W1
P2 lets learn linked list

Date ___ / ___ / _____

Saathi

I LINKED LIST FUNDAMENTALS

Linked List Intro

Defragmentation :- Storage in continuous sets to avoid fragmentation

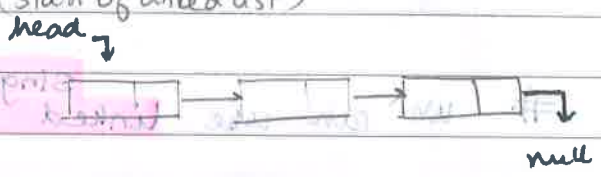
&a → address of
*a → value at

Singly Linked List

Pseudo Code :-

```
struct node {
    data_type variable_name;
    node* next;
    node(): val(0), next(nullptr) {}
    node(int x): val(x), next( ) {}
    node(int x, node* ptr): val(x), next(ptr) {}
};
node* head;
```

(Start of linked list)

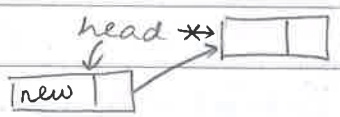


Size/length → 4

Note :- Never lose track of head in linked list codes

Inserting nodes :-

- Inserting at start



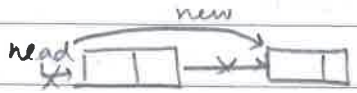
O(1)

- Inserting in between

O(N)

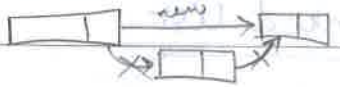
Deleting nodes :-

- Deleting from start



O(1)

- " " b/w



O(N)

freeing nodes

Page No.

Saathi

Date ___ / ___ / _____

IMP Q. Reverse the linked list

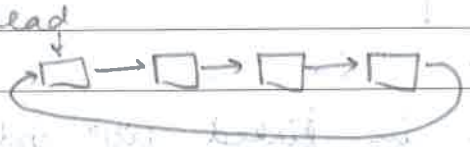
→ Pseudo Code :-

```
void reverse ( node *p ){
    if (p->next == NULL){
            head = p;
            return;
    }
    reverse (p->next);
    p->next->next = p
    p->next = NULL;
}
```

→ Idea of Invariance / Induction

We can use singly linked list as stack

Circular linked list



→ length = steps taken to reach head from head

Note:-

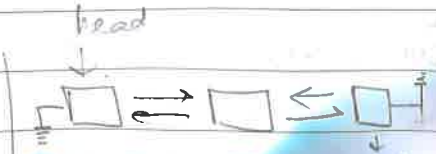
- Visualize
- Check Nulls (next of Null gives seg fault)
- Check Empty linked list as input
- Dummy Node to tackle Null ptr / empty linked list error (NOT RECOMMENDED)

Doubly linked list

Pseudo code :-

```
struct node {
    int data;
    node* next;
    node* prev;
};
```

```
node* head;
node* tail;
```



Saathi

Date ___ / ___ / _____

doubly linked list simulates
- queue
- deque without random access

Reverse → O(1) using a bool variable
Partial Reversal → O(N)

Linked List in STL

List <data_type> variable_name; // doubly linked list

→ insert
→ erase
→ bidirectional iteration

} supported

Sort :- l.sort();

front(), back(), push_back(), push_front(), pop_back(), pop_front(), size(), empty(), clear()

→ all supported

insert() → l.insert(it, 10) // inserts 10 at it
O(1)
1 2 3 4 5 ⇒ 1 10 2 4 5

erase → l.erase(it) // erases value at it
O(1)
1 10 2 4 5 → 1 2 4 5

Other operations :-
- Splice
- Unique
- Merge
- Sort O(N log N)
- reverse

II ATOMIC TECHNIQUES

Atomic Techniques 1

Q. Two LL's start separately but join at a node. Find the node.

→ Hashing one LL → O(L1+L2 log(L1+L2)) TC
O(L1+L2) MC

Counting length } → O(L1+L2) TC
Diff ptr technique } → O(1) MC
(Preferred)

Page No.

Date ___ / ___ / _____

Saathi

Atomic Technique 2

Q. Find middle Node of LL (First middle element in even case)

→ Slow & Fast Pointer Technique

Atomic Technique 3

Q. a) Find if there is a cycle in LL
b) " its start
c) Find its length
d) correct the LL

→ Hashing → Costly on memory → O(N)
× TC → O(N log N)

Doubling Scope Technique :- • Keeping a flag of expected cycle length
↓ ×
(can ans (b, c, d)) • Doubling expectation in each step

TC → O(N), MC → O(1)

upper bound (4y+z) ⟷ we will find cycle in ≤ 24 moves if 4 is the length of cycle
(y non cycle part)

Floyd's Cycle Detection → • Slow & Fast ptrs until the point at same node
• Middle node b/w bumping & start is start of cycle.

TC → O(N) ⟵ as fast ptr moves at most 2 times N
MC → O(1)
IMP (Proof)

& cycle can be at max of len = N. When fast = end, second is at middle & fast has to catch up at max $2 \times \frac{n}{2} = N$

Mixed Example with Codes

Q. Folding a linked list

→ Split in Middle → Reverse the second part → merge alternatively

Q. Unfolding linked list

→ → Put alternate nodes in 2 LL's
→ Reverse second LL
→ join it at the end of first LL

codes very imp

Date ___ / ___ / _____

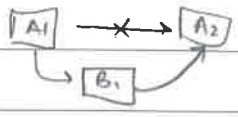
Saathi

III More Practice

Deep Copy Technique

Q. We have node with next & random pts. Create a deep copy

→ • insert new nodes b/w every 2 nodes

• 

rand of $(A_1 \rightarrow next)$ i.e $B_1 = (rand\ of\ A_1) \rightarrow next$

• Split alternative nodes into 2 LL's

XOR Linked List

→ Doubly linked list → In place of next & prev ptr save (Next ^ prev) for each Node

• Typecast node ptr to unsigned long long before taking XOR

• While saving back to node ptr after XOR, typecast to node*

Traversal :- Save XOR values of current & prev nodes

Insert/Delete :- use $x \wedge x = 0$

Use :- • Reversal in O(1)
• Splicing
• Partial Reversal in O(1)

—— X ——

Tough Questions :-

→ Merge LL :- Create a newHead ptr & go on adding nodes to it with the last node in it pointing to null

→ Detect cycle :- • To return null pointer, return 'nullptr' not 'NULL'
• Start slow & fast from head only

→ In case of k linked list merge use divide & conquer to get TC O(N log k)

→ For Node → next ALWAYS use node != NULL

Page No.

W2
Frequently Asked Problems

Date ___ / ___ / _____

Saathi

I. More concepts & Examples

Binary search on Infinite Array (sorted)

→ Doubling Slope technique to find hi & (with/without moving the lo to the prev position of hi)

→ Time for finding hi:

$2^m \geq ans$, (m is no. of times hi moves)

$m \geq \log_2 ans$

TC → O(log ans) as BS also takes O(log ans) time

Shortest Subarray with sum ≥ K (with -ve Array entries)

IMP

→ O(N) Solution.

Maintain Monotone deque of prefix sums

$P_i - P_j \geq K$ → for every P_i we need the closest P_j s.t. smaller the P_j, better it is

Idea:- Maintain Monotone deque of Smallest prefix sm with position.

- As ans is dependent on k, a slightly bigger P_j than previous but closer to i might suffice.
- Also, if a P_j ~~strict~~ suffices P_i, we can remove it from deque as, P_{i+1} would increase the length.

