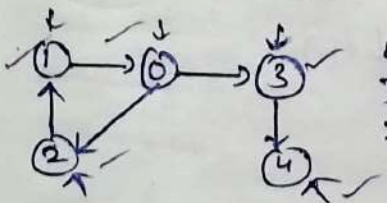


Kosaraju's Algorithm (works for only directed graph, merging diff algorithm into 1 algo.)

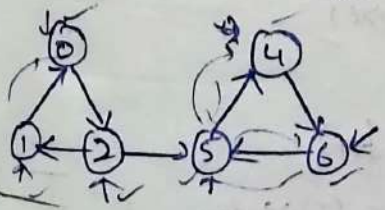
DFS

↳ reverse dfs (for scc) (top from last)

if not from rev dfs it will complete in 1 go no scc



4
3
2 1 0



6 5 4 scc

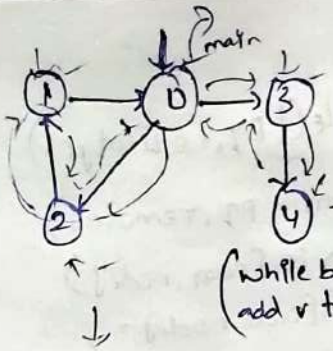
2 1 0 scc

Steps:

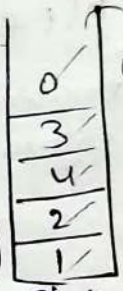
a. Get nodes in stack (topological sort)

b. Transpose the graph (reverse edges b/w vertices)

c. Do DFS according to stack nodes on the transpose graph.



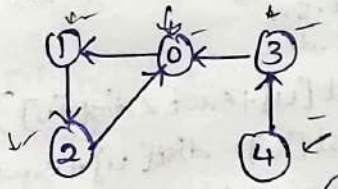
(1) (Topological sort)



0 3 4 2 1

(while backtracking add v to stack)

(2) (Transpose)



(3) (pop & do dfs & print scc)

0 1 2 - scc

3 - scc

4 - scc

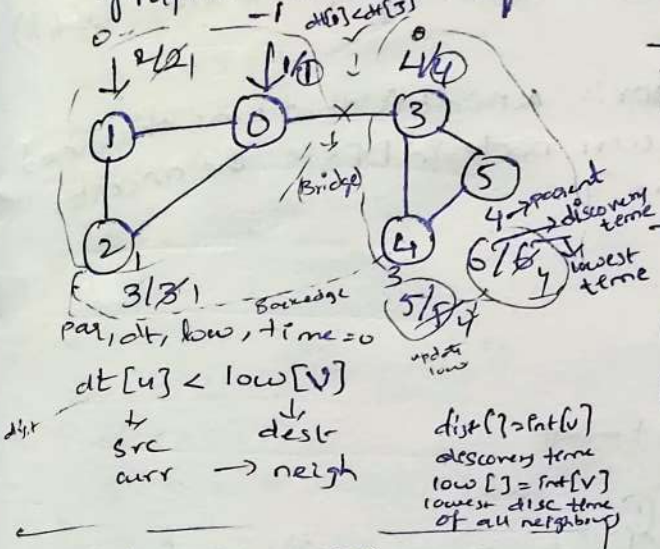
(ans → 0 1 2
3
4)

```
void kosarajuAlgo(graph[], int v)
{  // step-1 -O(V+E)
   Stack<Integer> s = new Stack<>();
   boolean vis[] = new boolean[v];
   for(int i=0; i<v; i++)
   { if(!vis[i])
     { topSort(graph, i, vis, s);
   } }
   // step-2 -O(V+E)
   ArrayList<Edge> transpose[] = new ArrayList[v];
   for(int i=0; i<graph.length; i++)
   { vis[i] = false;
     transpose[i] = new ArrayList<Edge>();
   }
   for(int i=0; i<v; i++)
   { for(int j=0; j<graph[i].size(); j++)
     { Edge e = graph[i].get(j); //e.src(i)->e.dest
       transpose[e.dest].add(new Edge(e.dest, e.src));
   } }
   // step-3 - O(V+E)
   while(!s.isEmpty())
   { int curr = s.pop();
     if(!vis[curr])
     { dfs(transpose, curr, vis);
       System.out.println();
   } }
}
```

```
void dfs(graph[], int curr, boolean vis[])
{  vis[curr] = true;
   System.out.print(curr+" ");
   for(int i=0; i<graph[curr].size(); i++)
   { Edge e = graph[curr].get(i);
     if(!vis[e.dest])
     { dfs(graph, e.dest, vis);
   } }
}

void topSort(graph[], curr, vis[], Stack s)
{  vis[curr] = true;
   for(int i=0; i<graph[curr].size(); i++)
   { Edge e = graph[curr].get(i);
     if(!vis[e.dest])
     { topSort(graph, e.dest, vis, s);
   } }
   s.push(curr);
}
```

Bridge in Graphs (using Tarjan's Algorithm)

↳ Bridge is an edge whose deletion increases the graph's number of connected components.



Algo – DFS

vis[curr] = T

dt[curr] = low[curr] = ++time

for (int i=0 to neigh)

Edge e.

① par → continue

② !vis[e.dest] → dfs

low[curr] = min(low[curr], low[neigh])

if (dt[curr] < low[neigh])

print(curr + neigh)

③ vis[e.dest]

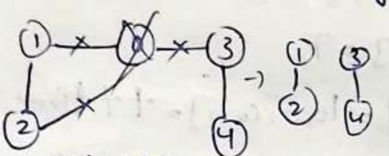
(low[curr] = min(low[curr], dt[neigh])

(if vis[0] ever connection exist if we remove so it is not bridge)

```
void dfs (ArrayList<Edge>[] graph[], int curr, bool vis[],
  int dt[], low[], time, par)
{ vis[curr] = true;
  dt[curr] = low[curr] = ++time
  for(int i=0; i<graph[curr].size(); i++)
  { Edge e = graph[curr].get(i);
    if (e.dest == par)
    { continue;
    } else if (!vis[e.dest])
    { dfs(graph, e.dist, vis, dt, low, time, curr);
      low[curr] = Math.min (low[curr], low[e.dest]);
      if (dt[curr] < low[e.dest])
      { S.o.p("bridge is: " + curr + " --- " + e.dest);
      }
      else
      { low[curr] = Math.min (low[curr], dt[e.dest]);
  } } }

Void getBridge (graph[], int v)
{ int dt[] = new int [v];
  int low[] = new int [v];
  int time = 0;
  boolean vis [] = new boolean [v];
  for (int i=0; i<V; i++)
  { if (!vis [i])
    { dfs (graph, i, vis, dt, low, time, -1);
} } }
```

Articulation point.

↳ A vertex in an undirected connected graph is an articulation point (or cut vertex) if removing it (and edges through it) disconnects the graph.



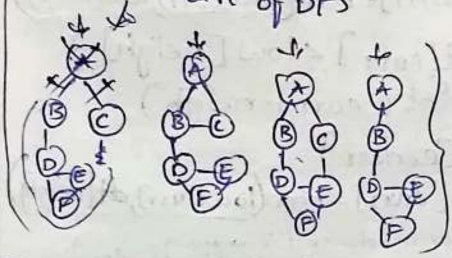
[Ancestor :- a node A that was discovered before curr node in DFS is the ancestor of curr]

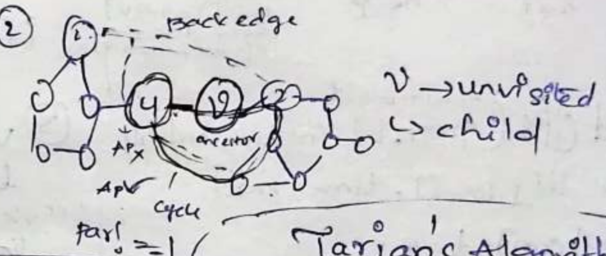
naive AP — O(V×(V+E))

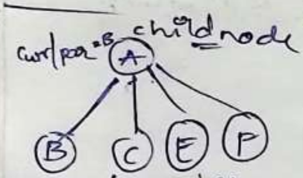
DFS

node — AP

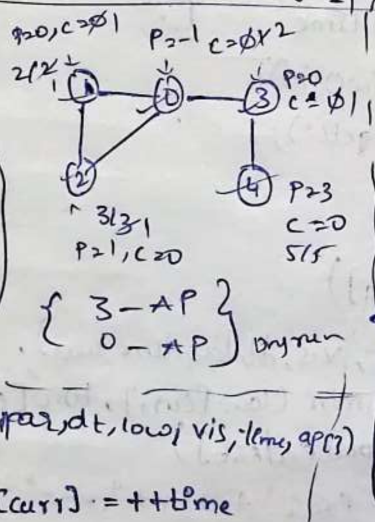
① node - corner node — par = -1 ✓ & disconnected children > 1 → AP — start of DFS







{neigh par — ignore} {neigh visited — backedge ancestor} {neigh unvisited — child}



{ 3 — AP, 0 — AP } Dry run

Tarjan's Algorithm

vis[curr] = t

dt[curr] = low[curr] = ++time

children = 0

for (neigh)

Edge e (src → dest)

① neigh = par → ignore

② if (neigh - vis)
low[curr] = min(low[curr], dt[neigh])

③ if (neigh - unvis)
dfs ( )
low[curr] = min(low[curr], low[neigh])
if (dt[curr] <= low[neigh] && par != -1)
curr → AP
child++

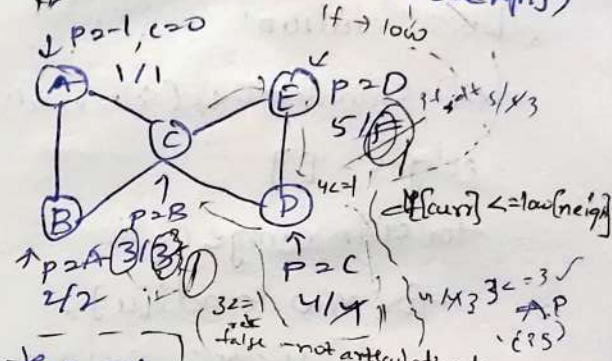
if (par = -1 && children > 1)
curr → AP

```
void dfs(graph[], curr, par, dt, low, vis, time, ap[])
{ vis[curr] = true
  dt[curr] = low[curr] = ++time
  int children = 0
  for (int i=0; i<graph[curr].size(); i++)
  { Edge e = graph[curr].get(i)
    int neigh = e.dest;
    if (par == neigh)
    { continue;
    }
    else if (vis[neigh])
    { low[curr] = Math.min(low[curr],
                           dt[neigh]);
    }
    else
    { dfs(graph, neigh, curr, dt, low, vis, time, ap);
      low[curr] = Math.min(low[curr], low[neigh])
      if (dt[curr] <= low[neigh] && par != -1)
      { ap[curr] = true;
      }
      children ++;
} }
```

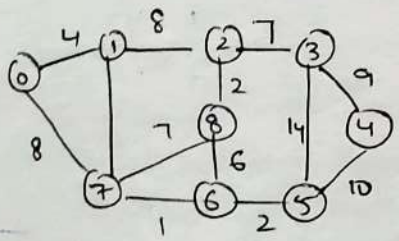
```
if (par == -1 && children > 1)
{ ap[curr] = true;
}
```

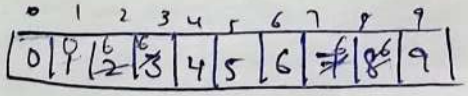
```
void getAP(graph[], v)
{ int dt[] = new int[v];
  int loco[] = new int[v];
  int time = 0;
  boolean vis[] = new boolean[v];
  boolean ap[] = new boolean[v];
  for(int i=0; i<v; i++)
  { if(!vis[i])
    { dfs(graph, i, -1, dt, loco, vis, time, ap); }
  }
  for(int i=0; i<v; i++)
  { if(ap[i])
    { S.O.P("AP : " + ap[i]); }
  }
}
```

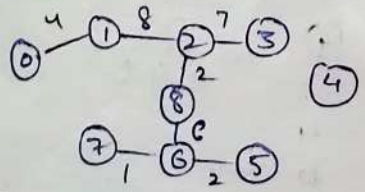
why ?! loco[curr] = min(loco[curr], dt[neighbour])



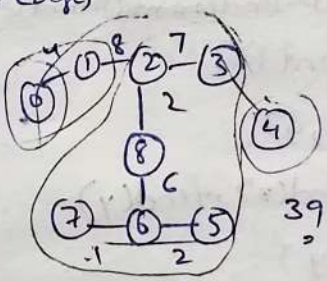
Kruskals Algorithm







8- edges



different set

Connect them

8 c

6 8

2- change

6- change

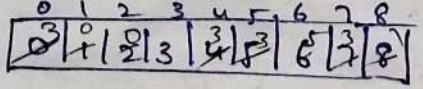
Disjoint set

O(1) constant

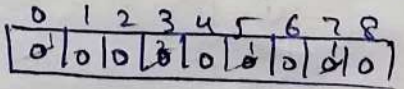
O(n)

(0,1)
(1,2)
(3,4)
(5,6)
(7,8)
(3,6)

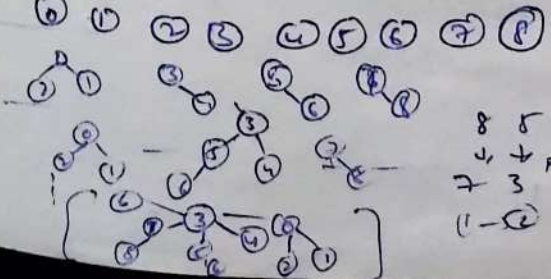
Ultimate parent, ?



rank



Which set to merge to which change ?



