Get the why of every steps → concept clear + not remembering

Page No.
Date 24-06-25

Linked List

(complete implementation in 3 pages) → just fns.

* Dynamically allocated, different from arraylist (doubles when full after copying earlier)
* Dont have indices, they are pointed via ref. variables

* Basic Declaration:- (DONE)

```
public class LL {
    class Node {
        Node next;
        int val;

        public Node (int val, Node next) {
            this.val = val;
            this.next = next;
        }
        [const^r of val only]
    }
    public LL () {
        this.size = 0
    }
}
```

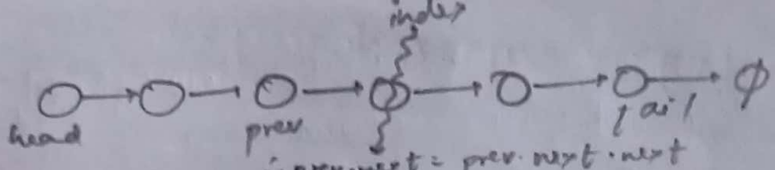
- const^r of both → reason:- insert index fn.
- this.val → helps it telling about which obj. calls for it.
- const^r of val only → reason → used in insert first fn., insert last
- public LL() → initialize size of LL (not Node) with 0. ★
- after this → declare ~~define~~ Node head, tail & int size as private

* Insert first, Insert last, Insert index
  Delete " Delete " Delete "
  Display
  → functions → easy & quick.

* Insert first → void return type → arg. value only

① public void Insertfirst (int val) {
② Node node = new Node (val); → create node of value passed

Now think you're inserting at start to pehle (starting mein) head to uske aage hi hoga na, phir uske bad assign kr denge to waha pe declare ~~krne ke baad~~ new head as:-



Ajanta
Page No.
Date

④. node.next = head;

⑤. head = node; (assign node val to head, cz ye head hai ab).

null case: ⑥ if (tail == null) return tail = head; → assign that head to tail also.

⑦. size++;

insert first:
summary:→ void return type, val as arg method → create node & pass value → next of node is head → node now became head → null ptr. exception if tail is null ∴ assign new head to tail → size++.

* Insert Last → same fn. type
① ~~count~~ If already null ptr. excep. ∴ insertFirst(val);
② else create node of value & [do same steps but node tail se linked hota hai & usse baad lagta hai]
③ ∴ tail.next = node;
④ now this is tail → ~~node = tail;~~ → this will same as insertFirst → tail = node; (give the val of node to tail)
⑤ size++;

Insert Last

* Insert index → void return type; arg both ~~node~~ & val (int index → jaha dalna hai)
① if index == 0 → insertFirst(val); { Main file me basis se socho kaise use karenge bina index bataye.
→ " " size → "last";
② Create temp node starting (initialize) from head → Node temp = head;
③ index tak pahuchao temp. ko → for(i=1 to index) temp = temp.next;
(index se ek pehle)
④ You know we made a node of val & Node (const.) → Create node & pass value & temp.next;
~~also can we as temp fit i==index tak pahuchate aur bina seedha temp pass kiye.~~
"temp jaha pahucha & usse aage rakhna hai
"when i just one less than index tab temp = temp.next ho gya jo index pe pahuch gya already
⑤ put node value on temp.next → temp.next = node;
⑥ size++;

Summary → pehle fn. banao void & val & index → if 0 or size we made fn. → temp node iteration from head to index → create node & pass val & temp.next → put node in temp.next → size++

Ajanta
Page No.
Date

* Delete first :- void type & obviously no arg. passed.

① get head value → int val = head.value ;
variablename ↳ based on Node class data member

getting the value just to show which one deleted (no need)
∴ needs return type int

Delete first
① head = head.next ***** → gets deleted
② null ptr. excep. if head == null ∴ tail = null
③ size --

* Use case (can use in insert index too) → make a get fn.
→ return type node & arg. int index (till where)

Get O(n)
① Create node initialised with head.
② for (i = 0 to index) [since node is item of LL (cant understand ?)]
learn
node = node.next ;
③ return node.

* Delete last → same fn. type as delete first.