Idea:- Remove elements from deque until we find closest P_j suffices a particular P_i

Majority Element & its variants

Type I:- Finding element with freq ≥ N/2 (Only 1 possible candidate)

→ Brute Force TC → O(N²) MC → O(1) ✗

→ Hashing TC → O(N) MC → O(N) ✗

→ Las-vegas Randomized Idea → Probabilistic Approach → ∞ tries possible

(✓) Expected tries → 2

Page No.

Saathi

Date ___ / ___ / _____

→ Boyer - Moore Voting Algo → TC → O(N), MC → O(1) ✓

IMP

Idea :- • Maintain a candidate ; if encounter same number vote for , else vote against

• else if candidate vote = 0, use current value as candidate

Type - 2 :- Find values with ≥ n/3 freq (2 values possible)

→ Boyer - Moore Algo → TC → O(N) MC → O(1)

Can be extended to n/k

Idea :- • Maintain 2 candidates c_1, c_2

• If $i == c_1$, vote for c_1, if $i == c_2$ vote for c_2

• else if votes of any one candidate = 0, i is taken as candi

• else vote against both candis

Note :- We need to check freq of candis in both cases to confirm if they are the majority element

DP FAQ Live Session I

IMP O(N) Q. Given Array of length N ($A_i \geq 0$). Each element represents max jump len from it. Can we reach N from 1 :-

Q. Convert word 1 to word 2 by replacing, deleting or inserting No. of min operations? O($L_1 \times L_2$)

IMP (Greedy Soln) Q. String matching '?' match 1 char, '*' matches 0 or more chars)

DP FAQ Live Session II

Space Optimization Q. A - 1, B - 2 ... Z - 26 (Encoding) Given an encoding, find the no. of ways it can be decoded into.

→ Optimize space using (& 1) technique TC - O(N), MC - O(1) IMP

Q. EGG DROP → learning.algorenith.com / problems / Dropping - Eggs - 600 } REDO

Q. Knapsack II → At coder

Page No.

Date ___ / ___ / _____

Saathi

Random Problem Solving

Ratio of 0's & 1's in Subarray

Q. Given Array of 0s & 1's, find # of subarrays with (a) #0/#1 = P/Q (b) #0/#1 ≥ P/Q

→ Idea (a) :- #0 × Q − #1 × P = 0 → Replace 0 with Q & 1 with −P

→ Prefix sum and map for finding # of $i < j$ such that $P_i = P_j$

Idea (b) :- #0 × Q − #1 × P < 0 → Same step 1 & prefix sum

→ Inversion count using merge sort

Iterative knapsack with Remove support

→ Idea :- • In knapsack dp[i][j] depends on dp[i-1][j] & dp[i-1][j-wt[i]]

• Also dp[i][j] ≥ dp[i-1][j] depend on dp[i-1][j]

Space optimization to 1D

```
for (i = 0 to n) { // i = level
for (j = W to 0) { // j = wt left,
  dp[j] = 0 if i = 0
  else dp[j] = max (dp[j], dp[j - wt[i]] + val[i])  if j ≥ wt[i]
}}
```

Note:- If we can take an object multiple times, we iterate j from 0 to W as dp[i][j] depends on dp[i-1][j] and dp[i][j-wt[i]]

Q. Given queries:- insert, remove, and find if any numbers add to x or find no. of subsets with sum x

→ Iterative knapsack with queries

insert :- Iterate from j = W to x, (dp[j] += dp[j − x]) if j ≥ x

Remove :- " " j = x to W, (dp[j] −= dp[j − x]) if j ≥ x

dp[0] = 1 (default)

dp(x) → gives no. of subsets with sum x

Page No.

Date ___ / ___ / _____

Saathi

Mixed Problem Discussion I

Q. Slices I & II → /problems/ slices - II - 612 / -613 → Try

Q. Kgp ka vir Diesel → BS Contest 17-18 → Try

Q. Avg Partition :- Given array, divide it in 2 subsets such that their avg is same. (Subset1 ≤ Subset2 in length if not then in less)

→ Sort & Solve using knapsack (DP(i, sum, cnt))

Q. Tree-XOR-Queries → Trie problem (Leet code) → Try

Mixed Problem Discussion II

★ Fenwick Tree → Read up (NOT IMP)

IMP Q. Given K burners in the lab & N gas cylinders with each diff amt of fuel. Find the max burning time of all k burners together. (change can only be done with idle cylinders) [Ball coloring Problem]

→ BS on ans :- Monotone fun :- Gas can contribute to min(x, Ni) ~~cylin~~ times

$(\leq \min(x, Ni)) > x * K$ → return Yes

else No

Q. Partition No.s into 2 sets such that diff is minimum → BS + DP

Q. Find numbers whose digits can be split up as question above, between A & B → A, B ≤ 10^9, B-A ≤ 10^6 → Digit DP → Try

Q. AP Addition → problems/AP-Addition-28 → TRY

W3

Part 2 Algorithmic Design

Date ___/___/_____

Saathi

Stream Mean & Variance

Structure to maintain Mean & Variance

O(1) per insert

→ Variables :- sum(), cnt, sq_sum

→ mean = sum / cnt

→ variance = (sq_sum / cnt) - (mean)^2 $\quad (V(E) = E(x^2) - (E(x))^2$

Mean = $\frac{\Sigma x_i}{N}$, Variance = $\boxed{\frac{\Sigma (x_i - \bar{x})^2}{N}} = \frac{\Sigma x_i^2}{N} - \left(\frac{\Sigma x_i}{N}\right)^2$

$= \frac{\Sigma x_i^2}{N} - (\bar{x})^2$

$$\boxed{V(x) = E(x^2) - (E(x))^2}$$

LRU Cache

least recently Used Cache

MP

Operations to support in the fixed size cache memory

→ get (x) :- Return val at key 'x' if present else ~~set 'x' get~~ return -1

- Set key x as most recent

→ put (x, y) :- If key 'x' is present, update to y ~~else~~

- else insert y with key 'x'. if size of cache is full → remove LRU key
- Update of 'x' as most recent.

Idea :- Use unordered map (O(1)) & doubly linked list.

→ Unordered map (key, (val, itr to listnode)) ‖ DS

→ list → keys saved in order of LRU

get (x) :- O(1)

- find 'x' in map. if not present return -1
- ~~#~~ return val & remove itr node from list. insert x in front and save the new itr at key.second in map

Saathi

Date ___ / ___ / _____

put(x, y) :- O(1)
- find x in map, if present then update, if not then insert x with value y.
- Remove itr from ~~for~~ list if present & push x in front. Save new_itr in x.second of map
- If size of map exceed cache, remove key from the back of list

O(log N) per query Soln :- Maintain map for $key \xrightarrow{x} val \xrightarrow{y}$
- Maintain set I for query no $\rightarrow$ x^{key} to remove x with least query no
- maintain $x^{key} \rightarrow$ query no set II to get the query no. to remove from set I on updation

LFU Cache

Least Frequently Used Cache → Same like LRU but we need to remove key with lowest freq

O(1) Idea :-

Maintain unordered maps of following type

i) key → (val, freq) | ii) freq → (list of the corr. keys)

iii) key → (itr to key in ii)

Whenever we reach a key 'x' we need to update its frequency or recency in both get & put fun^n. Create an update function for the same.