8 5

7 3

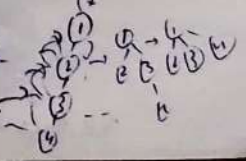
Union by rank/size

1. Find the ultimate parent of u & v
2. find the rank of ultimate (pu, pv)
3. Merge the smaller set into larger set (rank)

1. Find parent
2. Union by Rank

O(4 α), constant

Path compression



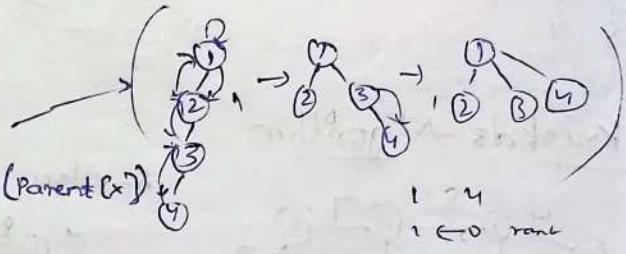
Code:

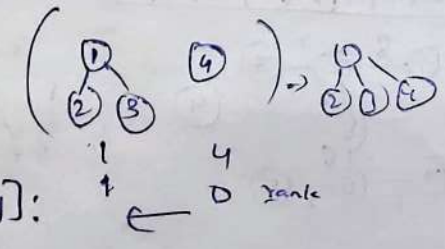
dsu → T.C. O(E log E) — sorting edge,
O(E, α(v)) = O(E)
→ S.C. O(V+E)
↑ edgelist - O(E)
DSU(parent+rank) - O(V)

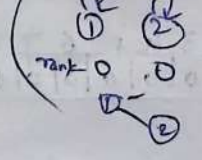
(kruskal → edge based, sparse graph)
prim → node based, dense graph)

```
class Solution:
    def spanningTree(self, V, adj):
        edges = []
        for u in range(V):
            for v, w in adj[u]:
                if u < v: # avoid duplicate
                    edges.append((w, u, v))
        edges.sort()
        parent = list(range(V))
        rank = [0] * V
        def find(x):
            if parent[x] != x:
                parent[x] = findParent(parent[x])
            return parent[x]
        def union(x, y):
            rx, ry = find(x), find(y)
            if (rx == ry):
                return false
            if rank[rx] < rank[ry]:
                parent[rx] = ry
            elif rank[rx] > rank[ry]:
                parent[ry] = rx
            else:
                parent[ry] = rx
                rank[rx] += 1
            return True
        mst_weight = 0
        edges_used = 0
        for w, u, v in edges:
            if union(u, v):
                mst_weight += w
                edges_used += 1
                if edges_used == V-1:
                    break
        return mst_weight
```

(edges.sort(key=lambda x: x[2]) // sort by wt)

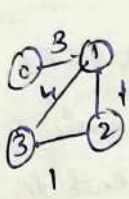






(1334)

1. Find the city with the smallest No. of neighbours.



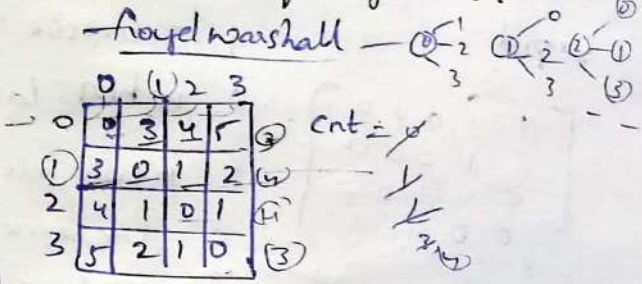
threshold = 4

city

0 → 1 2 ②
1 → 0, 2, 3 ③
2 → 0, 1, 3 ③
3 → 1, 2 ②

ans → 3
ret

floyd warshall



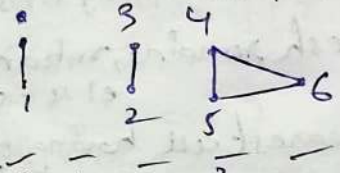
	0	1	2	3	
0	0	3	4	5	③
1	3	0	1	2	④
2	4	1	0	1	⑤
3	5	2	1	0	③

cnt = 0 → 1 → 3 (4)

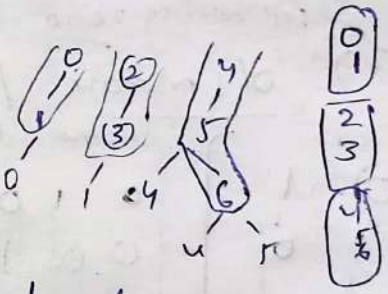
ini
cntMin = 5 → 3;
city = 1 0 3

1. Initialize n×n dist matrix with ∞
2. Fill edge wts and set d[i][i] = 0.
3. Apply floyd-warshall to get all pairs shortest path
4. count cities reachable within the threshold for each city.
5. return the city with min count (tie → larger index)

eg 8) 8. connected components in an undirected Graph.



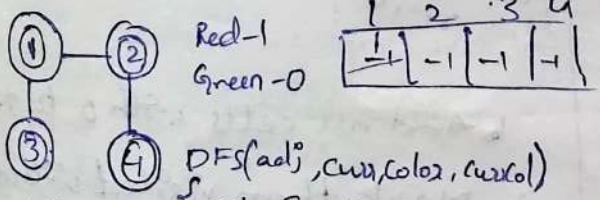
vis - 0 1 2 3 4 5 6



Algo steps:

1. Build adjList from edges
2. Maintain vis arr
3. Run DFS from each unvisited node
4. collect nodes as one connected component
5. Continue until all nodes are vis.

(284) 9. Is Graph Bi-partite?

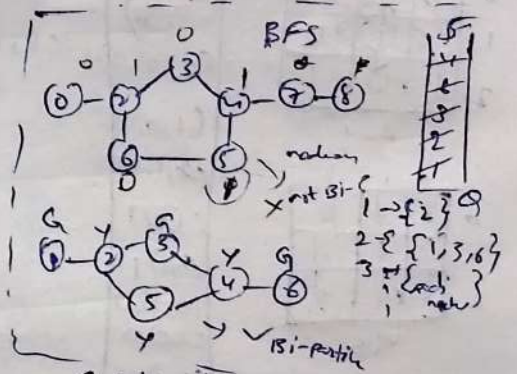


Red-1
Green-0

1	2	3	4
1	-1	-1	-1

adj:
1 → 2, 3
2 → 1, 4
3 → 1
4 → 2

```
DFS(adj, curr, color, currCol)
{ color[curr] = currColor;
  for(int v : adj[curr])
  { if (color[v] == color[curr])
      return false;
    if(color[v] == -1) // never colored
      colorOfV = 1 - currColor // oppCol.
      if (dfs(adj, v, color, colorOfV) == false)
        return false
  } return true
```



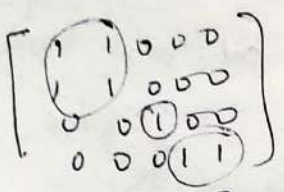
BFS

odd length cycle - Not Bipartite
even length - Bipartite

```
bool isBipartite(V, adj)
vector<color(V, -1);
for(i=0 to V)
  if(color[i] == -1) if(dfs(adj, V, col, 1)) return false
} ret true
```

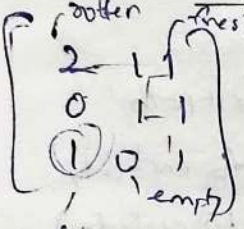
10. (200) Number of Islands

2d grid



1. Traverse each cell in the grid
2. When land ('1') is found, start DFS
3. Mark all connected land as water
4. Increment island count for each dfs start
5. Return Total island count

11. (994) Rotting oranges

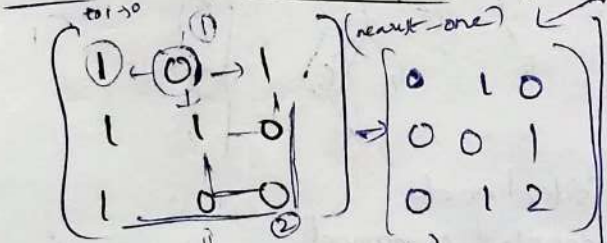


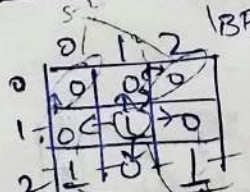
find minutes?

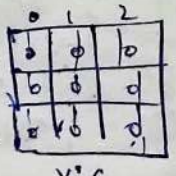
1. Push all initially rotten oranges into queue
2. count fresh oranges
3. BFS level by level to rot neighbours
4. Track time using BFS depth
5. If fresh remains, return -1 else time

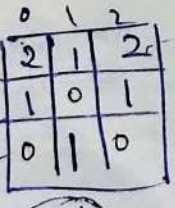
12. (542) 0/1 Matrix / Distance of nearest cell having one

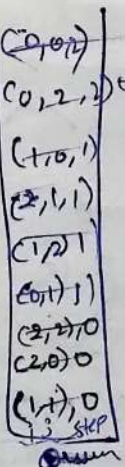
"cell having zero"



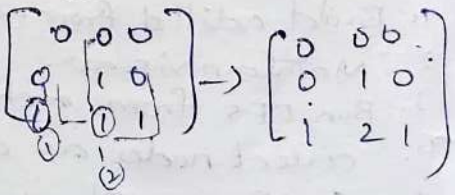


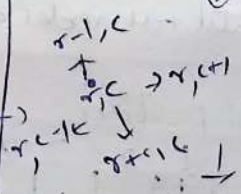






nearest zero

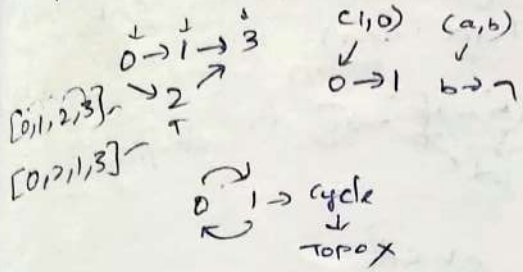




1. Add all cells with 0 to queue
2. Mark them visited with dist 0
3. Run BFS in 4 directions
4. Update distance using BFS level
5. Return distance matrix.