Delete last
① if (size <= 1 → return) → remember this null ptr. exception !!!
② create second last naam ka node and initialise it as get (size-2);
③ assign that second last as tail → tail = secondLast
④ tail.next = null → parampara + pratistha + anushashan
⑤ size --;

* Delete index → arg: int index

Delete idx
① if index == 0 → deleteFirst();
" " (size-1) → " last "
② Node prev = get (index - 1); → waha pe pahucho
③ prev.next = prev.next.next → value fit
④ size --;

* Display → void, no arg

Display
① Create temp from head
② while temp != null {
sout (temp.value + " → ");
} temp = temp.next

L.L. Ques. after implementation stuffs.

Page No. Date 25/06/25

Stacks and Queues

Implementation → real quick

→ Interface

* Stacks → FILO ; Queue → FIFO
* Stack<Integer> stack = new Stack<>();

class — push → pop

* Queue<Integer> queue = new LinkedList<>(); → remember

add, remove, offer, peek, poll

{ since interfaces are not instantiated directly.

* Deque<Integer> deque = new ArrayDeque<>();

addFirst, addLast → removeFirst, removeLast (O(1)) ⚡

Custom Stack Implementation :-

fixed size

Basic declaration :-

unlike two diff. class in LL → LL, Node here we just customStack.

declaration → constr(0) → ptr initialized

```
public class CustomStack {
    int[] data;  → data of stacks are stored in array
    private static final int DEFAULT_SIZE = 10;
```

static → no need to initialize in other ~~class~~ fn. (private)

final → fixed value; has convention variable name in capital

non-parameterized constr for if size not mentioned then we default size of 10

```
public CustomStack() {
    this(DEFAULT_SIZE);
}
```

constr if size is mentioned → if will used for data length

```
public CustomStack(int size){
    this.data = new int[size];
}
int ptr = -1;
```

Writers to avoid mess

Done close the class bracket as it is public class → contains all

based on file

Try → Can just write Exception except stack exception

Ajanta
Page No.
Date

* add (push) :→ return type = boolean , arg. int item
  ① if (isFull()) { → make this fn.
    print ("Stack is full");
    return false;
  ② ptr++ ; → item aaya hai to ++ use point kare to badhao!
  ③ jo item aaya hai us data me daalo
    data[ptr] = item;
  ④ return true ; → just showing yes you can do it add ho it's done.

* isFull fn. → return type = boolean , no arg. → ∵ made just for check.
  ↳ ① tab ptr last me hoga tab. (" ptr started from -1 ")
  directly → return ptr == data.length - 1 ;

* isEmpty → ditto same bs ptr at -1 ;
  ↓
  since yeha se initialize hua tha.
  first element index/ptr is 0.

* pop → return type → int , arg → nothing , throws StackException {
  (→ just to let you know which one popped (not imp.))
  ↓ we without this (void) too!
  ① if (isEmpty()) {
    throw new StackException ("cannot pop from empty stack");
  ② int removedItem = data[ptr]
  ③* ptr--; → main thing
  ④ return removed item;

  or directly
  return data[ptr--];
  ↓
  show then decrement

* peek → check top element → return type int , arg → nothing , throws StackException {
  ① if (isEmpty()) {
    throw new StackException ("Can't peek from empty");
  ② return data[ptr];

* In psvm throws StackException { (focus here only.)

Ajanta
Page No.
Date

Stack

Dynamic Stack not imp.

revise & check reasons for all conditions → all conditionals!

Custom Queue implementation :-

*Basic Declaration same as of Custom Stack but here ptr name is end (since FIFO) & it's initialized with 0.

i.e. int end = 0;

add :- return type → boolean, arg. → int item

① if (isFull()) { → make fn. → Full when end == data.length
return false; } → same steps.

② data[end] = item;

③ end++;
return true

Similarly Empty when end == 0.

Array started from 0.

remove → return type → int, arg → nothing throws Exception {

① if Empty throw new Exception("Queue is empty");

② directly return data[end--] ✗ → wrong. Diff. from stack only

② removed will be first → int removed = data[0];

③ ek ek karke sb data values ki location piche lao :-

for(int i=1; i<end; i++){ → not started with i=0 ;; data[0-1] = data[-1] ho jaga

data[i-1] = data[i];

④ end--; ☆ → ek kam hua hai na!