Min Stack Problem

Maintain Stack with normal stack operations + ~~stack~~ giving min of all elements present in stack

× • Recurse on stack To find min → O(N) - TC / query

× • Maintain an extra stack with min of current elements present → O(N) - M(extra)

✓ • Idea :- Maintain a min_element = min. ele

O(1) - TC / query
O(1) - M(extra)

• Push(x)
→ push x if x > min_ele
→ else push (2x - min_ele), set min_ele = x

Date ___ / ___ / ___

Saathi

- Top & pop
  return top if > min_ele, else return min_ele
- Pop
  (if top ≤ min_ele, min_ele = min_ele - top) pop

Stack with Increment → Increment the last 'k' elements with 'x'
→ support push, pop & top

Sol 1
TC → O(N)/query
MC → O(1) extra
→ Maintain stack in vector, Back of vector is top
Traverse through last k elements

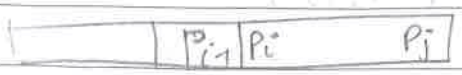
Sol 2
TC → O(1)/query
MC → O(N) extra
- Maintain 2 vectors → 1 for stack, 1 for prefix sum, 0 default
- for increment, add x at k^th position from front as back acts as top. If size < k, add x to back of second vector
- While popping add the increment value to the actual value and pop. move the increment value to the next cell by adding

Product of last K in Stream

→ Maintain DS which can support:
- insert
- give out last k elements product, if k > size give '0'

→ Idea: TC → O(N)/query → traverse on last k & return product
MC → O(1) extra

→ Maintain prefix product → TC → O(1)/query
MC → O(1) if entry are not required later
O(N) otherwise



→ Product of i to j = $\frac{P(j)}{P(i-1)}$

problem:- when there is a 0 at or before P(i-1)

Solution:- • maintain an variable 'zero' that contains the index of last zero added. Insert 1 in place of 0 & maintain the prefix product.

only valid if we can return 0 for k > size of array

Saathi

Date ___/___/_____

- Empty the array on encountering insert '0' as any product beyond '0' ≡ any product beyond size of vector
- If k ≥ zero or k ≥ size of array, return 0 else return the prefix product of valid range

Snapshot Array

Q. Maintain an dynamic array of constant size. Support the following
- Set (i, x) → set index i to value x (0 indexing)
- snap() → take a snap (snap-id = # of snaps - 1)
- get (i, s) → get value at index i of snap-id s.
  If snap id does not exist → return '0'

⟹ Idea 1:-

- maintain vector<vector<int>> as a snap of all versions
- MC → O(N + (snaps) N), TC → set O(1), snap O(N), get O(1)

Idea 2:-

- For each index maintain snap only if its value changes

Eg:-

Snap-id	Val at index i
0	1
1	1
2	1
3	5

⟹ maintained array: [0,1] → [3,5]

- To get value at index (i) at snap-id s, get upper-bound of i → j value at (j--) is returned
- To set (i, x) ⇒ keep updating the value at current snap-id in the array
- for snap(), just increase the snap-id

MC → O((snaps) N) TC :- set → O(1) snap → O(1) get → O(log N)

Week 4 P2

Binary Tree

Date __/__/____

Saathi

- Only (2 children) at max

Implementation

```
Struct node {
    int data;
    node * left, * right;
}
```

§ Binary search tree (BST) :- max(left) < data
O(N) search min ~~max~~(right) > data

Balanced BST (BBST) :- |height(left) - height(right)| <= 1
O(log N) search

Traversal Ideas → Visiting All nodes

Trav(x) → (a) Trav(x → left) (b)
→ Trav(x → right) (c)
} recursive DFS style idea

In Order Traversal :- (b)
Pre " " :- (a)
Post " " :- (c)
→ mark x visited or do any operation

Level Order Traversal :- mark visited / do operation level by level → use BFS

Used for Top view or Bottom view

Q → Connect Nodes on same level

Q. Given Any 2 Traversals from above, comment on uniqueness of tree (nodes ~~dot~~ have unique data)

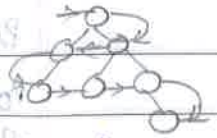
⇒ Inorder + (Anything) → Unique

else → Not unique

Saathi

Date ___ / ___ / _____

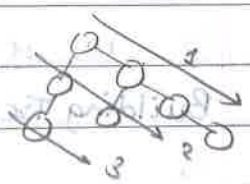
Spiral order Traversal :- level order in zig-zag way



→ Find level order traversal

→ At the end, reverse the order of every alt. level

Diagonal Order Traversal :- Traverse diagonally



→ (0,1) BFS → left cost 1
→ right cost 0

Recover Tree From Pre-In

Q Given Inorder & Pre order Traversal of tree, create the tree

→ Inorder :- left ← 0 1 2 3 4 5 6 7 8 9 10 → Hashing
G H I C J D A F B K E → right

Post Preorder :- A | H G I D J | B F E K → right subtree
↳ root ↳ left subtree

Given a range of In order & Pre order, we can find the root of current range & the left + right subtree elements

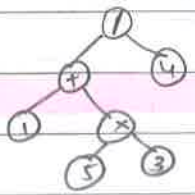
→ we can recursively do this to find whole tree

→ TC → O(N log N) MC → O(N)
↳ Hashing

There exists another stack based idea for which TC → O(N) MC → O(N) ⟹ Idea :- • build the root with right subtree
• ~~join~~ build left subtree & join to the root present in stack.

Expression Tree

Trees built on maths expressions



eg :- (1 + (5 × 3)) / 4

Given an expression tree,

we can find its value by :-

→ If (operator) { val = l = val of left subtree
r = val of right subtree
return (l operator r); }

else return val

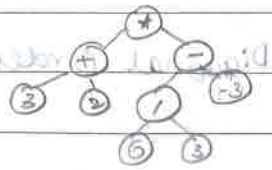
Page No. ____

Saathi

Date ___ / ___ / _____

Pre-Order Traversal → Pre-fix Notation
Post-Order " → Post-fix "
In-order " → In-fix "

Eg:- Post fix:- 3 2 + 6 3 / (-3) - *



Building Tree from Post fix notation

Pseudo Code:-

```
Stack s;
traverse through expression {
  if curr = operator { r = s.top(); s.pop();
                       l = s.top(); s.pop();
                       connect (l) curr (r);
                       push curr;
  }
  else push curr;
}
```

→ Building tree

⇒ If we save the value of each node rather than building a tree, we get VALUE of postfix expression

Converting Postfix to any-fix:- • Build expression tree
• Convert to any fix

While building Tree from infix:- • Consider proper brackets
• " " precedence of operators

BST & BBST

→ Inorder traversal → Gives sorted tree

→ Find x :-

Pseudo Code:-

If curr == x → return Yes, If curr == NULL return NO
else if curr < x → return find(curr → right)
else return find(curr → left);