Course Schedule - II

13. numCourses = 4, pre-req = [[1,0],[2,0],[3,1],[3,2]]




```
bool isCycleDFS(src, vis, recPath, edges)
{ vis[src] = true;
  recPath[src] = true;
  for(int i=0; i<edges.size(); i++)
  { int v = edges[i][0]; // v -> u
    int u = edges[i][1];
    if(src == u)
    { if(!vis[v])
      { if(isCycleDFS(v, vis, recPath, edges))
        { return true; }
      }
      else if(recPath[v]) // Backedge -> cycle
      { return true; }
    }
  }
  recPath[src] = false;
  return false;
}
void topoOrder(src, vis, s, edges)
{ vis[src] = true;
  for(int i=0; i<edges.size(); i++)
  { int v = edges[i][0];
    int u = edges[i][1];
    if(src == u)
    { if(!vis[v])
      { topoOrder(v, vis, s, edges); }
    }
  }
  s.push(src);
}
vector<int> findOrder(n, edges)
{ vector<bool> vis(n, false);
  vector<bool> recPath(n, false);
  vector<int> ans;
  for(int i=0; i<n; i++)
  { if(!vis[i])
    { isCycleDFS(i, vis, recPath, edges);
      { return ans; }
    }
  }
  // Topological sorted order
  Stack<int> s;
  vis.assign(n, false);
  for(int i=0; i<n; i++)
  { if(!vis[i])
    { topoOrder(i, vis, s, edges); }
  }
  while(s.size() > 0)
  { ans.push_back(s.top());
    s.pop();
  }
  return ans;
}
```

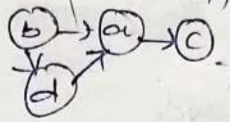
14. (gfg) — Alien Dictionary

baa
abcd
abca
cab
cad

(normal dic — a, b, c ... x, y, z
abca
abcd
baa
cab
cad)

compare k = 4
s1 = baa — arr[i]
s2 = abcd — arr[i+1]
b → a (b < a)

abcd / abca → d → a
abca / cab → a → c
cab / cad → b → d



→ (Topological)

- find out alien order?
[b d a c]

cycle → Toposort ✗

abc / abcd → error fix —

abcd / abc → ✗ (shorter)

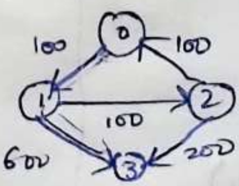
abc
bat
ade
✗

1. Initialize graph, vis, recStack
2. build graph from adj words
3. check invalid prefix condition
4. Run dfs for topological sort
5. Detect cycle using rec stack
6. Reverse DFS order for result.

15. (787) Cheapest flights within k stops :

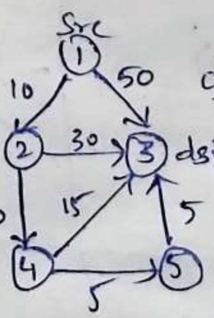
n = 4, flights = [{0, 1, 100}, {1, 2, 100}, {2, 0, 100}, {1, 3, 600}, {2, 3, 200}]
(cities) — from, to, cost

src = 0
dst = 3
k = 1 (max taken stops)



(k = 1)
700
(100 + 600)

adj
1 → {2, 10}, {3, 50}
2 → {3, 30} {4, 10}
3 → { }
4 → {3, 15} {5, 5}
5 → {3, 5}



src = 1
dst = 3
k = 2

	src 1	2	3	4	5
dist	0	∞ 10	∞ 50 40 35	∞ 20	∞

q: [(1, 0)] → {2, 10} {3, 50} {3, 40} {4, 20} {3, 35}

u = 1
cost = 0
{2, 10} 0 + 10 = 10
{3, 50} 0 + 50 = 50

u = 2
cost = 10
{3, 30} 10 + 30 = 40 → {3, 40}
{4, 10} 10 + 10 = 20 → {4, 20}

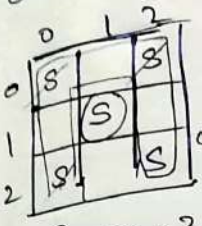
u = 3, c = 50
u = 3, c = 40
u = 4, c = 20
→ {3, 15} 20 + 15 = 35 → {3, 35}
→ {5, 5} 20 + 5 = 25 → {5, 25}
steps 3

while (!que.empty() && steps <= k) {
size = q.size()
while size > 0
u, cost = q.front() ... for v, wt in adj[u]:
if cost + wt < dist[v]
dist[v] = cost + wt
q.push((v, dist[v]))
size--
step += 1

16. (Call) most Stones Removed with same Row or Column

stones = [(0,0), (0,2), (1,1), (2,0), (2,2)]

Approaches
DFS or DSU ✓ (find union)



Groups = 2

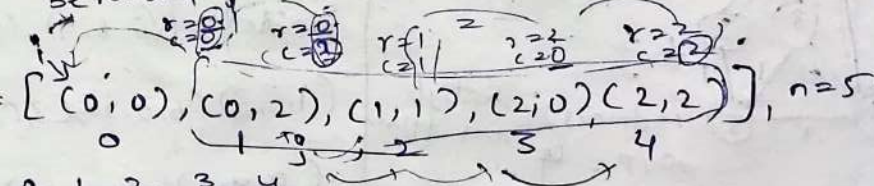
G_1 G_2 G_3 ---
x_1 x_2 x_3 ---

removed = $(x_1-1)+(x_2-1)+1$ ---
$= (x_1+x_2+x_3) - 1 - 1 - 1$
$= (x_1+x_2+x_3+\text{---}) - (1+1+1\text{--})$
$= \boxed{n - (\text{no. of groups})}$
(total stones)

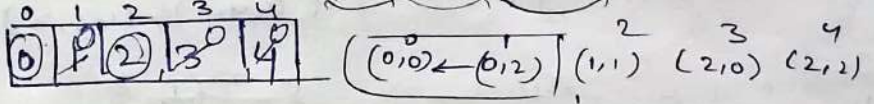
G_1 G_2
4-1 1-1
$= 3 + 0$ ⇒ 3 will be removed

$\therefore 5 - 2 = 3$.

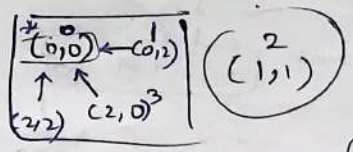
stones = [(0,0), (0,2), (1,1), (2,0), (2,2)], n = 5



parent =



Group = 1 + 1
= 2
n − (no. of g) ⇒ 5 − 2 = 3




```
remove Stones (stones)
{ n = stones.size();
  parent.resize(n);
  rank.resize(n);
  for(int i=0; i<n; i++)
  { parent[i] = i;
  } rank[i] = 1;
  for(int i=0; i<n; i++)
  { for(int j=i+1; j<n; j++)
    { if(stones[i][0] == stones[j][0]
      || stones[i][1] == stones[j][1])
      { union(i,j); // O(alpha)
  } } }
  int groups=0;
  for(int i=0; i<n; i++)
  { if(parent[i] == i)
    { groups++;
  } }
} return n - groups;
```

// S.C : O(n)

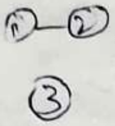
// T.C : $O(n^2 \times \text{alpha}) \to O(n^2)$ (alpha = small)

(compare if 2 rows or 2 col same
eg: (0,0) with remaining in stones)

(dsu)

```
find(i)
{ }
union(i,j)
{ }
```

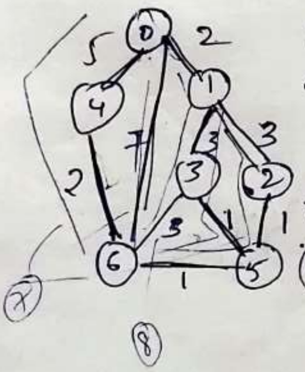
17. no. of provinces (547)



isconn = [[1,1,0], [1,1,0], [0,0,1]]

o/p = 2

18) (1976)
Number of ways to arrive at destination

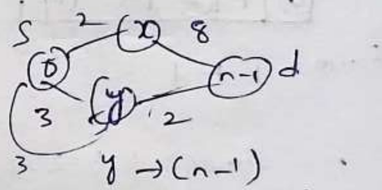


S = 0
d = n-1 (6)

t = 7
(no of ways) in shortest amt of time
o/p = 4

- Dijkstra's algo
(Single source shortest path)

n = 7
roads = [(0,6,7), (0,1,2), (1,2,3) (1,3,3) (6,3,3), (3,5,1), (6,5,1) (2,5,1) (0,4,5), (4,6,2)]



y → (n-1)
res[y] + 2 → (n-1)
3+2 = 5

result (s=0)
0	1	2	- -	n-1
0	∞	∞	1	∞ 5

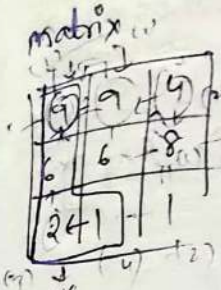
s to (n-1)
res [n-1]
Count [n-1]

Count
0	1	2	- -	n-1
0		1	3	5

ngbr = (n-1) (u,v,t)
t = 2 (nd) (distance dist)

```
if (currtime + 2 < res[ngbr])
{
    res [ngbr] = currtime + 2 (nd)
}   pq.push ((res[ngbr], ngbr));
    countpath [ngbr] = countpath [curr node];
elseif (currtime + 2 == res [ngbr])  (nd)
{
    countpath [ngbr] += countpath [y]; (curr node)  % m
}
ret. pathCount [n-1];
```

19. (329) Longest Increasing path in a Matrix.



LIS

T.C: O(m×n)

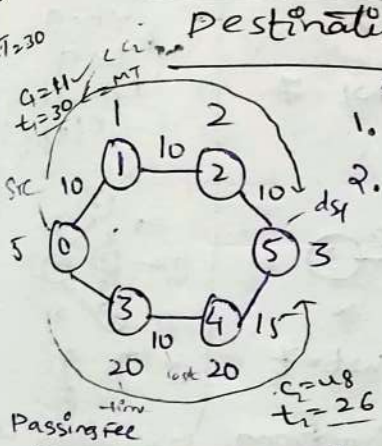
S.C: O(m×n)

CIP (DP)

	0	1	2
0	1	1	2
1	2	2	1
2	3	4	2

1. Use DFS from each cell to explore increasing paths
2. Stop DFS if out of bounds or value not increasing
3. Use memorization to avoid recomputation
4. Try all 4 directions from each cell.
5. Return the max value stored in DP.

20. (928) Min Cost to reach destination in Time

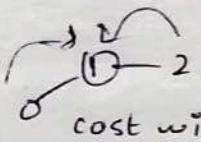


0	1	2	3	4	5
5	1	2	20	20	3

1. minimum possible cost
2. time ≤ maxTime

(simple Dijkstra's)

{cost, t, neigh}



cost will be same
time diff
∴ edge relaxation will be for time.

```
int dfs(maxTime, edges, passingFee)
{ // make adjacency list
  // node → {neigh, time}
  vector<int> minTime(n, INT_MAX);
  priority_queue<vector<int>, vector<vector<int>>, greater<vector<int>>> pq;
  pq.push({passingFee[0], 0, 0}) // {cost, time, node}
  minTime[0] = 0
  while (!pq.empty())
  { auto top = pq.top();
    pq.pop();
    int cost = top[0];
    int time = top[1];
    int node = top[2];
    if (node == n-1)
      return cost;
    for (auto &[neigh, t] : adj[node])
    { int nextTime = time + t;
      int newCost = cost + passingFee[neigh];
      if (newTime <= maxTime &&
          newTime < minTime[neigh])
      { minTime[neigh] = newTime;
        pq.push({newCost, newTime, neigh});
      }
    }
  }
  return -1
}
```

~~pq.push({newCost, newTime, neigh});~~

21. (1489) :- Find Critical and Pseudo-critical edges in MST

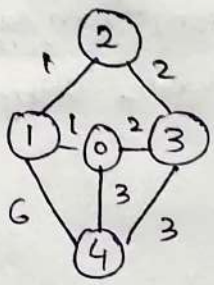
(kruskal, DSU) (prims)

Ex: n = 5

edges = [{0,1,1}, {1,2,1}, {2,3,2}, {0,3,2}, {0,4,3}, {3,4,3}, {1,4,6}]
(0) (1) (2) (3) (4) (5)

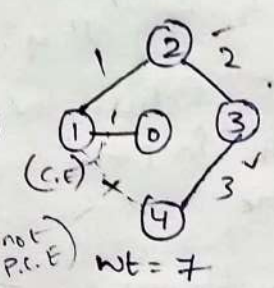
C.E = {1, 0}

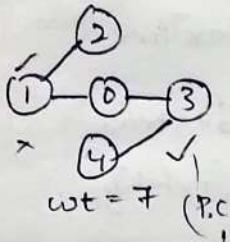
P.C.E = {2, 3, 4, 5}

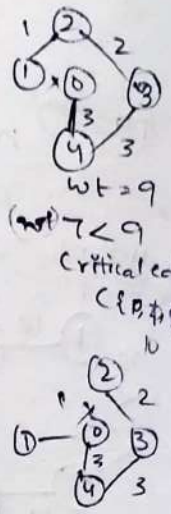


wt = 21
V = 5

(MST's) (may cycle min cost)







Critical Edge → An MST edge whose deletion from the graph would cause MST weight to increase.

Pseudo-Critical edge → is that which can appear in some MST's but not all.

* Sorted wt & we added idx.