⑤ return removed;

front → return data[0]; empty → exception

Display :- return type void; arg nothing

for(i=0 to <end){ ☆☆☆ → since jo last me aaya wo sabse piche hoga & last me nikalega

print(data[i] + "←")

print("End");

0←0←0←0←0

first last

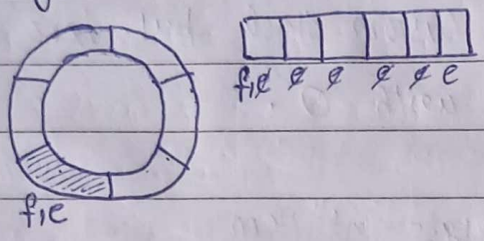
same for stack just arrow diff.

5, 5, 1, 2, 3, 0, 0, 1

Page No.
Date 26/06/25

Circular Queue implementation:-

- first and end ~~ptr~~ will be declared and all the items in circular queue will be b/w first and end (they'll be considered only).



- concept → • at first, "first" and "end" will be at 0th index.
  - item inserted → end + 1
  - item removed → front + 1 (∵ items b/w front + end only considered)
  - Now if size of the array is reached for the end the to repeat its path we use remainder concept → easy → [end = end % size] → if end is at data.length
    - ↓ v. popular concept + easy
    - ↳ explanation

for array size of 5 if end reached till 6 i.e. (>5) then modulo it by size to revert from 1 → 6%5=1 ; 7%5=2 ....

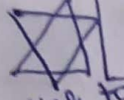
- Since it is a queue, ∴ we'll use a end_ptr also to track.

except end_ptr;

anything is increasing ∴ add safety

- Basic declaration → same as of queue + initialize front + end as 0 with end_ptr = 0 (this is already in queue)
- isFull & isEmpty → " " " " for end_ptr
- insert → return type = boolean, arg = item
  1. if(isFull) return false
  2. end aage badhega on insertion (read concept) to usme item daalo.
     data[end] = item;
     end++;
  3. end = end % data.length; → !: For safety purpose if end > data.length then only used otherwise no use i.e. 3%5 = 3 only
  4. parampara → end_ptr++;
  5. return true;

insert
- full → false
- data jaega & end badhega
- data safety
- ptr badhega



*I've mistakenly used front in place of first kindly understand accordingly.

Aaya just stick to concept.

Page No.
Date

just showing purpose

* remove → int = return type, args = nothing, throws Exception
  ① if empty throw exception
  ② get value for returning purposes → int removed = data[front];
  front++; → concept
  ∴ item aage se nikalega (FIFO)
  ③* safety for front too → front %= data.length;
  (Circle, wo bhi aage badh rha na)
  ④ end--ptr--;
  ⑤ return removed;

* offer → return type int, arg → nothing throws exception.
  ① if empty throw exception.
  ② ~~dat~~ return data[front]; → not data[0]; ★
  ↓
  done mistake while coding

* Display → void, no args
  ① if empty → print empty
  ② DO while loop →

```
int i = front;
do {
    print(data[i] + " -> ");
    i++; → increasing ∴ add safety
    i = i % data.length;
} while(i != end);
print("END");
```

★★★★★
Cannot write as
while(i <= end)
↓
because front can be greater than or less than or equal to end.

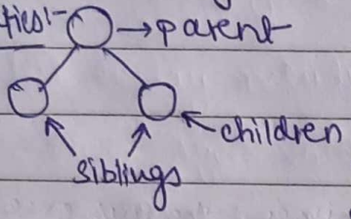
Ajanta
Page No.
Date 26-06-25

Binary Trees and BST

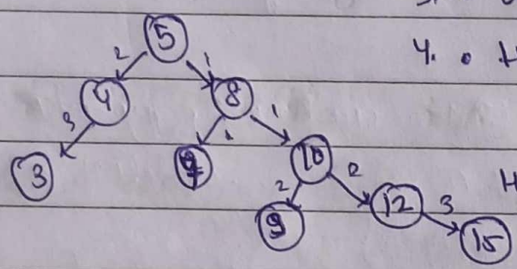
Implementation & Top use of recursion

* Optimind approach → O(logN) ↳ except skewed = LL
* BST → small = left ; big = right
  ↳ skewed for sorted data ∴ AVL is used.