Date ___ / ___ / _____

Saathi

Checking if a given tree is BST :- • Return (min, max) of each subtree

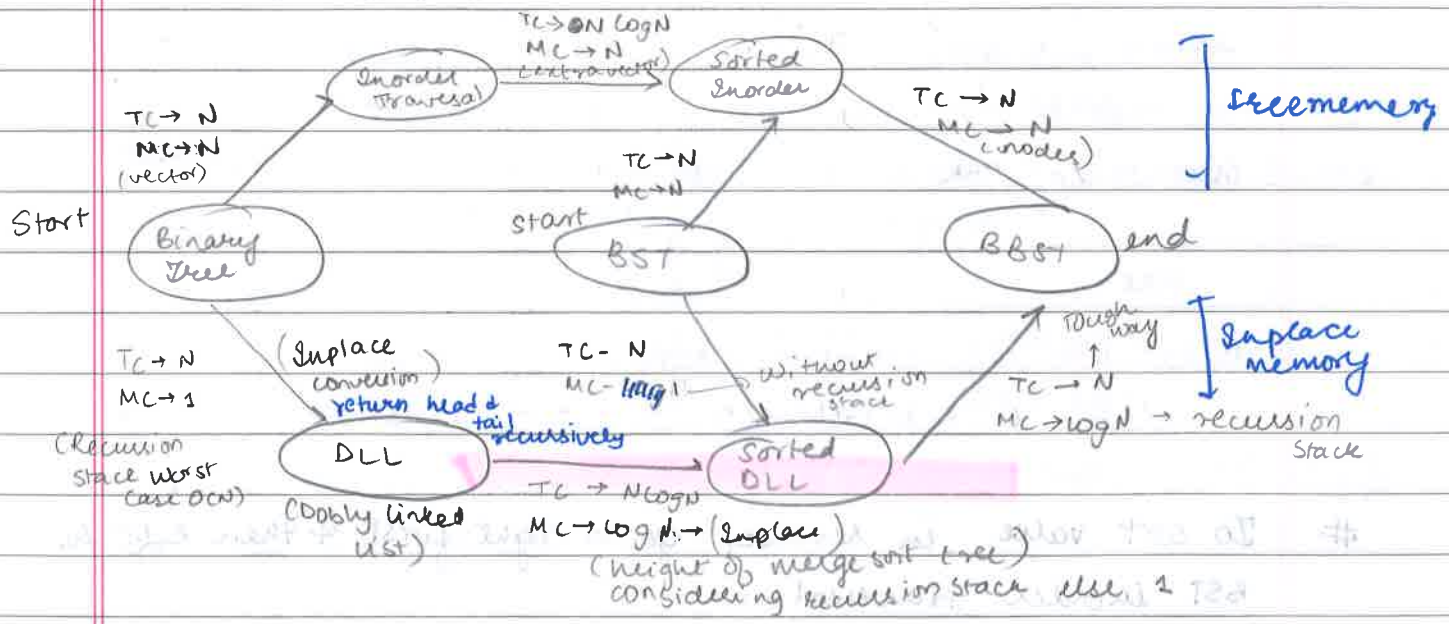
• if min(right) <= curr or max(left) >= curr → No

else Yes

Checking BBST :- • Return height of each subtree

• if |height(left) - height(right)| > 1, No, else Yes

Tree - DLL Interconversion



In Interviews :- Ask interviewer if MC must include memory Stack or not in case of recursion

Merge Sort :- MC - $2\left(N + \frac{N}{2} + \frac{N}{4} \dots\right) \leq 2 \times 2N \rightarrow O(N)$

DDL to BBST :- Complicated Method TC → O(N) → Don't need up

Easy method → TC → O(N log N)

Q Given 2 BST → merge

⟹ Convert to DLL → merge both into 1 DLL → create BST

Page No.

Saathi

Date ___ / ___ / _____

Checking if a given tree is BST :- • Return (min, max) of each subtree

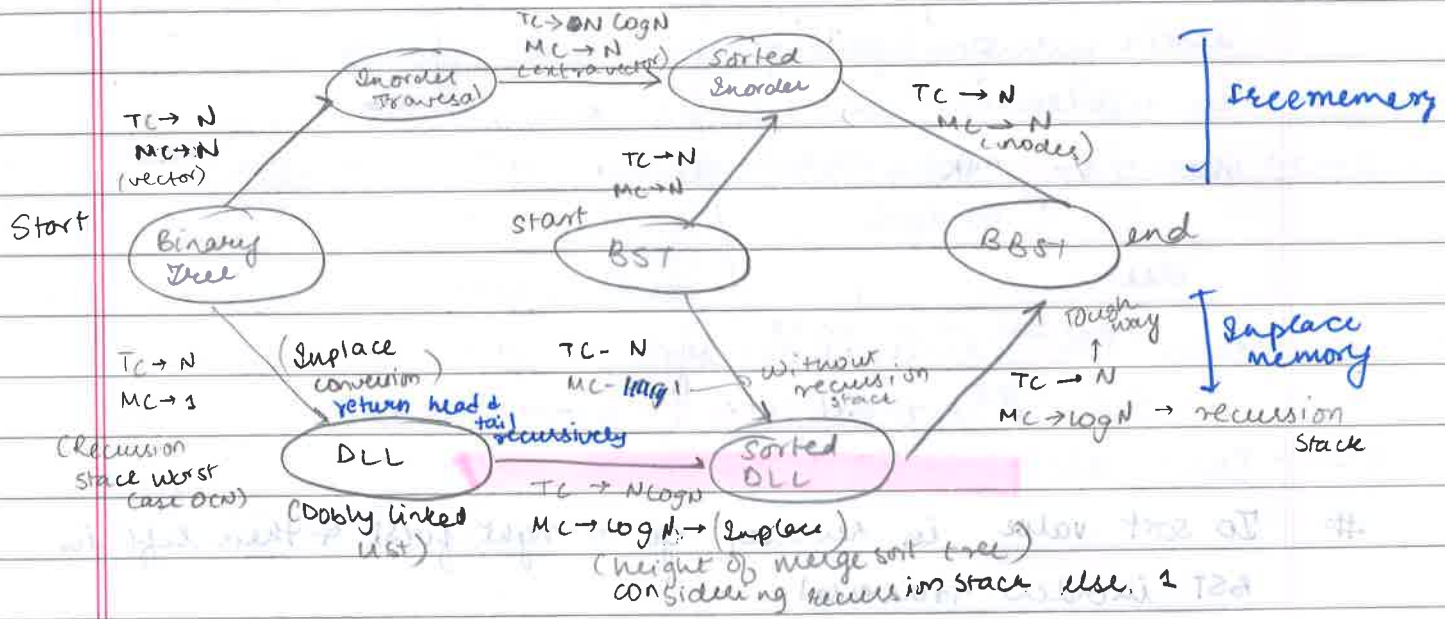
• if min(right) <= curr or max(left) >= curr → No

else Yes

Checking BBST :- • Return height of each subtree

• if |height(left) - height(right)| > 1, No, else Yes

Tree - DLL Interconversion



In Interview :- Ask interviewer if MC must include memory Stack or not in case of recursion

Merge sort :- MC - $2\left(N + \frac{n}{2} + \frac{n}{4} \dots\right) \leq 2 \times 2N \rightarrow O(N)$

DDL to BBST :- Complicated Method TC → O(N) → Don't need up

Easy method → TC → O(N log N)

Q. Given 2 BST → merge

⟹ Convert to DLL → merge both into 1 DLL → create BST

Saathi

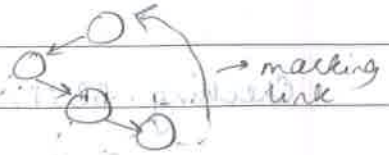
Date ___ / ___ / _____

Morris Order Traversal

TC → O(N) (as each edge is traversed at max 3 times

MC → O(1) (Iteratively)

Idea: Join right most node (inorder) of left child to parent.



Pseudo code:- (Inorder)

```
while cur != NULL
    If left exists
        If marking link inactive:
            • mark marking link on
            • go left
        else
            • mark marking link off (• process this node)
            • go right
    else
        • process this node
        • go right
```

for ~~pos~~ pre order.

Q → Think abt ~~position~~ order' morris traversal ?