[{0,1,1,0}, {1,2,1,1}, {0,3,2,3}, {2,3,2,2}, {0,4,3,4}, {3,4,3,5}, {1,4,6,6}]
(0) (1) (2) (3) (4) (5) (6)

u=0, v=1, wt=1, idx=0

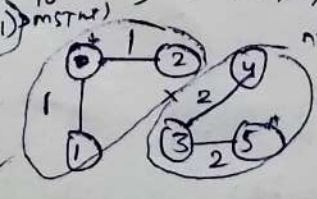
skipwt = kruskal(skip=i);
if (skipwt > MST_wt) — skip
{ Critical.push_back(idx); }
else if
== MST.wt — force include
Pseudo.critical

① Push original idx also
② Sort based on wt
③ Find MST_wt → kruskal
④ Check each edge
{ ① Skip (CE)
② Add (PCE) }

— kruskal(add edge) { wt == } (kruskal DSU)

edge case:
[{0,1,1,0}, {0,2,1,1}, {1,2,1,2}, {3,4,2,3}, {3,5,2,5}, {4,5,2,6}]

if(kruskal(-1,-1) == MSTwt) CE,
else (> MSTwt) C.E



for(i=0; i<n; i++)
if(find(i) != find(0))
return ∞
(connected - MST)

{2, 3, 4, 5}
skip

code

```
int kruskal(vec, skip_edge, add_edge)
{ int sum=0
  unionfind uf(N);
  if(add_edge != -1) // PC
  { uf.Union(vec[add_edge][0], vec[add_edge][1])
    sum += vec[add_edge][2];
  }
  for(int i=0; i<vec.size(); i++)
  { if(i == skip_edge) // CE
      continue;
    int u = vec[i][0];
    int v = vec[i][1];
    int wt = vec[i][2];
    int parent_u = uf.find(u);
    int parent_v = uf.find(v);
    if(parent_u != parent_v) // add this
    { uf.Union(u, v);
      sum += wt;
    }
  }
  for(int i=0; i<N; i++) // not connected
  { if(uf.find(i) != uf.find(0))
      return INT_MAX;
  }
  return sum;
}

vector findCriticalAndPseudo(I(vec1, vec2)
{ N = n;
  for(int i=0; i<edge.size(); i++) // add idx's to sort
  { edges[i].push_back(i);
  }
  sort(begin(edges), end(edges), lambda, // close
  auto lambda = [&](vector<int>&vec1, vector<int>&vec2)
  { return vec1[2] < vec2[2];
  };
  int MST_wt = kruskal(edge, -1, -1); // skip and
  cout << MST_wt << endl
  vector<int> critical;
  vector<int> pseudo-critical; // arr
  for(int i=0; i<edges.size(); i++) // E
  { if (kruskal(edges, i, -1) > MST_wt) // skipping
      critical.push_back(edges[i][3]); // idx
    else if (kruskal(edges, -1, i) == MST_wt) // force add this edge
      pseudocritical.push_back(edges[i][3]);
  }
  return {critical, pseudocritical};
}
```

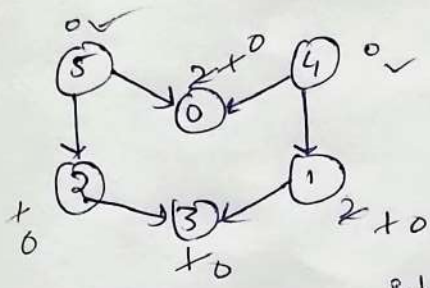
```
public:
class UnionFind
{ public:
  vector<int> parent;
  vector<int> rank;
  UnionFind(int n)
  { parent.resize(n);
    rank.resize(n, 0);
    for(int i=0; i<n; i++)
    { parent[i] = i;
    }
  } // ds u
  int find(int x)
  {
  }
  bool union(int x, int y)
  {
  }
};
```

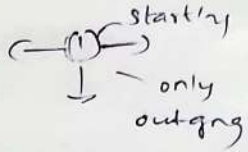
(graph)

using Kahn's algorithm (BFS)
Topological sorting $u \to v$

indegree (no. of incoming edges to the node)

ans → | 4 5 0 2 3 1 |





Q | 4 | 5 | 0 | 2 | 3 | 1 |

indeg | 2 | 2 | 1 | 1 | 0 | 0 |

① calc indeg of all nodes
② 0 indeg → Q.push (start)
③ while (Q.size() > 0)
curr = Q.front()
for (neighs of curr)
neigh indeg--
↓
Q.push

① vector <int> indeg(V, 0)
for (u = 0 to v)
for (int v : l[u])
indeg[v]++

② for (i = 0 to v)
if (indeg[i] == 0) → Q.push(i)

③ while (Q.size() > 0)
{ curr = Q.pop();
res.push_back(curr);
for (int v : l[curr])
indeg[v]--;
if (indeg[v] == 0) → Q.push(v)
}

res ✓ { re.size() == v ✓ DAG
nodes ——→ cycle ✗ }

5/1/26

Dynamic programming (DP)

1. (121) Best time to buy & sell Stock

```
mini = a[0], prof = 0
for (i = 1 to n)
{
    cost = a[i] - mini
    prof = max(prof, cost)
    min = min(mini, a[i])
}
ret prof
```

arr[] → [7, 1, 5, 3, 6, 4] n=6

buy — 1

sell — 6

5

mention only done once

(If you are selling on ith day you can buy on the min price from (1st to i-1)

2. (416) Partition Equal Subset sum

[2, 3, 3, 3, 4, 5]

(2 subsets such that their sum is same)

[2, 3, 5] [3, 3, 4]

c(0) = c(6)

Subseq - must follow order

Subset - can or cannot follow order

(S) → (S1) (S2)

S1 = S2 ✓ (S = total sum)

S1 = S2 = S/2

S → odd — ✗

target → (check subset sum = = target)

Subset → (sum S/2

Subset sum Equals to Target

arr → [2, 3, 1, 1] (0 1 2 3), target = 4

idx tar

f(3, 4)

take (eq-1) / not take

f(2, 3) f(2, 4)

f(1, 3) f(1, 4)

f(1, 2) f(1, 3)

take (3-2) / not take

f(0, 2) f(0, 1) f(0, 2)

idx tar

false

take = false

not take = True

recursion

```
f(idx, target)                    (1)
{ // B. case
  if (tar == 0) return true;
  if (idx == 0) return (a[0] == tar);
    if (dp[idx][tar] != -1) return dp[idx][tar];
  bool nottake = f(idx-1, target);
  bool take = false
    if (tar >= a[idx])
      take = f(idx-1, tar - a[idx])
  dp[idx][tar]
  return (take) or (nottake)
```

memo idx target TC → O(N×target) (2)
dp [10^3+1] × [10^3+1] SC → O(N×tar) + O(N)
Aux. space

1) Base case
2) Form the nested loop
{ind → (1 → n-1)
{tar
(1 → tar)
}}

n-1 Top-do..
n-2
n-3
0 bottom-up

prev[0] = curr[0] = True)

```
bool SubsetToK (n, k, arr)
{ vec<bool> prev(k+1, 0), curr(k+1, 0).
  prev[0] = curr[0] = true;
  prev[arr[0]] = true;
  for(int idx = 1; idx<n; idx++).
  { for(int tar = 1; tar<=k; tar++)
③   { bool notTake = prev[target];
      bool take = false
      if (arr[ind]<=tar) take = prev[tar-arr[idx]];
    } curr[targ] = take | notTake;
    prev = curr
  } return prev[k];
```

S.C → O(tar)

idx→0 [T | | T |] prev
idx→1 [T | - | - | - | - | - | -] curr pre
cur

```
① SubsetSumToK(n, k, arr)
{ vec<vec<bool>> dp(n, vec<bool>(k+1, 0));
  for (i=0; i<n; i++) dp[i][0] = true;
  dp[0][arr[0]] = true
  for(int ind = 1; ind<n; ind++)
  { for(int tar = 1; tar<=k; tar++)
    { bool notTake = dp[ind-1][tar]
      bool take = false
      if (arr[ind] <= tar) take = dp[ind-1][tar-arr[ind]]
  }  } dp[ind][tar] = take | notTake;
  return dp[n-1][k]
```

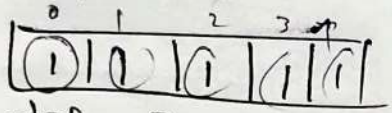
3. (494) Target Sum

Given an arr int arr and a tar sum. Find the total no. of ways you can build an expression to reach the target. Just use '-' and '+'.

[1, 1, 1, 1, 1]

target = 3

eg: +1 −1 +1 −1 +1 = 1
+1 +1 +1 +1 +1 = 5
−1 −1 −1 −1 −1 = −5
+1 +1 +1 +1 −1 = 3



dp[sum] = no_of_ways

① dp[0] = 1 — '+1' = 1, '-1' = −1 | dp[1] = 1, dp[-1] = 1

to get sum '0' there is only one way (no ele)

T.C → O(n²)
S.C → O(n)

② dp[1] = 1 — +1 = 2, −1 = 0
dp[-1] = 1 — +1 = 0, −1 = −2
| dp[-2] = 1, dp[0] = 2, dp[2] = 1

③ dp[-2] = 1 — +1 = −1, −1 = −3
dp[0] = 2 — +1 = 1, −1 = −1
dp[2] = 1 — +1 = 3, −1 = 1
| dp[-3] = 1, dp[-1] = 3 ways, dp[1] = 3, dp[3] = 1

④ dp[-3] = 1 — +1 = −2, −1 = −4
dp[-1] = 3 — +1 = 0, −1 = −2
dp[1] = 3 — +1 = 2, −1 = 0
dp[3] = 1 — +1 = 4, −1 = 2
| dp[-4] = 1, dp[-2] = 4, dp[0] = 6, dp[2] = 4, dp[4] = 1

⑤ dp[-4] = 1 — +1 = −3, −1 = −5
dp[-2] = 4 — +1 = −1, −1 = −3
dp[0] = 6 — +1 = 1, −1 = −1
dp[2] = 4 — +1 = 3, −1 = 1
dp[4] = 1 — +1 = 5, −1 = 3
| dp[-5] = 1, dp[-3] = 5, dp[-1] = 10, dp[1] = 10, dp[3] = 5, dp[5] = 1

(to get target sum '3' we have 5 ways)

knapsack

0/1 knapsack — fractional knapsack

3kg
take 2kg — 1kg left

weights = {3, 4, 6, 5}
profits = {2, 3, 1, 4} [W = 8, n = 4] $\sum w_i x_i \le W$

(values)

i \ wt →	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1 (3, 2)	0	0	0	0	2	2	2	2	2
2 (4, 3)	0	0	0	2	3	3	3	5	5
3 (5, 4)	0	0	0	2	3	4	4	5	6
4 (6, 1)	0	0	0	0	2	3	4	5	6

(sorted)

8−5 = 3
3−3 = 0

$m[i,w] = \max(m[i-1, w], m[i-1, w-w[i]] + p[i])$

$m[4,7] = \max(m[3,7], m[3, 7-6] + 1)$

$= (5, 0+1) \Rightarrow (5, 1) \Rightarrow 5$

9kg — 5kg 2kg

$\max(4+0, 3)$ $\max(4+0, 5)$
$\max(4+0, 3)$ $\max(4+2, 5)$

$x_i = \{1, 0, 0, 1\}$ 6 − 4 = (2) − 2 − 2 = 0

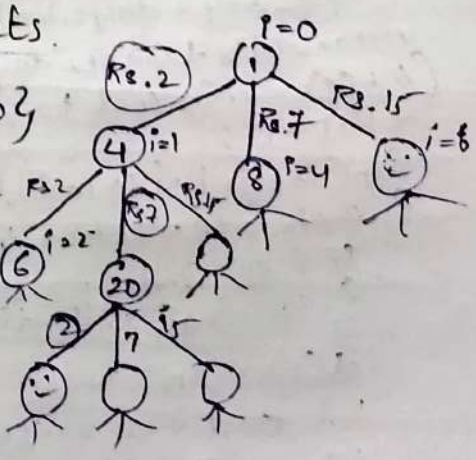
3 4 6 5 = 8

4.(983) Minimum cost for Tickets

Input :- days = {1, 4, 6, 7, 8, 20} (1 2 3 4 5 6)

costs = {2, 7, 15}

1-day pass	7-day pass	30-day pass



o/p :- 11

11

4+7 = 11

<11

(Rec with memo)

```
int t[366];
int Solve(days, costs, n, i)
{ if(i>=n) return 0   // you can't travel, so no cost
  if(t[i]!=-1) return t[i];
  // 1 day pass
    int Cost_1 = cost[0] + solve(days, costs, n, i+1);
  // 7 day pass
    int j=i;
    int max_day = days[i]+7;      // 4+7=11
    while(j<n && days[j] < max-day){
        j++;
    }
    int cost_7 = cost[1] + solve(days, costs, n, j);
  // 30 day pass
    j = i;
    max-day = days[i] + 30;
    while(j<n && days[j] <max-day){
    }   j++;
    int cost_30 = cost[2] + solve(days, costs, n, j);
}   return t[i] = min({cost_1, cost_7, cost_30});
memset(t, -1, sizeof(t));
solve(days, costs, n, 0);
```