Properties:-



1. size of tree = total no. of nodes
2. edge = lines connecting nodes
3. leaf = last nodes
4. Height = max. no. of ~~node~~ edges b/w that node to leaf

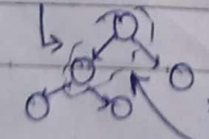


Height of 8 = 3 (max for leaf 3, 15)

5. • Starting node = ancestor
   • leaf (last) node = descendent
   } → b/w a node to leaf.
6. • Degree = no. of children a node has (leaf node degree is 0).

Types:- 1. Complete : all levels filled apart from last level + last level filling from left to right
2. Full/Strict : either 0 or 2 children of any node.
3. Perfect : all leaf on same level & all levels are filled.
4. Balanced : when avg. ht. is O(logN) → opp. of skewed → eg. ~~BST~~, AVL
5. Skewed : every node has 1 child.
6. Ordered : when every node has some properties. e.g BST.

Observations:-
- (take eg. & check yourself) Total nodes of perfect BT of height H → $2^{(H+1)} - 1$.
- For perfect B.T. → total no. of leaves = 2^H.
- " Strict " → " " " " = internal nodes + 1



⇒ leaf = 3
Proof ⇒ internal = 3 - 1 = 2

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();
int t = sc.nextBoolean();
int f = sc.nextFloat();

Ajanta
Page No.
Date

* Implementation:- (Binary Tree) → import java.util.Scanner;
→ Insert → Display
import scanner coz you gonna use it

* Basic Declaration:- public class BinaryTree {

class Node {
int val; → same
Node left;
Node right;
} Constr ∃ val.

Diff. from LL this node has left & right rather next.

defined node of tree needs declaration (not general, a constant ~~is used~~ ∴ declared) ← private Node root;

→ How beautifully recursion tree is made; just understand thoroughly

* Insertion:-

a. Firstly it will take root only & itself will call for further insertion via method overloading / may use different method name too!

Takes first value from user

return type void; arg → Scanner sc

① public void insert(Scanner sc){
② print("enter root node"); // Integer type
③ int val = sc.nextInt();
④ root = new Node(val); ← make a new node & put that val in it as root.
⑤ insert(sc, root); ← [first value as root reserved]

b. Further insertions based on left or right of root.

① public void insert(Scanner sc, Node node){ ← root will be passed initially then next nodes
② print("enter left of:", node.val);
③ boolean left = sc.nextBoolean();
④ if(left){ → (if user write true after this only)
⑤ print("enter left of:", node.val);
⑥ int val = sc.nextInt(); * thought process checked ✓
⑦ node.left = new Node(val);
⑧ insert(sc, node.left);
↓ this will now carry forward
∴ right;

Ajanta
Page No.
Date

Display

a. First root then carry forward
return type = void, no arg.

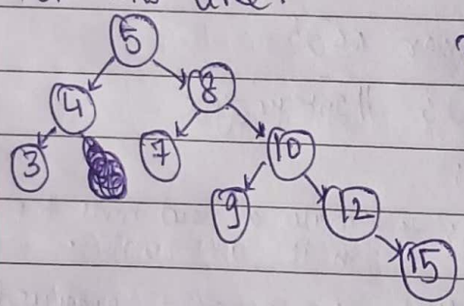
* [this.root why?] display(this.root, "->");
↓
first one passed will be root in display(node, indent)

b. All display including root as start (overloading)
return type = void, arg = node, String indent
① if (node == null) return;

Displaying based on insertions.
② print(node.value, indent);
③ left → display(node.left, indent);
④ right → " (" right ");

think recursively as tree

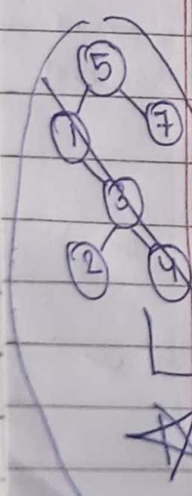
For this tree:-



Displayed as: 5 4 3 8 7 10 9 12 15
↓
(N-L-R)
(Pre-order)