To sort value in reverse, go to right first & then left in BST inorder traversal

Q. Find K^{th} largest element in constant space

→ Reverse morris traversal → Finish right child before left

- marking link would be left most node of right ~~of~~ child to parent

Flip a Binary Tree → Flip right & left of every node recursively

Diameter of binary Tree

Idea: For each node find longest path on left child & longest path on right child

diameter = max (left + right + 1) for all nodes

Page No. ____

Saathi

Date ___/___/_____

LCA of Binary Tree (Lowest Common Ancestor)

Q1:- LCA in Binary Tree

⟹ Idea 1:- Maintaining stacks of ancestors & comparing
TC → O(N) MC → O(N)

Idea 2:- Maintain stack till finding the first element
after traversing only nodes sub-tree, pop from stack & move to the other child of the top till we find next element
TC → O(N) MC → O(N)

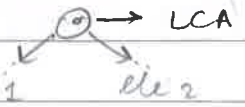
Q2:- LCA in Binary Tree with parent for every node

⟹ If parent are known, it is like Linked list with intersection → Find intersection node TC → O(N) MC → O(1)

IMP

Q3:- LCA in Binary Search Tree

⟹ The node at which both elements lie on different child sub-tree is LCA



TC → O(H) MC → O(1)
(height)
If BBST TC = O(log N)

K^th Element Idea

Q. Given a BST, find K^th number

→ Do in order traversal & find k^th element → TC → O(N) MC → O(1)

Q. Q queries of above kind are given

→ Idea:- Store size of subtree in each node → Store inorder traversal
TC → O(Q log N) MC → O(N) TC → O(Q) MC → O(N)

Q. Find # elements btw n & m

→ Ans = (# elements ≤ n) − (# elements ≤ m) + 1
↳ if can be found using above structure

Page No.

Saathi

Date ___ / ___ / _____

KSum Path

Q Find # paths with sum K from root

→ maintain current sum while doing DFS

Q Find # paths with sum k

→ For every node, get the possible sums till left node in a freq map. Same with right node

- Find → k - node_val) on left map. Add freq - left to ans
  " on right map. " freq - right to ans
- Traverse on left map & find k - (node_val) - (left_map_val) in right map. Add freq_left × freq_right to ans
- Create a map merging left & right maps & adding node_val to all values. Also insert node_val separately. Return to parent node

ML → O(N) (clear previous map in each recursion) else O(N²)

TL → O(N²) (using hash map (unordered map))

Tough Questions

- is BST :- To get min & max → check min & max of both subtrees.
- To check leaf node :- both left & right have to be NULL

Page No. ____

Date ___ / ___ / _____

Kadane's Algorithm

Q. Find largest non-empty sub array sum (ans) in O(N) time.

⟹ Pseudo Code :- m = 0, max_till_now = INT MIN

Traverse on every element of array :-

```
m = m + a[i]
if (m < a[i])
    m = a[i]                    // m is the value of max sub array sum ending at i

max_till_now = max(m, max_till_now)   // max_till_now saves max of all sub array sums
```

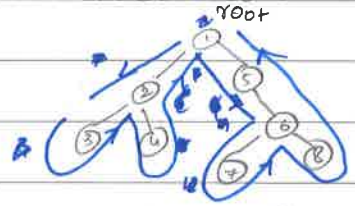
Idea :- maximum sub array sum ending at (i) = max (mss ending at (i-1) + a(i), a(i))

If empty sub-array is allowed, make max_till_now 0 if it is <0 at the end

LCA in a normal tree

- Eulerian path method :- In path b/w x & y, the node with lowest depth is LCA

IMP → Start from root node & explore every node as follows



TC O(N) → Maintain the order of visit & the depth at that order

↑ Preprocessing → For every node, maintain the first time it was visited

↓

MC O(N)

Index :-	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A → order of visiting :-	①	②	③	②	④	②	①	⑤	⑥	⑦	⑥	⑧	⑥	⑤	①
B → depth :-	0	1	2	1	2	1	0	1	2	3	2	3	2	1	0

Build segment tree over B

C → first time node was visited :- ① → 1, ② → 2, ③ → 3, ④ → 5, ⑤ → 8, ⑥ → 9, ⑦ → 10, ⑧ → 11

TC O(logN) per query → To find LCA b/w any 2 nodes x, y, find C[x], C[y]

Eg :- x = 6, y = 2 → C[6] → 9, C[2] → 2

→ find smallest index in range B[C[x]] & B[C[y]]

Here :- 0 at index 7

→ A[y] is the LCA → Here ① is the LCA of 6 & 2

Page No.

Date ___/___/_____

Saathi

Pseudo code for traversal :-

```
DFS (node) :-
vis(node) → True
  for(all children) {
     // Insert depth & node in A, B
     visit child x
  }
  // Insert depth & node in A, B
return
}
//Build C by traversing once in A
```

MEX (Minimum excluded non-negative value)

→ Set Queries → Module 0 → check it out

(only +ve)

Q → Find MEX, when we only have +ve values :- (1 index)

⟹ Observation :- MEX lies b/w 1 & N+1 in an array of size N

Idea :- Go to each index i, if $i \leq N$, Multiply value of at a[i] with -1 index

This ensures that index a[i] is not MEX

- After doing this for all i's, find the i with +ve value, that 'i' is MEX. (If all match, N+1 is MEX)

TC → O(N) MC → O(1)

(only +ve)

Q → Find MEX, when we have +ve, -ve & 0 values :- (1 index)

→ Idea :- for every i, move a[i] to index a[i] by swapping

- Index 'i' with value ≠ i is the MEX
- If all match N+1 is MEX

TC → O(N) MC → O(1)

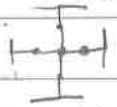
If MEX can be 0 → change the indexing to 0 and change $i \leq N$ to be applicable for non-ve ints

Page No.

Saathi

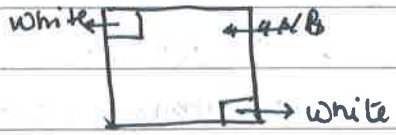
Date ___ / ___ / _____

Chess Board Moves

1. King :- (राजा) (1) — 1 sq at a time 
2. Queen :- (रानी) (1) — multiple squares 
3. Rook :- (हाथी) (2) — multiple squares 
4. Bishop :- (ऊंट) (2) — " " 
5. Knight :- (घोड़ा) (2) — 3 squares at a time 
6. Pawn :- (सिपाई) (8 Pawns)
   - only on sq at a time ↑
   - first move → one or 2 squares ↑
   - To kill, 1 step ↗

Chess-Board :- 8 × 8

Queen on her own color



Order

Rook → Knight → Bishop → Queen, King → Bishop → Knight → Rook

Cards

1. Heart ♥
2. Diamonds ♦
3. Spade ♠
4. Clubs ♣

Red: 13 + 13 = 26
Black: 13 + 13 = 26
52 cards (excluding joker)

A, K, Q, J, 10, 9, 8, 7, 6, 5, 4, 3, 2

Date ___ / ___ / _____

Saathi

Tough Questions :- Algorithmic design

→ Product of last k numbers :- Take long long while *

→ Min Stack :- While accessing top() always check empty()

Imp → ADDMULL → Idea

We have a → addition variable
m → multiplication "
} → For values added till now

to validate a, m, while adding v, add (v-a)/m instead, so that a & m are valid beyond their scope as well

→ LFU Cache :- • Access maps, unorded_maps using mp[index]

• unordered_maps are not sorted

• list → push_back, pop_front, back, front, end, begin ✓
~~rbegin~~ ✗

• While storing list in map:
✓ access → mp[index].begin()
✗ do Not mp.insert(makepair(index,list)) ✗

→ All One :- We can only use un_map, when min_freq/max_freq are continuous like LFU (goes to 1 or jums in interval of 1, 5) else use normal map

→ Circular Queue :- Implementation

Page No.