Bottom-up

days: | 1 | 4 | 6 | 7 | 8 | 20 |

Bottom up: lastday = 20

n=21 t: 0 1 2 3 4 5 6 7 .. 20

t[21]

st: {1, 4, 6, 7, 8, 20}

```
int minCostTickets(days, costs)
{ unordered_set<int> st(begin(days), end(days));
  int last_day = days.back();
  vector<int> t(last_day+1, 0);
  (// t[i] = min cost to travel till day i of your travelplan)
  t[0] = 0;
  for(int i=1; i<=last_day; i++)
  { // check if you have to travel on ith day or not
    if(st.find() == st.end())
    { continue; t[i] = t[i-1]
    }
    t[i] = INT_MAX;
    int day1_pass = costs[0] + t[max(i-1, 0)];    // if i-1 <0 t[0]
    int day7_pass = costs[1] + t[max(i-7, 0)];    // 8-7=-1 to not get neg keep 0 t[0]=0
    int day30_pass = costs[2] + t[max(i-30, 0)];
  } t[i] = min({day1_pass, day7_pass, day30_pass});
}  return t[last-day];
```

5. (322). Coin change

T.C - O(amt * len(coins))

S.C - O(amt)

(1143) Longest common subsequence

6 Q) Given two strings, find the length of the longest common subsequence (order)

a b c d a f → abc / abd / abdf "abcf"

a c b c f

4 - Ans

		a	b	c	d	a	f
	0	0	0	0	0	0	0
a	1	1	1	1	1	1	1
c	1	1	2	2	2	2	2
b	1	2	2	2	2	2	2
c	1	2	3	3	3	3	3
f	1	2	3	3	3	4	→ Ans

"ab" "ac"

a b c d a f

a c b c f

a, b, c, f ..

(where is this 3 come from)

T.C - O(m*n)

S.C - O(m*n)

Longest common Substring

(length of) b/w 2 strings

"z f b c d z b"

"z b c d f"

Ans → ③ "bcd"

"z f b b"

"b"

① "b"

"a b c x y z a"

"x y z a b c b"

④ "x y z a"

(If both are same add 1 to diagonal opposite)

(If not put '0')

		z	f	b	c	d	z	b
	0	0	0	0	0	0	0	0
z	1	1	0	0	0	0	1	0
b	2	0	0	1	0	0	0	2
c	3	0	0	0	2	0	0	0
d	4	0	0	0	0	3	0	0
f	5	0	1	0	0	0	0	0

Each ele is telling me, the longest common substring ending at that particular position.

```
int longestCommonSubstring(String m, n)
{   s1 = m.toCharArray();
    s2 = n.toCharArray();
    int[][] dp = new int[str1.length+1][str2.length+1];
    int max = 0;
    // Iterate over each position in the matrix
    for(int i=1; i<=str1.length; i++)
    {   for(int j=1; j<=str2.length; j++)
        {   // If chars are equal
            if(str1[i-1] == str2[j-1])
            {   // Get the no. from diagonally opp
                dp[i][j] = dp[i-1][j-1] + 1;
                max = Math.max(dp[i][j], max);
            }
        }
    }
    return max;
}
```

7. Max Length of Repeated Subarray

Given two integers arrays, find the len of the max subarray that appears in both of them.

[1,2,3,2,1] | [0,0,0,0,0]
[3,2,1,4,7] | [0,0,0,0]

3 → ans | 4 → ans

Converting To Longest common substring

abcba
cbadg (same as longest common substring)

1 → a
2 → b
3 → c
:

		1	2	3	2	1
		0	1	2	3	4
3	0	0	0	1	0	0
2	1	0	1	0	2	0
1	2	1	0	0	0	3
4	3	0	0	0	0	0
7	4	0	0	0	0	0

- If both row & col characters are diff, put 0 in the matrix
- If both are same, copy the value from top left cell and add 1 to it.

8. (72) Edit Distance

Given two strings, return the min no. of operations required to convert string 1 to string 2 (insert, del, substitute)

"horse"
"ros"
remove 'r'
replace 'h' with 'r'
remove 'e'

horse → rose → ros → Ans 3

(only 1 operation at once).

- Insert 'x' at 3rd pos → Inxtention
- Delete first 't' → Inxention

"intention"
"execution" 5

(remove 't')
(replace 'i' with 'e')
(replace 'n' with 'x')
(replace 'n' with 'c')
(insert 'u')

intention → enention → exention → exection → execution

Brute force - "abcdef" → "abcde"

recursively pass func after each op not optimal

horse
ros
3+1 = 4

"abcdef"
"azcde"

		a	b	c	d	e	f
	0	1	2	3	4	5	6
a	1	0	1	2	3	4	5
z	2	1	1	2	3	4	5
c	3	2	2	1	2	3	4
d	4	3	3	2	1	2	3
e	5	4	4	3	2	1	2

ans

ab (1)
az
ab
az (1)

2 2,2
ab
az
a (0)
az

```
int minDistance(String w1, String w2)
{ m = word1.length();
  n = word2.length();
// dp[i][j] := min operations to convert word1 to word2.
int[][] costDP = new int[m+1][n+1];
// Initialize DP matrix
for(int i=1; i<=m; ++i) costDP[i][0] = i;
for(int j=1; j<=n; ++j) costDP[0][j] = j;

for(int i=1; i<=m; ++i)
  for(int j=1; j<=n; ++j)
    if(w1.charAt(i-1) == w2.charAt(j-1))
      // copy from top left
      costDP[i][j] = costDP[i-1][j-1];
    else
    { // Get min of all 3 neighbours
      int topleft = costDP[i-1][j-1];
      int top = costDP[i-1][j];
      int left = costDP[i][j-1];
      costDP[i][j] = Math.min(topleft, Math.min(top,left))+1;
    }
return costDP[m][n];
}
```

9. (300) Longest Increasing Subsequence

[10, 9, 2, 5, 3, 7, 101, 18]

10, 101

2, 5, 7, 101

4 -> Ans

[0, 1, 0, 3, 2, 3]

0, 3

0, 2, 3

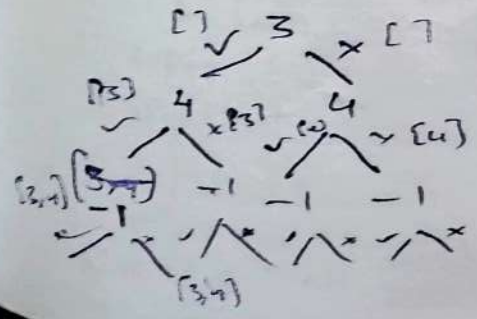
0, 1 2, 3

4 -> Ans

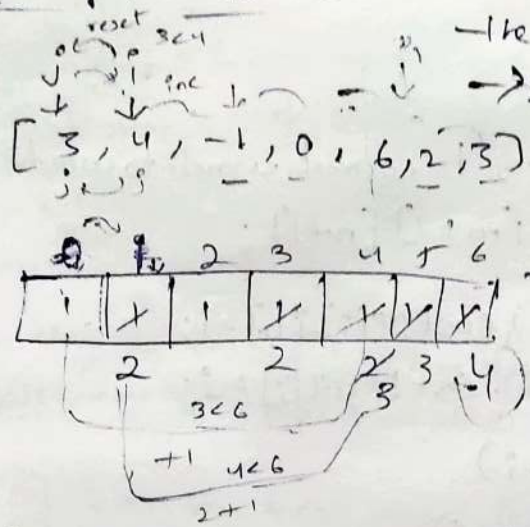
[7, 7, 7, 7]

1 -> Ans

[3, 4, -1, 0, 6, 2, 3]



Using memoization



→ Each element in arr represents the length of the LIS ending at that idx.

→ If element at "j" < element at "i", the length of LIS at i, is atleast 1 greater than length of LIS at "j".

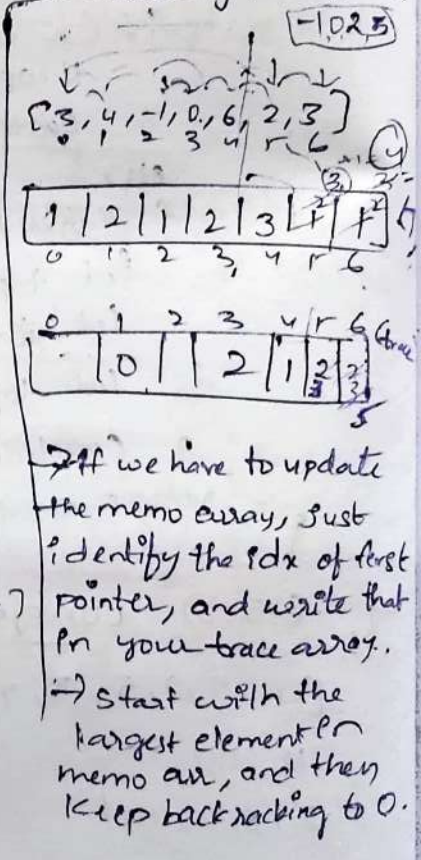
→ If element at 'i' > element at "i", then don't do anything

T.C – $O(n^2)$
S.C – $O(n)$

Actually finding the sequence

```
lengthOfLIS(nums)
{ int T[] = new int[nums.length];
  // start main ptr
  for(i=1; i<nums.length; i++)
  { // start sec ptr
    for(j=0; j<i; j++)
    { if(nums[i] > nums[j])
        if(T[j]+1 > T[i])
          T[i] = T[j] + 1
    } }
  find maxvalidx :- (maxIdx=0
  for(i=0 to T.len())
    if(T[i] > T[maxIdx])
      maxIdx = i
  return T[maxIdx] + 1;
```

ans



→ If we have to update the memo array, just identify the idx of first pointer, and write that in your trace array.

→ Start with the largest element in memo arr, and then keep backtracking to 0.

10. palindrome partitioning

11. (152) Maximum product subarray

(find contiguous sub-array with largest prod)

[2, 3, -2, 4] — Ans -> 6 | [-2, 0, 1] — 1 | [2, 3, -2, -5, +6, -1, 4] — 360

case 1: All numbers are positive — e.g.: [2 | 3 | 4 | 2 | 5 | 6 | 4]

case 2: Both '+ve' & '-ve' numbers — [2 | 3 | -2 | -5 | 6 | -1 | 4]

- Even count of neg numbers
- odd " " " "

case 3: positive, negative and 0 — [2 | 3 | 0 | -5 | 6 | -1 | 4]

[2 | 3 | -2 | -5 | 6 | -1 | 4]

start from left	start from right	max so far
2	4	4
2×3 = 6	4×-1 = -4	6
6×-2 = -12	-4×6 = -24	6
-12×-5 = 60	-24×-5 = 120	120
60×6 = 360	120×-2 = -240	360
360×-1 = -360	-240×3 = -720	360
-360×4 = -1440	-720×2 = -1440	360 ans

leftprod = x
rightprod = x
ans = x6

case of zero: [2 | 3 | 0 | -5 | 6 | -1 | 4]

Start from Left	start from right	max so far
2	4	4
2×3 = 6 [2,3] 6	4×-1 = -4	6
6×0 = 0 → (1)	:	:
1×-5 = -5 (fresh start - [-5, ...])		

```
int maxProduct (int[] nums)
{ int n = nums.length;
  "  leftProd = 1;
  "  rightProd = 1;
  "  ans = nums[0];
  for (int i=0; i<n; i++)
  { leftProd = leftProd==0 ? 1 : leftProd;     // if leftProd == 0: leftProd = 1
    rightProd = rightProd==0 ? 1 : rightProd;
    leftProd *= nums[i];
    rightProd *= nums[n-1-i];
  } ans = Math.max(ans, Math.max(leftProd, rightProd));
} return ans;
```

O(n)
O(1)