For these 3 steps :- (Traversal methods implement)
1. Pre-order (N-L-R) : same as in display ②→③→④
2. In-order (L-N-R) : ③→②→④
3. Post-order (L-R-N) : ③→④→②

Learn



for ~~this~~ example Inorder: 1-2-3-4-5-7
" " " Postorder: 1-2-3-4-7-5
" " " Pre-order: 5-1-3-2-4-7

Page No. 8
Date 27-06-25

* BST Implementation → for checking only; no need in implementation

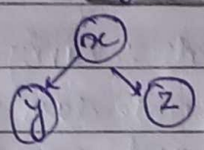
Only height and balance fn. are new here.

- Basic Declaration → only diff. is a new data member of int height inside Node class rest same.

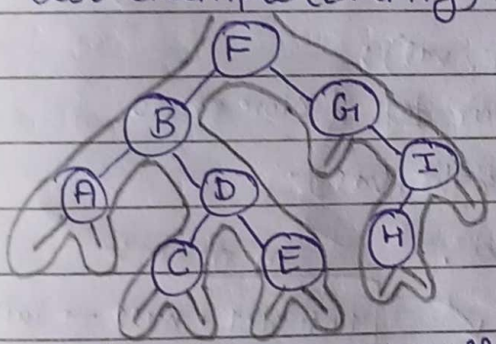
Concept :- * in order traversal of BST always gives sorted seq.
* Search in BST comprises of O(H) = O(log n) (H = height)

Traversal method - Pre, In, Post order - correction and clarity

Pre-order - ~~left~~ Root left right : x y z
Inorder - Left root right : y x z
Postorder - Left right root : y z x



Best example (clarity):



Preorder - F B A D C E G I H
Inorder - A B C D E F G H I
Postorder - A C E D B H I G F

Based on Priority

method-2 : To write all 3 directly → Kisi bhi node pe pehli baar pahuche = Preorder
dusri baar = Inorder
Teesri baar = Postorder

Preorder - F B A D C E G I H
Inorder - A B C D E F G H I
Postorder - A C E D B H I G F

9-06-25

* Insert in BST
* Search in BST
* Delete in BST
* Print in range → direct use of concept
* Print Path
* Inorder display

Ajanta
Page No.
Date 29/06/25

Basic Declaration :-

```
public class BST {
    class Node {
        int data;
        Node left;
        Node right;
        Constr of data only
```

that's it!

Insert :- return type = Node , arg = Node root, int val

(since starting with root)

reason? = null check karna aur update krna

(inserting val.)

each insertion starts from root

① if (root == null) { → initialised root with null to place first val as root
    root = new Node(val); → first val gets inserted in root
    } return root; → return type Node ∴ carry on after root

② Concept of BST :- if (val < root.data) { → ∴ left (data member of Node class (not always val, understand conceptually)

③ } else
    root.left = insert(root.left, val);
    left me daalo aur left me store kr do
    root.right = insert(root.right, val);

④ return root; → return type purpose.

in main :-

```
Node root = null → initialised
val[] = {5,1,3,4,2,7}
root = insert(root, val[i])
```

same store hoga since :- DEBUG :-

1. val[0] = 5 → root == null → true ∴ root = 5 ✓
2. val[1] = 1 → " " " → false ∴ jo val root me update hua wo ~~both~~ to root me dobaara aake store ho gya.

read debugging line by line tabhi samaj ayega

→ 1 < 5 ∴ left (true)

→ then root.left (null initially) ∴ jo root.left inside insert call hoga wo null hoga (insert(root.left, val)) — talking about this

null initially

∴ goes to first line if (root == null) → true ∴ root = 1 → jo root.left hai

and isse root.left me store karado

root.left = insert(root.left, val) → then return root

in the end ki wapas root se checking start ho

Ily for right.