Saathi

Date ___ / ___ / _____

Syntax :- Module 0

- getline(cin, variable_name);
- s.compare(t);
- Pi = acos(-1);
- for(auto &v : s)

- __gcd(a, b), a, b > 0
- Master theorem :- $a T(\frac{n}{b}) + f_n$
  - $n^{\log_b a}$ compare with f_n (ignore $\log n$ in O or Ω f_n)

- Binary search cannot be performed on list $\rightarrow O(N)$ & un_map
- int, double are hashable → un_map → (only use these as keys)
- m.end() → invalid after decrease in size
- multimap → [] ✗
- Default value of map → 0
- $O > \Theta > \Omega$ → ✗ TC

- next_permutation(a, a+n);
  - ↳ returns false on lexographically largest permutation

MODULE 1 - W1

- for(auto &v : s) → pass by reference
- always use !s.empty() before s.top() → else RE

Questions:-

Module 0:- • Phone Notification, • No. of unique chars query, • dice roll, • twos power, • MEX ≥ x.

- use {x, 1e9} while finding lower/upper limit of pair
- for Multiplication, use LLI to avoid overflow
- Binary exponentiation also works with matrix / float
- $(a \cdot b) \cdot c \neq (a \cdot c) \cdot b$
- rbegin & rend → only work for random iterators
- Before erasing, always find → else RE
- To avoid TLE, pass vectors by reference

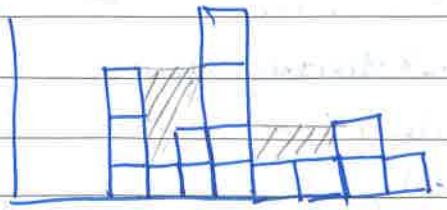
Questions:-

Module 1 W1 :- • Monotone deque, • Buy Max items with budget, • Cards game, • Struct to find mode & median dynamically, • Struct to maintain ranges, count intervals & intersects

Date ___/___/_____

Rainwater Trapping Problem

Q. Given buildings, finds the units of rainwater they trap



→ Idea :- For every index :- amt of water stored above it = min (left tallest building height, right " " ") − current building height

TC - O(N)
MC - O(N)

Store left tallest building height & vice versa for right for every index

original height 3 1 2 4 1 1 2 1

left → 3 3 3 4 4 4 4 4

right → 4 4 4 4 2 2 2 1

water above → min (left(i), right(i)) − height(i)

0 2 1 0 1 1 0 0

Total → 2+1+1+1 = 5 → Ans.

→ space optimization :- maintain left_max & right_max variable O(1) SC

Space optimization :-

maintain left_max & right_max :→ 2 ptr technique

Pseudo code :-

both won't contribute to storing water

```
hi = n-2, lo = 1, ans = 0, left_max = a[0], right_max = a[n-1];
while (hi >= lo) {
    if (left_max < right_max) { // left_max is tighter bound, move lo →
        if (a[lo] < left_max) ans += left_max - a[lo]
        else left_max = a[lo]
    } lo++
    else { // right is tighter bound, move right → left
        if (a[hi] < right_max) ans += right_max - a[hi]
        else right_max = a[hi]
    } hi--;
```

Page No.

Date ___/___/_____

Idea:- • Initiate lo:- 1 & hi:- n-2 as 0 & n-1 wont store water. ~~lol~~

TC-O(N) MC-O(1)
• let left_max = a[0] & right_max = a[n-1]
• if (left_max < right_max), it is a tighter bound so move left → right else otherwise

Another solution:-

Idea:- • Maintain two pts hi = n-1, lo = 0, ans = 0

TC- O(max_ht x P)
• Find max_height & sum_of_all_buildings_heights (a)
• Iterate from 0 to max_height - 1

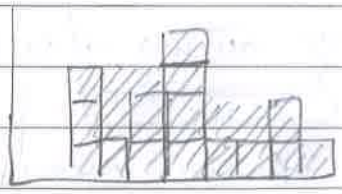
MC → O(1)

```
while (height[lo] <= i) lo++
while (height[hi] <= i) hi--
ans += hi - lo + 1
```

P = (Total no. of changes in heights)

• finally ans -= k

while in for loop, ans save the naked area



MODULE 1 - W2

→ 2D Prefix & Partial Sums | → BS on every start
→ Inversions ⟷ O(NlogN) → Bubble sort & Parity
IMP → Q:- Given 2 arrays, find the k^th smallest pairwise sum
→ BS on real domain → avoiding TLE using fixed iterations
→ Getting ans acc. to accuracy using 'ε'

W1:- Tough Questions.
→ Foundational Maths:- Find if Ax+By=C for some x,y, • No of diagonals/intersections/parts, • Recurring fractions → Applied STL • Collisions

W2:-
→ Prefix & Partial sums:- • k^th Val → BS:- • Color Balls • Maximize fraction • Number & Sum of digits • Median of Subarray Sum

Max Rect Hist → O ?

Saathi

Date ___ / ___ / _____

Q:-
- Egg-Drop ✓
- Gasstation ✓
- Max Product Subarray ✓
- Max Rect. with 1's ✓
- Regular expression II →?
- Connect Nodes on same level ✓
- Encoding & Decoding (Ways to decode) ✓ O ✓
- Min Diff. Subsets
- Collect Resources

O? → • Max Profit by 2 stock transactions (Best time to buy stocks) ✓

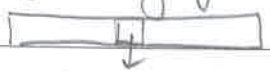
→ Unique BST II .

→ Min Jumps array → Greedy

→ Max Sum path in Binary tree

Q:- Max Profit by buying & selling shares atmost twice

→ Idea :-



for any index i, we need
→ max profit by selling on or before i (and buying)
→ " " " " " " " after i

maintain 2 DPs → DP left :- Stores max profit by buying & selling after i

DP right :- Stores max profit by buying & selling before i

DP_left(i) = max(DP_left(i+1), 0, max_val − DP_left(i));
(max_val is maximum value in i+1 to n)

DP_right(i) = max(DP_right(i−1), 0, DP_right(i) − min_val);
(min_val is minimum value in 0 to i−1)

ans = max(DP_left(i) + DP_right(i)) ∀ i ∈ (0, n−1)

Saathi

Date ___ / ___ / _____

Q. Ways to decode (Decode ways) :-

A → 1, B → 2 .... Z → 26

Given string of int :- Find # ways to decode

→ If first char ≥ 1 & ≤ 9 → DP[i] += DP(i+1)

" " 2 chars ≥ 10 & ≤ 26 → DP[i] += DP(i+2)

Optimization :- use (% 3)

Q. All permutations

→

```
rec ( l , r) {
  if ( l == r)
     print + return;
  for ( i = l to <= r )
     swap (a(i) & a(l))
     rec( l+1, r)
     swap (a(i) & a(l))
}
```

Graph Coloring? , Simulate 5 sided dic ↔ 7 sided die

Q. diameter of binary tree

→ for every node :- dia = max(dia, longest left route + " right ")

Q. Gas-station Problem (Greedy)

⟹ Idea :- Iff Σ(gas[i] - cost[i]) ≥ 0 → ans possible

Pseudo code :

```
total=0, start = 0, cur = 0   → stores curr fuel if we start from start
for (i=0 to <n) {
    cur += (gas[i] - cost[i])
    tot += (gas[i] - cost[i])
    if (cur < 0) {
        start = i+1; cur = 0;
    }
}
if (tot < 0) return -1
else return start;
```

Page No.