12. (95) Unique Binary Search Trees II

```
mp = {} //map (memo)

Solve (int start, int end)
{
  if (start > end) {
    return {NULL};
  }
  if (start == end)
  { TreeNode* root = new TreeNode(start);
    return {root}; mp[{start, end}] = {root};
  }
  if (mp.find({start, end}) != mp.end())
  { return mp[{start, end}];
  }
  vector<TreeNode*> result;
  for (int i = start; i <= end; i++)
  { vector<TreeNode*> left_bsts = solve(start, i-1);
    vector<TreeNode*> right_bsts = solve(i+1, end);
    for (TreeNode* leftRoot : left_bsts)
    { for (TreeNode* rightRoot : right_bsts)
      { TreeNode* root = new TreeNode(i);
        root->left = leftRoot;
        root->right = rightRoot;
      } } result.push_back(root);
```

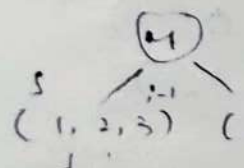
$T\left(n * \left(\frac{4^n}{n^{1.5}}\right)\right)$ (catalan no.) $n \times m$,

1 2 3 (4) 5 6 — start, i, end

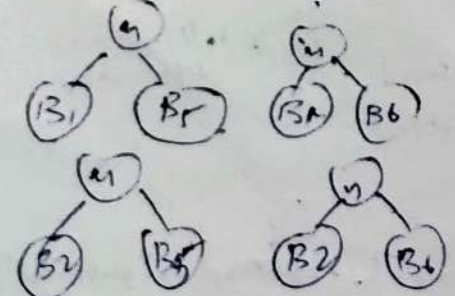
L = (start, i-1);

R = (i+1, end);

{2 (4) root -}



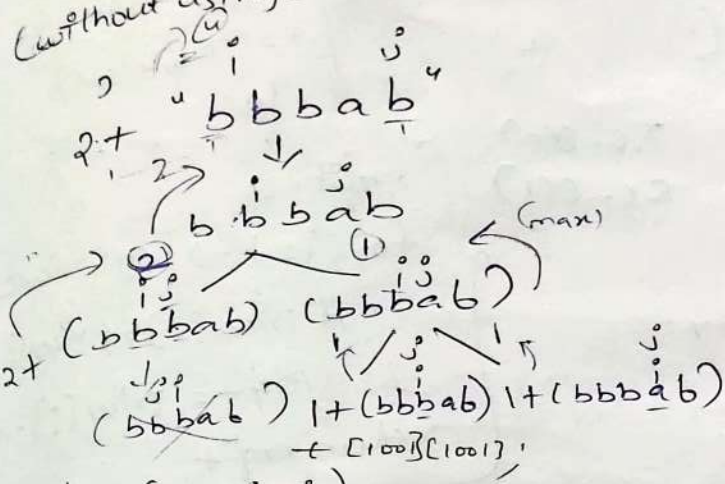
{B1, B2, B3 ...} {B5, B6}




```
left_BSTS
{ for (left_BST) {
    for (right_BST)
    {
  } }
```

13. (516) Longest Palindromic ~~Substring~~ Subsequence

(without using LCS)



bbbab
babbb - (4)

```
solve (s, i, j)
{  if (i>j)
      return 0;
   if (i==j)                 if (t[i][j] != -1)
      return 1;                  return t[i][j];
   if (s[i]==s[j])
                t[i][j]=
      return 2+ solve(s, i+1, j-1);
   else    t[i][j]=
      return max (solve(s, i+1, j), solve(s, i, j-1)).
}
```

(with LCS) - (6) check Q(1143)

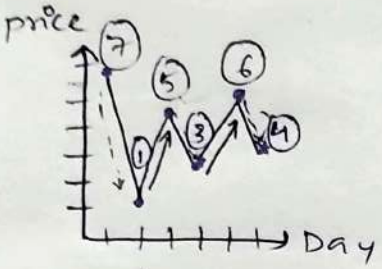
```
s1 - - -
s2 = reverse - (s1)
longestCommonSubsequence(s1, s2)
{  m = s1.length()
   n = s2.length()
vec<vec<int>> t (m+1, vector<int>(n+1));
   for (i=0 -> m+1)
   {  for (j=0 -> n+1)
        if (i==0 || j==0)
           t[i][j]=0;
        else if (s1[i-1] == s2[j-1])
           t[i][j] = 1+ t[i-1][j-1]
        else
           t[i][j] = max (t[i][j-1], t[i-1][j]);
   }
   return t[m][n];
}
```

14. (122) Best time to buy & sell a stock - II

Prices = [7, 1, 5, 3, 6, 4] (buy, sell, buy, sell)
profit = 4 + 3 = 7

P = [1, 2, 3, 4, 5] : (4)

P = [7, 6, 4, 3, 1] (can do as many transactions as you like) : (0)



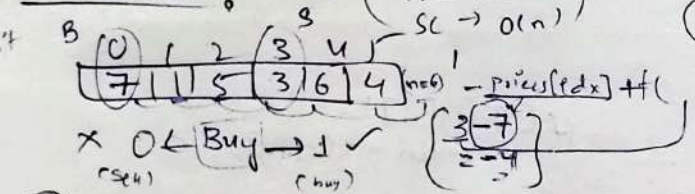
profit → 5-1=4
6-3=3
(7)

T.C - O(n)
S.C - O(1)

```
profit = 0
for(i = 1 -> len(prices))
    if prices[i] > prices[i-1]:
        profit = prices[i] - prices[i-1]
return profit
```

Recursion:

T.C → 2^n
S.C → O(n)



(overlapping) → memorization

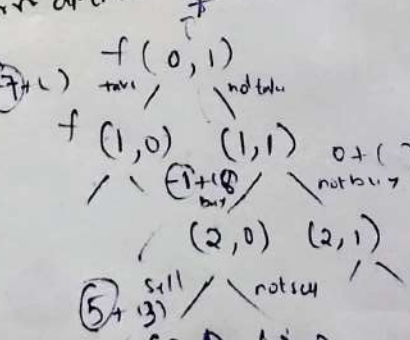
ind, buy → dp[N][2]

```
f(ind, buy)
{
  if(ind == n)
    return 0;
  if(dp[ind][buy] != -1)
    return dp[ind][buy];
  if(buy)
  {
    profit = max{ -prices[ind] + f(ind+1, 0),   // take
                  0 + f(ind+1, 1) }             // not take
  }
  else
  {
    profit = max{ prices[ind] + f(ind+1, 1),    // sell and change to buy
                  0 + f(ind+1, 0) }             // sell next day
  }
  return dp[ind][buy] = profit
}
```

// [7,6,3,1] even though you buy stock and didn't sell prof is 0 (change to tabulation)

dp[ind+1][0] change to sell

dp[ind+1][1] buy next day



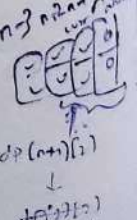
TC → O(N×2)
SC → O(N×2) + O(N) (recursion stack)

Tabulation

```
dp[n][2] = {0}
for(ind = n-1 to 0)
  for(buy = 0 to 1)
    { recursion code }
```

Rec → 0 → n (reverse)

1. Base case
2. i → (n-1 to 0)
   buy → 0 → 1
3. Copy the recurrance



dp[ind][buy] = prev & curr

recursion ahead(1,0), cur(2,0);

```
getMaxProf(values, n)
{ vector<long> ahead(2,0), curr(2,0);
  ahead[0] = ahead[1] = 0;
  for(int ind = n-1; ind >= 0; ind--)
  { for(int buy = 0; buy <= 1; buy++)
    { long profit = 0;
      if(buy)
      { profit = max((-values[ind] + ahead[0]), (0 + ahead[1]));
      }
      else
      { profit = max(values[ind] + ahead[1], 0 + ahead[0]);
      }
      curr[buy] = profit;
    }
    ahead = curr;
  }
  return ahead[1];
}
```

(DP) Python

```
def maxProfit(self, prices):
    n = len(prices)                    O(n)
    if n == 0:                         O(1)
        return 0
    dp_hold = -prices[0]
    dp_free = 0
    for i in range(1, n):
        dp_hold = max(dp_hold, dp_free - prices[i])   # buy or keep holding
        dp_free = max(dp_free, dp_hold + prices[i])   # sell or stay free
    return dp_free
```

15. Catalan Number

1	1	2	5	14				
0	1	2	3	4	5	6	7	8

dp

$C_0 \to 1$
$C_1 \to 1$
$C_2 \to C_0 . C_1 = 1.1 + 1.1 = 2$
$C_3 \to C_0 . C_2 + C_1 . C_1 + C_2 . C_0 = 1.2 + 1.1 + 2.1 = 5$
$C_4 \to C_0 C_3 + C_1 C_2 + C_2 C_1 + C_3 C_0 =$

i, j, i-j-1

4 (0, 4-0-1 = 3)

n => input()

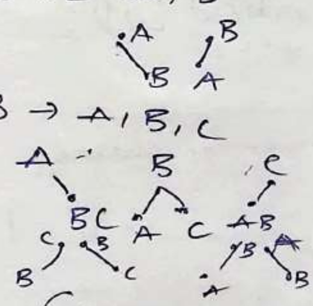
```
int dp[] = new int[n+1];
dp[0] = 1;
dp[1] = 1;
for(int i=2; i<dp.length; i++)
{ for(int j=0; j<i; j++)
  { dp[i] += dp[j] * dp[i-j-1];
  }}
return dp[n];
```

1, 1, 2, 5, 14, 42, 132 - - -

$$C_n = C_0 C_{n-1} + C_1 C_{n-2} + \dots C_{n-1} C_0$$

→ No. of BSTs: (C_n)
- 0 → 1
- 1 → 1   A
- 2 → 2   A, B

- C3 → A, B, C



$(C_3 = C_0 C_2 + C_1 C_1 + C_2 C_0)$

- C4 → A, B, C, D

A: BCD   B: A CD   C: AB D   D: ABC

$C_0 C_3 + C_1 C_2 + C_2 C_1 + C_3 C_0$

→ Unlabeled Trees: (C_n)
C_n → Unlabeled trees
$C_n \times n!$ → labeled trees

→ N paranthesis:
- 0 → 1
- 1 → 1 ()
- 2 → 2 (()) ()()
- 3 → (₀) (₂), (₁) (₁), (₂) (₀)

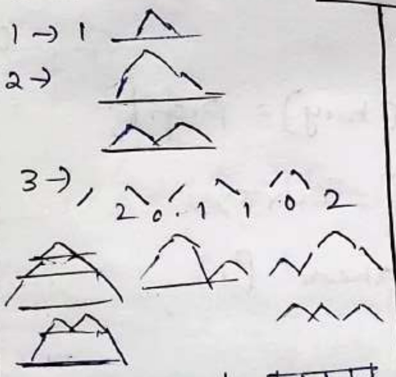
(()) ( ), (()()), ((()))

$= C_0 C_2 + C_1 C_1 + C_2 C_0 = 5$

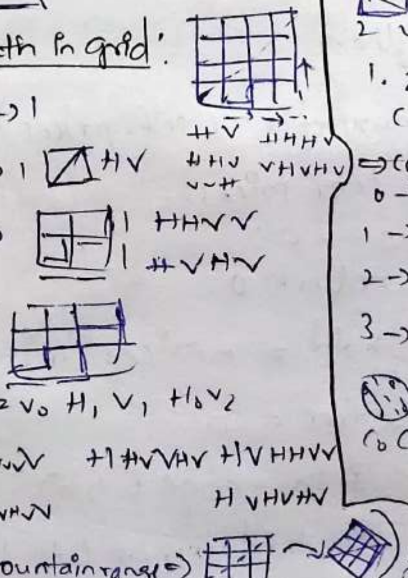
$(_0)_2$ ⇒ ()(()), ()()()
$(_1)_1$ → (())()
$(_2)_0$ ⇒ ((())), (()())

(a,b)((c d)), (a b)(c)(d)
((a b)) ((d))
(((a b (d))), ((ab) ((d))

→ Mountain ranges:
- 0 → 1
- 1 → 1
- 2 →
- 3 →



→ Path in grid:
- 0 → 1
- 1 → 1 HV
- 2 → HHVV, HVHV
- 3 →



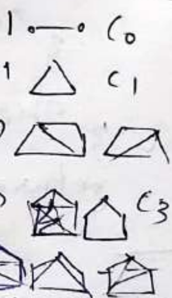
HHHVVV HHVHVV HHVVHV HVHHVV HVHVHV

$H_2 V_0$ $H_1 V_1$ $H_0 V_2$

→ mountain range ⇒ grid

DYCK words - C_n

→ Convert polygon C_{n-2}
- 2 → 1 C_0
- 3 → 1 C_1
- 4 → C_2
- 5 → C_3

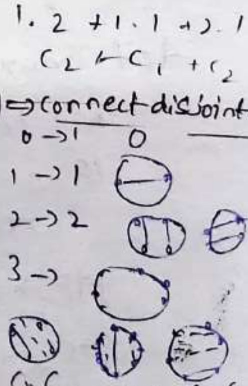


2 4 3 3 4 2

1.2 + 1.1 + 2.1

$C_2 + C_1 + C_2$

⇒ Connect disjoint chords
- 0 → 1
- 1 → 1
- 2 → 2
- 3 →



$C_0 C_2$ $C_1 C_1$ $C_2 C_0$

(with letters)
- 0 →
- 1 → x.y
- 2 → xxyy, xyxy
- 3 → xy xy xy

xxyxyy xxyyxy xyxxyy
xxxyyy xyxyxy

$C_2 C_0$ $C_1 C_1$ $C_0 C_2$

→ catalan Triangulation

	0	1	2	3	4	5	6
0	1	1					
1	1	2	2				
2	1	3	5	5			
3	1	4	9	14	14		
4	1	5	14	28	42	42	
5	1	6	20	48	90	132	132

16. Partition A Set into Two Subsets with minimum absolute sum difference (gfg)

[1, 2, 3, 4]

{1, 2} {3, 4} = |7 − 3| = 4
(3) (7)

{1, 3} {2, 4} = |6 − 4| = 2
(4) (6)

{1, 4} {2, 3} = |5 − 5| = 0
(5) (5)

dp[4][7] ⇒ ?