Page No.
Date

* Inorder → return type = void, arg = Node root ← ∵ display starting from root.
(LNR)

① if root == null ∴ return
② inorder (root.left); → left
print (root.data + " "); → root
inorder (root.right); right.

** Search → return type = boolean, arg = Node root, int key
O(H) ∵ Yes or No (exist or not) ↓ from root start ↓ to be searched

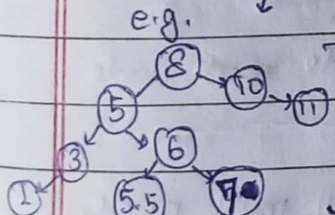
① if (root == null) → return false;
② if (root.data == key) → return true;
③ else if (key < root.data) { ∴ left search
return search (root.left, key);
based on this line further it will say true or false ★★★★★
④ ∴ right for else
⑤ return false; → if not found

★ Adobe
** Delete → return type = Node, arg = Node root, int val

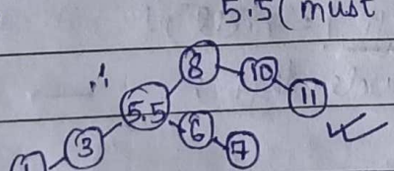
Concept: 1. No child / leaf Node → return null to it's parent
2. One child → replace with child node
3. Two children → " " inorder successor
and further inorder " can be deleted by 1 or 2.

think by visualising
BST Tree

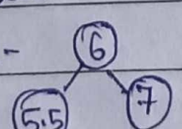
e.g.



→ To delete 5 → inorder successor = 5.5 ∴ replace with 5.5 then delete 5.5 (must lie in case 1 or 2 only)



★ Note:- Inorder successor is always the left most node of right sub-tree for that node

Here, right subtree of 5:-  leftmost in this → 5.5 ✓

★ Hence, get the inorder successor always from right.

① if (val > root.dat){ → ∴ delete from right subtree

root.right = delete (root.right, val);

isme wapas store kara rahe hi as again root for right subtree proceed karo.

}

② else if (val < root.data){

} root.left = delete (root.left, val); } similarly left

Then,

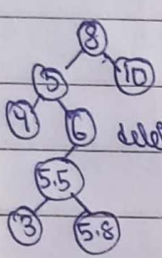
③ else {

// case 1 (NO child) → if (root.left == null && root.left == null){

return null; → return that node as null

// case 2 (1 child)

in left only → else if (root.right == null){



return root.left; ~~→ delete~~

in right only → ✓

// case 3 (2 child)

get inorder successor from right subtree (reason page before)

Node IS = findIS (root.right);

root.data = IS.data → (replaced) ✓

then delete this IS from its older position

root.right = delete (root.right, IS.data); ★

}

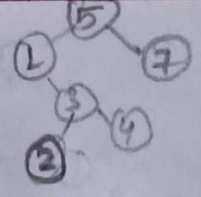
④ return root;

Function → findIS : return type Node, arg Node ~~node~~ node

① while (node.left != null){

node = node.left; → left most node.

}return node;



← 5, 1, 3, 4, 2, 7

insertion (each) starts from root

Ajanta

Page No.

Date

Q.1 Print in range → get inorder ~~delete~~ & print from particular to particular

Q.2 Print Paths in BST from Root to leaf.

Root 2 leaf → return type = void, arg = Node root, ArrayList<Integer> path.

for storing all paths one by one.

① if (root == null) → return;

② path.add(root.data);

③ leaf → if (root.left == null && root.right == null){
then print ~~all~~ path if reached till leaf → printPath(path);

④ check for right & left :-

printRoot2leaf (root.right, path); } → proceed further
" " " (" left ") ;

⑤ start removing from end to check for further paths from second last, if not then third last; proceed.

path.remove (path.size() - 1);

Func. printPath → return type void, arg = ArrayList<Integer> path.

```
for (int i=0; i< path.size(); i++){
    print (path.get(i) + "→");
}
println("Null");
```

Main function :-

```
psvm {
BST bst = new BST();
int[] val = {5, 1, 3, 4, 2, 7};
```

initialise null to ← Node root = null;

start inserting

```
for ( i=0 to val.length ){
    root = bst.insert (val[i]);
}
```

root,

```
bst.inorder(root);
println(bst.search(root, 2));
bst.delete(root, 4);
bst.inorder(root); → check purpose
bst.printRoot2leaf(root, new ArrayList<>());
}
```

Page No.
Date 29-06-25

GREEDY

Concepts via Questions

- simple and easy with good time complexity but alot of times will not get global optimum.