Date ___ / ___ / _____

// if we start from i → we have (gas[i] - cost[i]) as we move to $i+1$, but at any point $j \geq i$, if $\sum_{n=i}^{j}$ gas[n] - cost[n] < 0, we cannot move to $j+1$, that is when we shift our start point. //

Egg Drop Problem

$DP(n \text{ floors}, k \text{ eggs}) = \min\left(1 + \max\left[DP(i-1, k-1), DP(n-i, k)\right]\right) \forall i \in 1 \text{ to } n$

↓ gives

min # of throws

egg breaks → go check below

egg doesn't break → check above

if $k == 1$ ⟶ return n (Base case)

if $n == 1$ or 0 ⟶ return n

Date ___ / ___ / _____

Structures

```
struct part{                    | int main(){
    int x;                      |     part p1;
    int y;                      |     p1.x = 1; p1.y = 2; p1.z = 3.00;
    float z;                    |     part p2 = {1, 2, 6.00};
};  IMP                        |     p1 = p2;
```

(continuous memory)

Struct inside Struct

```
Struct Room{                    | main(){
    dist len;                   |     Room R1;
    dis width;                  |     R1.len.feet = 12; R1.len.inch = 6;
};  IMP                        |     R1 = {{12,6},{24,2}};
```

Struct → Same as objects (Contain both data & fun)

default public — default private

Enumerations

→ `enum days{ Sun, Mon, Tues, ..., Sat };` // default start = 0

```
main(){
days d1, d2;
d1 = Sun; d2 = Tues;
cout << d1 << "," << d2 << endl; // prints "0,2"
```

→ `enum days{ Sun=1, Mon ... Sat };` // start with 1

Page No.

Saathi

Date ___ / ___ / _____

Class :-

```
class foo{
private:
    // member fun^s & data members
public:
    //
protected:
    //
};

main(){
 foo f1;
 f1.set_data(1);
}
```

→ Member fun^s → Messages

Constructor :- → no return type, has arguments

```
public:
    foo(_ _ ): x(_), y(_), _ {
        // cout msg if necessary or other initializations
    }
```

Destructor:- → no return type, no arguments

```
public
    ~foo(){};
```

Class fu^n outside class :-

```
class foo{
 public:
  void fun();
};

void foo::fun(){
}
```

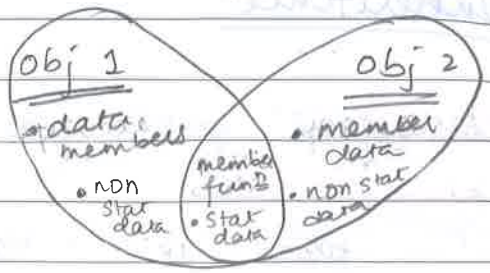
→ Objects can be passed as arguments & also returned

Date ___ / ___ / _____

Saathi

Static Data:-

```
foo {
private:
  static int s;
};
int foo::s=0;
```



Const:-

```
void fun() const {
   // can't change any value of data members
};
main(){
cosnst foo f1(...); // can only call const member fun^ns
```

⟹ `fun( const foo &x )` // can't change value of x's data members

Note:- Private data members of another object can be accessed in the first objects functions.

Copy Constructor

```
foo f1(1,2);
foo f2(f1);   // Default copy constructors → Copy member by member
foo f3 = f2;
```

Objects created last are destroyed first (stack)

Date ___ / ___ / _____

Saathi

Inheritence

```
class foo2 : public foo1 {
  public:
      foo2() : foo1(), temp(0) { } // constructor
};
```

Overriding fun^n :-

```
foo2 {
  int fun1() {
      foo1::fun1();
      //others → foo1::fun1() can also be returned
};  }
```

→ If derived class does not access data members directly → can be declared as private in parent class

→ If inheritance • private → derived class objects cant access any features of parent class,
• public → can access only public fun^{n}s & data

Abstract Class :- One which has no objects and is used only to derive other classes

Multilevel Inheritence :-

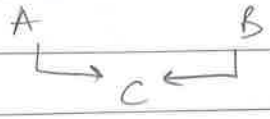
→ If fun^n not defined in C → goes to B → goes to A

→
```
C {
  f1 {
  // A::f1   invalid as C is child of B
     B::f1   // valid if A to B is public inheritance
}
```

cannot access grandparent fun^n directly

Date ___ / ___ / _____

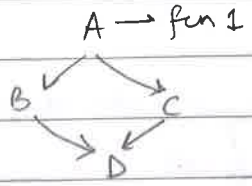
Multiple Inheritence



→ ambiguity → error (if A & B both have same fun)

→ C c1;

c1.A::f1(); // non ambiguous → no error



D. D1;

D1. fun1(); // ambiguous

D1 - B::fun1(); // non-ambiguous

Aggregation: Classes inside Classes

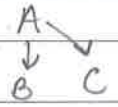
```
class A{ };
Class B{    private:
              A objA;
            public:
              f1(){
                objA.f1A();
              }
};
```

Date ___ / ___ / _____

Saathi

Virtual Functions

```
main(){
  A *ptr;
  ptr = new(B());
      Or
  B b;
  ptr = &b;
  ptr->fun;      // calls base fun if not virtual else calls B fun
```

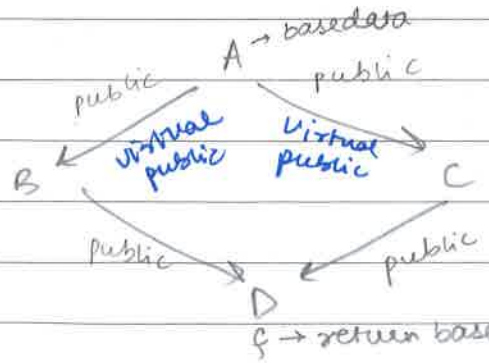



```
  returntype
⟹ virtual fun() = 0 ;   // pure virtual ~~fun~~ class
  //....
}
```

→ Note:- The virtual fun in base class has same return type + arguments as in child class

→ obj.a // when a is a data member
obj.a() // " " " " member fun

- virtual ~Base(){ }; // Destructs the child being pointed at first then base
  ~Base(){ }; // " " base
- delete ptr; // invokes destructor to free memory



- → gives no error
- → ambiguous

Saathi

Date ___ / ___ / _____

Friend Functions → has access to all private data of both classes

```
class B;
class A {
  private: data
  public:
  friend int fun (A, B);
};
class B {};
int fun (Aa, Bb){
   return a.data + b.data;
}
```

→ Friend fun^{n} has 1 more argument as compared to its normal equivalent

overloading + → imp use case

```
friend A operator + (A a1, A a2){   // constructor req to convert doubles in to a2.F & a2.S
   A temp(a1.F+a2.F, a1.S+a2.S);
   return temp;
}
```

↳ can handle $x = a + 20.0$ & $x = 20.0 + a$

↳ cannot be handled by normal operator overloading.

Friend Class :-

```
class A {
  friend class B;
};
class B {
// functions can access private data of A on objects defined locally or sent as arguments
};
```

Saathi

Date ___ / ___ / _____

Static functions :-

```
class A & gamma{
  public:
    static void fun(...){
      :
    }
};
main(){
  gamma::fun(...);
}
```

Copy constructor & assignment operator

```
A a(b); // copy constructor call  or  A a=b;
a=b;    // assignment operator call

operator A = (A & b){   // operator
    data= b.data;
}
  A (A& b){             // constructor
    data = b.data;
}
```

Note :-

- Make both private if you do not want copy constructor
- Define both → assignment operator & copy constructor
- Cannot return local objects as reference as they are destroyed
- " define A (A a) for copy constructor as to create a copy, another object is created, this causes memory confusion.