	0	1	2	3	4	5	6	7
0	T	T						
1	T	✓	✓	✓	✓	✓	✓	✓
2	T	✓	✓	✓	✓	✓	✓	✓
3	T	✓	✓	✓	✓	✓	✓	✓
4	T	✓	✓	✓	✓	✓	✓	T/F → ans

Subset → yes/no, Tabulation

If we check for a target (k)
- we can determine if every possible target b/w (1 & k) is ✓/×

totalsum 2 (12) {3, 2, 7}

(2) (3,7)
S_1 S_2

(2) 12 − 2 = (10)

dp[3][12+1]

✓	×	✓	✓	×	✓	×	✓	×	✓	✓	×	✓
0	1	2	3	4	5	6	7	8	9	10	11	12

S_1 ⇒ 0 2 3 5 | 7 9 10 12 (target − S1 = S2)

S_2 ⇒ 12 10 9 7 | 5 3 2 0 } → abs diff

12 8 6 2 | 2 6 8 12

min = (2) ✓

(arr)
|
Subset + (sum = target (k)) — revision

```
f(ind, target)      // ind = n-1
{ if (target == 0) return True;
  if (ind == 0) return (arr[0] == target);
      // if it is last element next tar should be 0.
  not_take = f(ind-1, target)
  take = false
  if (arr[ind] <= target)
     take = f(ind-1, target-arr[ind])
} return take | not_take;
```

⇓

Tabulation

```
dp[n][target]

for (ind = 0 -> n-1)          // for target = 0 is possible
    dp[ind][0] = true;        // b.c
if (arr[0] <= target)
    dp[0][arr[0]] = true;

for (i = 1 -> n)
{ for (tar => 1 -> k)
  { nottake = dp[i-1][target];
    take = false;
    if (arr[i] <= target)
    { take = dp[i-1][target - arr[i]]
    }
    dp[i][target] = take or notTake;
}}
```

(tar s1 s2, s_2 = t − s_1)

```
mini = 1e9
for ( i=0 -> totsum/2
{  if (dp[n-1][i]==T)
   {
      S1 = i ;
      S2 = totsum - i ;
      mini = min(mini , abs(S2-S1));
   }
}
```

(44) Wildcard Marketing

S_1 = "xdy" ? → matching with single char

S_2 = "xxdy" * → Matches with sequence of length 0 or more

(lowercase)

S1 = "ab*cd"
S2 = "abdefcd" (True)

S1 = "**abcd" (T)
S2 = "cbcd"

S1 = "ab?d" (F)
S2 = "abcc"

S1 = "ab*cd"
S2 = "abdefcd"

S1 [0 to 4]
S2 [0 to 6]

(i-1,j) ab* | j
abdef (i,j-1)

ab | abdef   ab* | abd

ab | abde   ab* | abd

ab | abd   ab* | ab

ab | ab   ab* | a

ab | a

```
wildcardMat(pattern, text)
n, m ; dp[n+1][m+1] -> false
dp[0][0] = true;
for (int j=1; j<=m; j++){
} dp[0][j] = false
for (int i=1, i<=n; i++)
{bool flag = true;
  for (int k=1; k<=i; k++)
  { if (pattern[k-1] != '*')
    { flag = false;
  } } break;
  dp[i][0] = flag;
}
```

```
① f(i, j)   Recursion   n, m, dp[n][m] -> -1 (both are exausted)
② { if (i<0 && j<0) return true;
    if (i<0 && j>=0) return false;
    if (j<0 && i>=0)   (S2 is remaining)
      for (k=0 -> i)   if dp[i][j] != -1 return dp[i][j]
        if S1[k] != '*' { return false }
        else return true
    if (S1[i] == S2[j] || S1[i] == '?')
      return dp[i][j] = f(i-1, j-1);
    if (S1[i] == '*')
      return dp[i][j] = f(i-1, j) or (i, j-1)
    return dp[i][j] = false;
```

T.C → exponential
S.C → O(N+M)
overlapping subproblem

Recursion → memorization
T.C → O(N×M)
S.C → O(N×M) + O(N+M) Aux. Space

→ Tabulation
S.C → O(N×M)

```
for(int i=1; i<=n; i++
{ for(j=1; j<=m; j++)
{ if(pattern[i-1]==text[j-1] || pattern[i-1]=='?')
  { dp[i][j] = dp[i-1][j-1];
  }
  else if (pattern[i-1]=='*')
  { dp[i][j] = dp[i-1][j] | dp[i][j-1];
  }else dp[i][j]=false;
}}
```

② space optimization

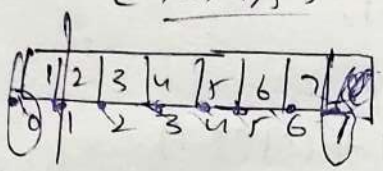
```
{ vector<bool> prev(m+1,false), cur(m+1,false);
  prev[0]=true;
  for(int j=1; i<=m; j++)
  { prev[j]=false;
  }
  for(int i=1; i<=n; i++)
  { bool flag=true;
    for(int k=1; k<=i; k++)
    { if(pattern[k-1]!='*')
      { flag=false
        break;
    }}
  // for every row - you are assigning the 0th col's value
    dp[i][0] curr[0]=flag
    for(int j=1; j<=m; j++)
    { if(pattern[i-1]==text[j-1] || pattern[i-1]=='?')
      { curr[j]=prev[j-1];
      }
      else if(pattern[i-1]=='*')
      { curr[j]=prev[j] | curr[j-1];
      }
      else curr[j]=false;
    } prev=cur;
} return prev[m];
```

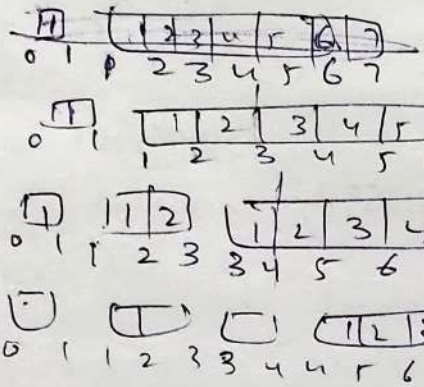
	""	a	b	*	c	d
""	T	F	F	F	F	F
a	F	T	F	F	F	F
ab	F	F	T	F	F	F
abd	F	F	F	T	F	F
abde	F	F	F	T	F	F
abdef	F	F	F	T	F	F
abdefg	F	F	F	T	T	F
abdefgh	F	F	F	T	F	T

18. Minimum cost to cut the stick (1547)

Cost -> length of the stick you are cutting

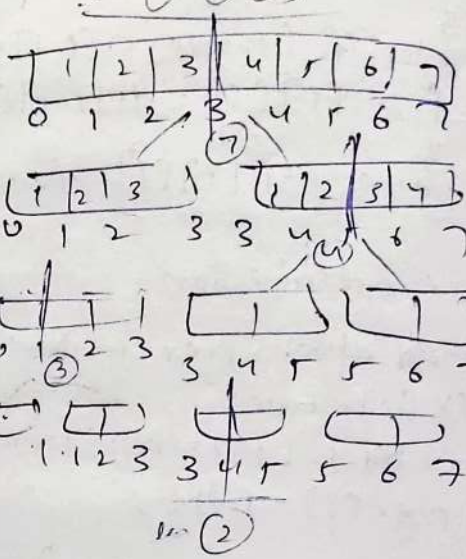
cuts[] = [1,3,4,5] n=7



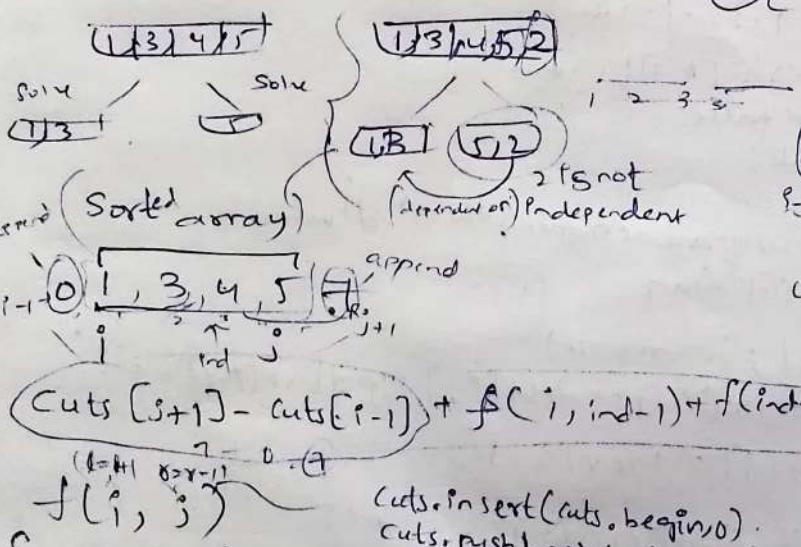


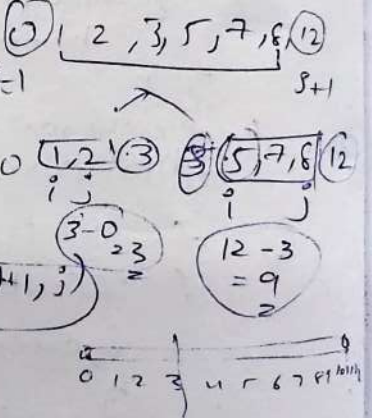
7 + 6 + 4 + 3
= (20)

[3,5,1,4] another case



7 + 4 + 3 + 2
= (16) min





Cuts.insert(cuts.begin(), 0)
Cuts.push_back(n);

```
{ If (i > j) return 0;
  mini = INT_MAX
  for (ind i -> j)
  { cost = (cuts[j+1] - cuts[i-1]) +
           f(i, ind-1) +
           f(ind+1, j)
    mini = min(mini, cost)
  } return dp[i][j] = mini
```

Sort -> cuts
return f(1, (cuts).size() - 1);

If (dp[i][j] != -1)
return dp[i][j]

dp -> [c+1][c+1] -> 0

T.C -> (m² x m) m3

S.C -> O(m²) + aux. Stack space

```
int cost( int n, int c, vector<int> &cuts)
{ cuts.push_back(n);
  cuts.insert(cuts.begin(),0);
  sort(cuts.begin(), cuts.end());
  vector<vector<int>> dp(c+2, vector<int>(c+2,0));
  for(int i=c; i>=1; i--)
  { for(int j=1; j<=c; j++)
    { if (i>j) continue;
      int mini = INT_MAX;
      for(int ind=i; ind<=j; ind++)
      { int cost = cuts[j+1] - cuts[i-1] + dp[i][ind-1] +
                   dp[ind+1][j];
        mini = min(mini, cost);
      }
      dp[i][j] = mini;
  } }
  return dp[1][c];
}
```

---

19. Super Egg Drop (887) { c (critical floor) (where egg break) < >=c - breaks ; <c - survives }

I/p: k=1, n=2 (no. of eggs, no. of floors) — 'e' eggs, 'f' floors

O/p: 2

— Dont DP on floors → DP on moves

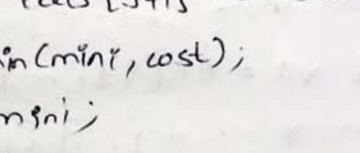
— With 'm' moves & 'k' eggs, how many floors can we guarantee.

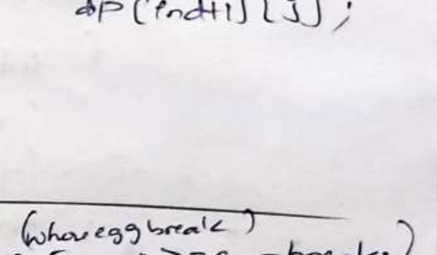
(min no. of attempts/moves that you need to determine with certainty what val of f is)

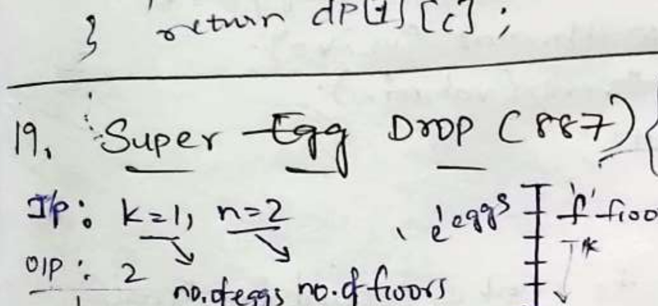
(If egg break no longer use, else reuse it)

choice

Best of the worst / Min of the maxes → luck







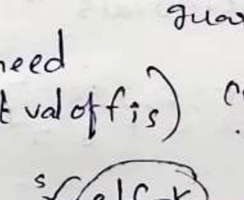
f=0 → 0
f=1 → 1
e=0 → x
e=1 → f

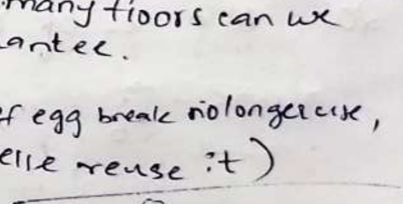
(floors)

	0	1	2	3	4	5	6	7
0	x	∝	∝	∝	∝	∝	∝	∝
1	0	1	2	3	4	5	6	7
2	0	1	2	2	3	3	3	3
3	0	1	2	2	3	3	3	3

max → 2, 2, 2, 3

min → 2 + 1 = 3





3, 3, 2, 3, 4, 5

2 + 1

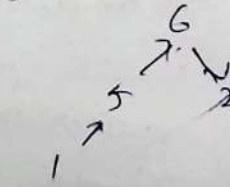
= 3

```
int eggDrop(int n, int k)
{ int [][] dp = new int [n+1][k+1];
  for (int i=1; i<=n; i++)
  { for (int j=1; j<=k; j++)
    { if (i==1)
      { dp[i][j] = j;
      }
      else if (j==1)
      { dp[i][j]=1;
      }
      else
      { int min = Integer.MAX_VALUE;
        for (int mj = j-1; pj=0; mj>=0; mj--, pj++)
        { int v1 = dp[i][mj]; // egg survive
          int v2 = dp[i-1][pj]; // egg break
          int val = Math.max(v1, v2);
          min = Math.min(val, min);
        }
      dp[i][j] = min+1;
      }
  }
  return dp[n][k];
}
```

20. Min no. of removals to make Mountain Array

nums = [2, 1, 1, 5, 6, 2, 3, 1]
idx = 0 1 2 3 4 5 6 7
O/P = 3

i → consider left side strictly increasing pattern
i → consider right side strictly decreasing pattern



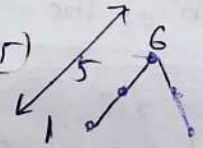
n = 8

LIS = [1, 1, 1, 2, 3, 2, 3, 1] (idx 0 1 2 3 4 5 6 7)

LDS = [2, 1, 1, 3, 3, 2, 2, 1] (idx 0 1 2 3 4 5 6 7)

Removal = n - LDS[i] - LIS[i] + 1

left side:
elements = (i+1) to (i)
LIS[i] = 3
Removal = (i+1) - LIS[i]; // 2



right side:
ele = n - i = 4
LDS[i] = 3
Removal = (n-i) - LDS[i]
4 - 3 = 1

21. (GFG) Matrix chain multiplication

- An optimization problem that finds the most efficient order to multiply a sequence of matrices.
- Goal – minimize the no. of scalar multiplications by choosing the best parenthesization.
- used in – compilers, db query optimization, scientific computing, deeplearning graph execution.

→ New pattern – [partition DP] (Tough pattern)
(solve a problem in partition)

(1+2+3)×(5)
(1+2)·(3×5)

$\begin{bmatrix}1&2\\3&1\end{bmatrix} \times \begin{bmatrix}2\\3\end{bmatrix}$
2×(2)   (2)×1

$\begin{bmatrix}1\times2+2\times3\\3\times2+1\times3\end{bmatrix}$ 2×1

2×2×1 = 4 multiplications

→ MCM
ABC
[A = 10×30, B = 30×5, C = 15×60]
• (A)(BC)
• (AB)(C)

A [ ] 10×(30)   B [ ] (30)×5

no. of C = R ✓

Operations = 10 × 30×5 = (1500)

(AB)(C) = (10×30×5) + (10×5×60)
[ ] [ ]
10×5 (5)×60
(3000)
= 1500 + 3000 = (4500) operations
min ✓

A(BC) = (30×5×60) + (10×30×60)
30×5 5×60
10×30
[ ] [ ]
10×30 30×60
= 9000 + 18000
= (27000)

→ ABCD

A(B(CD)) (AB)(CD) (A(BC))D

Given, arr[] → [10, 20, 30, 40, 50] (indices 0 1 2 3 4) (N=5)
dimension of N-1 matrix

A → 10×20
B → 20×30
C → 30×40
D → 40×50

1st → A[0]×A[1]
2nd → A[1]×A[2]
3rd → A[2]×A[3]
:
ith → A[i-1]×A[i]

→ partition DP → Rules:, basecase ✓
1. start with entire block/array f(i,j) (start, end)
2. Try all partitions → [Run a loop to try all partitions]
3. Return the best possible 2 partitions

→ (A B | C | D) (i = start, j = end)

(AB)(CD) A(BCD) (ABC)(D)
i j i j ij i j i j ij

f(1,n-1)

[10, 20, 30, 40, 50] (indices 0 1 2 3 4, k)
A B C D
i ↓ … j ↓
1 … n-1

f(1,4) → ret min multiplication to multiply (matrix 1 → mat 4)

```
f(i,j)            singlematrix (A) (BCD)    dp[n][n]
{ if(i==j) return 0;                         if dp[i][j] != -1
  mini = 1e9                                  ret dp[i][j]
  for (k → (i) to k → (j-1))
  { steps = (A[i-1] × arr[k] × arr[j])
          + f(i,k) + f(k+1,j)
  } mini = min(mini, steps);
} return mini      dp[i][j] = (min. of all steps)
```

→ k = (i → j, -1)

k = 1 · f(i, k) f(k+1, j)

k = 2 f(1,2), f(3,4)

k = 3 f(1,3), f(4,4)

(or)

k = (i+1 → j)

f(i, k-1), f(k, j)

k = 2 f(1,1), f(2,4)

k = 3 f(1,2) f(3,4)

k = 4 f(1,3) f(4,4)

f(0, 1(i), 2, 3, 4(j))
(10, 20, 30, 40, 50)
A B C D

A | BCD
[20] [30, 40, 50]
i j

AB | CD
20 30 40 50
i k k j

ABC | D
20 30 40 50
i k k j

(10×20×50) + f(1,3) + f(3,4)

BCD
30 40 50
i j

[B] [CD] [BC] [D]

(T.C) Recursion → TC : exponential ✗
- overlapping subproblems ✓
↓
memorization
(partially T.L.E)
f(i, j) (changing variables) dp[N][N]
→ T.C → O(N×N×N) → ≅ O(N³)
SC → O(N²) + O(N)
(auxiliary space)

Bottum-up (Tabulation)

(Smaller prob to bigger prob ↑)

Rules:

1. copy the Base case
2. write down changing states
3. i = (n-1 to 0)
   j = (i+1 to n-1)
3. Copy the Recurrance
   dp[N][N]

f(1,1) → min cost to mul(Mat1 - Mat1)

T.C → N³
S.C → N²

---

int matrixMultiplication (vector<int> &arr, int N)
{ int dp[N][N];
for (int i=1; i<N; i++)
{ dp[i][i] = 0; } // B.C
for (int i=N-1; i>=1; i--)
{ for (int j=i+1; j<N; j++)
{ int mini = INT_MAX;
for (int k=i; k<j; k++)
{ int steps = (arr[i-1] * arr[k] * arr[j])
+ dp[i][k] + dp[k+1][j];
if (steps < mini) mini = steps;
} dp[i][j] = mini;
} return dp[1][N-1];

- A.B → 10×20×30 = 6000
- A.(B.C) → best way to multiply A, B, C = 14000
- (A (B C) D → final optimal order = 38000

i \ j	1 A	2 B	3 C	4 D
1 A	0	A.B 6000	A.BC 18000	A(BC)D 38000
2 B	0	0	B.C 24000	B.(C.D) 64000
3 C	0	0	0	C.D 60000
4 D	0	0	0	0

(312) Burst Balloons

arr[] = {3, 1, 5, 8}

each ith balloon has arr[i] no. on it

calculate max coins u can collect by bursting in any order

⇓

nth → nums[i-1] × nums[i] × nums[i+1]

[3, 1, 5, 8] 1×3×1 = 3

[3, 5, 8] 1×1×5 = 5

[3, 8] 1×5×8 = 40

[8] 1×8×1 = 8

56

[3, 1, 5, 8] 3×1×5 = 15

[3, 5, 8] 3×5×8 = 120

[3, 8] 1×3×8 = 24

[8] 1×8×1 = 8

→ max (167)

→ mcm

{b1 b2 b3 b4 b5 b6}

b3 × b4 × b5 + b1 b2 b3 + b5 b6

f(i) f(i)

can I say this? NO!

[b1 b2 b3 b5 b6]

[b1 b2 b3 b6]

(left & right are interdependent)

f(1, -)

(1, 5, 3, 8)

(1 3 1 5 8 1)

3×1×5 = 15

[3, 5, 8] 3×5×8 = 120

[3, 8] 1×3×8 = 24

[8] 1×8×1 = 8

(go reverse from below)

167

0 1 2 3 4 5

[b1 b2 b3 b4 b5 b6]

i ind j

arr[i-1] × a[ind] × a[j+1] + f(i, ind-1) + f(ind+1, j)

(b1 b4) or [b2 b4] or [b3 b4] or [b4 b5] or [b4 b6]

(b4)

[b4] = 1 × b4 × 1

→ f(i, j)

{ if (i > j) return 0;

if dp[i][j] != -1;

ret dp[i][j]

maxi = INT_MAX

for (ind = i → j)

{ cost = a[i-1] * a[ind] * a[j+1]

+ f(i, ind-1) + f(ind+1, j)

maxi = max(maxi, cost)

} } ret dp[i][j] = maxi

T.C. → 2^n

T(→ n×n×n ≈ n^3, S(→ O(n^2) + stack(dp))

Bottom - up

f(1,n)
↑ ↑
i j

Recursion → Top Down
Tabulation → Bottom up

dp[n+1][n+1] → 0

1. write B.C.
2. i = n → 1
   j = 1 → n
3. copy the recurrance on the recursion

```
int maxCoins(vector<int>& a)
{ int n = a.size();
  a.push_back(1);
  a.insert(a.begin(),1);
  vector<vector<int>> dp(n+2, vector<int>(n+2,0));
  for(int i=n; i>=1; i--)
  {  for(int j=1; j<=n; j++)
     {  if(i>j) continue;
        int maxi = INT_MIN;
        for(int ind=i; ind<=j; ind++)
        {  int cost = a[i-1]*a[ind]*a[j+1]
                    + dp[i][ind-1] + dp[ind+1][j];
           maxi = max(maxi, cost);
        }
        dp[i][j] = maxi;
     }
  }
  return dp[1][n];
}
```