Date ___ / ___ / _____

Template

```
template <class T, class B>          → template function
int fun (T a, T b, B b2, int c)
    //return ...
}
main() {
  int x = fun(y, z, l, t)
          ↳ same type                ↳ int
          → same or diff from T
}
```

— × —

```
template< class A, class B >          → Template class
class foo{
  private:-
    A a;
    B b;
    .
    .
};
```

```
template <class A, class B>
A foo<A,B>::fun1(A a, B b){
    // ---
}
```

```
main(){
  foo<int, float> x, y;
  y=x.fun1(10, 0.1);
```

Date ___ / ___ / _____

Saathi

Template LL

```
template < class T>
struct Node{
   T data;
   Node * nxt;
};

template <class T>
class LL {
private:
   Node <T> *head;
public:
      LL(): head(nullptr) { };
      void insert_head (T x) {
         Node<T> temp;
          temp.data = x;
          temp.nxt = head;
           head =&temp;
      }
};

main() {
  LL <int> x;
}
```

Date ___ / ___ / _____

Saathi

Exception

```
class A {
private:
public:
    class exception1 { };
    :
    void fun() {
        // if error
        throw exception1();
    }

main() {
A a;
try {
    :
}
catch( A :: exception1 ) {
    // do necessary actions
}
catch (A :: exception2) {
    :
}
```

exception with arguments

```
class exception1 {
public: data variables F, S
    construction (x, y)...
};

throw exception1 ( x , y );

catch (A :: exception1 a)
int x = a.F;
string y = a.S;
- - - - - -
}
```

bad_alloc :- Built-in exception class which is called when memory cannot be allocated

```
try{}
catch(bad_alloc){
}
```

Date ___ / ___ / _____

Operator Overloading

Prefix

data type for post fix

```
class_name operator ++( ){
    return (class_name (++data));
}
```

Arithmetic operator

```
// function definition inside class
Distance operator + ( Distance d2) {
    Distance temp;
    temp.F = F + d2.F;
    temp.S = S + d2.S;
    return temp;
}

d3 = d1 + d2; // calling
```

outside class

use of friend function

```
friend distance operator + ( d1 , d2 ){
    Distance temp;
    .
    .
    return temp;
}
```

→ similarly for + used to concatenate strings

```
Distance . Distance :: operator + (Distance d2)
```

→ < & += operators are defined similarly

Date ___ / ___ / _____

DAA.

find position of curr element — select smallest

1. Insertion Sort, Selection Sort, Bubble Sort
2. Merge sort
3. Master Theorem
4. Priority Queue
5. heaps → max-heap, insert, delete ($\log N$), heapify, build max heap ($N\log N$) → call heapify from leaves to root
   - heapify ↳ make sub array a max heap assuming all nodes below current node have a max-heap sub array
6. heapsort → $N \log N$

can be seen as BST

7. Quick Sort :- [ ] pivot [ ] → $N \log N$
non comparison
8. Count sort: $O(N+k)$ → range
9. Radix sort (Stable sorting) LSB to MSB → $O(nd)$ → # digits in largest number
   ↳ stable count sort for every place from LSB to MSB
   ↳ $O(n+10) = O(N)$ per place ⇒ Total = $O(Nd)$

lec 6 / Code

10. Median → Random Partition Algo — Best → $O(N)$, Worst → $O(N^2)$
11. Order Statistics → Median of Medians

lect 6/lect 7

12. hashing → $k \% 10$, $k \% n$, folding, mid val sq technique
   ↳ collision → chaining (open hashing), open addressing (closed hashing)
   - ↳ linear probing
   - ↳ quadratic probing → $(h + i^2) \% n$ — checked probes
   - ↳ double hashing

handling collision → 1) Chaining → worst $O(N)$, extra space

2) Linear Probing → $(h(k) + i) \% n$ → mod factor
   - i → # of times of hitting a filled cell
   - $h(k)$ ↳ hash of key
   - +ve → saves spaces
   - −ve → search time → worst $O(N)$
   - → Deletion tough → handle using marking deleted cells
   - → clustering

secondary clustering

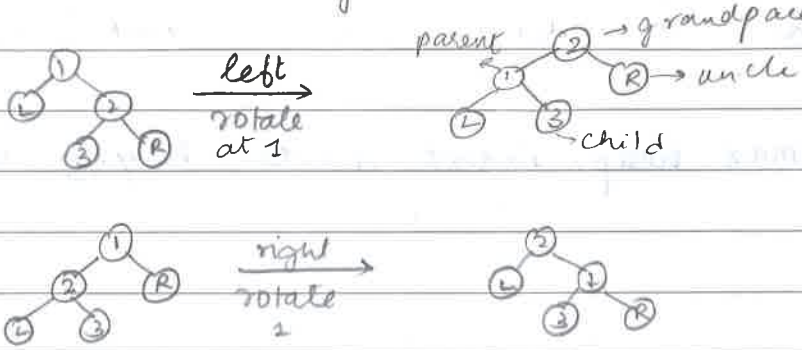
3) Quadratic Probing → $(h(k) + i^2) \% n$
   - +ve → saves space, no primary clustering
   - −ve → worst case → $O(n)$, no guarantee of finding

AVL

Date ___ / ___ / _____

13. **Red-Black Tree** (BBST) → root, nil → Black
→ red children → Black
→ node to nil → # of black nodes equal

rotations :- rearrange subtree → decrease height

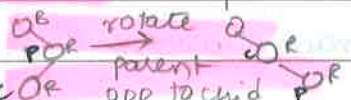


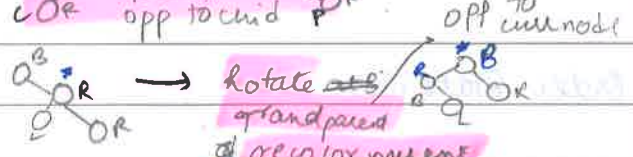
Insertion :- • insert 'x' in red

→ if x = root → x (black)

→ if x(uncle) = red → recolor parent, uncle & grandparent
↳ i.e parent, uncle red
grand parent → black

→ if x(uncle) = black & parent is red

(i) Triange 

(ii) line 

→ while inserting do changes at current node & check for violations of parent until root is reached.

Deletion :-

Idea :- • Replace the node to be removed with its right childs left most child (in order successor)

• ~~Remove the~~ Repeat until the node to be removed is not a leaf node.

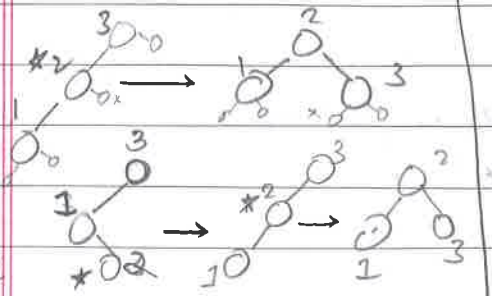
• If leaf node is red, remove it, else handle the case of double black node

Self Balancing

Date ___ / ___ / ____

Saathi

AVL Trees → BBST



→ • uses same rotation as RB Tree
• finds Balance factor, of nodes in path of inserted node till root
→ if |Balance factor| > 1, then rotate
Balance factor = height of left subtree − " " right "

Double hashing :- $(h_1(k) + i\,h_2(k)) \% p$

+ve → no primary/secondary clustering, no extra space
-ve → worst case → O(N)

Uniform hashing, perfect hashing, dotproduct, multiplication

Date ___ / ___ / _____

Saathi

- logistic Regression → linear regression
- seasonality, trend, noise
- $\frac{1}{1+e^{-x}}$
- softmax vs sigmoid → $\frac{e^{x_i}}{\sum e^{x_i}}$
- Stationarity, Differencing