Concept-1 : Activity Selection - (selecting disjoints) or (max length chain of pairs)

→ * There are n activities with start & end times.
* select max. no. of activities performed by one person.
* assume work on single activity at a time.
↳ ∴ Disjoint

→ e.g. start = [10, 12, 20] } already sorted (must) ans. $A_0, A_2 = 2$
end = [20, 25, 30]
↓
end times for respective start times

→ unhi activities ko select kr skte jo end time se disjoint ho + maximum one

→ Approach : * Always select first activity (A_0) → Just simple obs. not global optimum like DP
* for next activity : start >= last chosen end time
* count ++

→ Easy code :

```
psvm {
① int[] start = {...};
   "  "  end  = "  ;
② ArrayList<> list = new ArrayList<>();
③ list.add(0); → pehla to lena hi hai

   int maxAct = 1;
④ int lastEnd = end[0];
   for (int i = 1 to end.length){
     if (start[i] >= lastEnd){
       maxAct ++;
       list.add(i);
       lastEnd = end[i];
     }
   }
```

print (maxAct); ✓

better → print ("A" + ~~ans~~ list.get(i) + " ");

inside for (int i=0 to ans.size())

Concept-2 : Fractional Knapsack

→ * There are weights and values of 'n' items
* Put these values in sack of weight 'W'
* In such a way that we get the max^{m}. total value.
* Here we can take fractional values also ∴ 0-1 knapsack & unbounded knapsack are subsets of fractional knapsack.

→ e.g. val = [60, 100, 120]
wt = [10, 20, 30]
W = 50

→ * "obviously" koi ek item ek baar hi jaa skta hai

60+100+80 = 240 ∴ 1. Pehla poora daal diye (10kg) → 60

2. Dusra " " " ± (20 kg) → 100
30 kg

∴ 20 kg left.

3. Teesra wale ka se 20 kg jitna daal diye (50 full) exact → $\frac{120}{30} \times 20 = 80$

→ Approach: * weight ↓ (to put more items), value ↑ (more profit) } favourable situation = $\frac{val}{wt}$ ↑↑

* ∴ select that one jiska $\left(\frac{val}{wt}\right)$ ratio max. ho (∵ wt kam chahiye tha aur val jada)

* If space of last > rem. wt. ∴ $\frac{val}{wt}$ x space left

→ * from above eg. $\frac{v}{w}$ = [6, 5, 4] 1st 2nd 3rd ∴ 1. ∴ if we choose 0th index first capacity left = 40

2. ∴ then " " 1st " ∴ capacity = 20 left

* 3. ∴ finally if last space > rem. wt.

∴ Fraction used → (left highest ratio) x (space left)
= 4 x 20
= 80 ✓

∴ capacity full

Ajanta
Page No.
Date 30-06-25

import java.util.*;

→ Code :- psvm {

int[] val = {---};

int[] wt = {----};

int W = 50;

int n = val.length;

double[] ratio = new double[n];

for (i=0 to n){

ratio[i] = (double) (val[i]/wt[i]);

}

data type used to store fractional values

general

Descending sort → Arrays.sort(items, (a,b) → Double.compare(b.ratio, a.ratio));

ascending sort → Arrays.sort(items, (a,b) → Double.compare(a.ratio, b.ratio));

↳ this compares a.ratio

why this?

→ If you sort ratio directly you'll lose track of which val & wt it belongs to

→ Integer not int because comparator lambda req. object array like Integer[]

Integer[] index = new Integer[n];

for (i=0 to n){

index[i] = i;

}

Arrays.sort(index, (a,b) → Double.compare(ratio[b], ratio[a]));

eg. val = {60, 20, 100}

wt = {10, 20, 30}

ratio = {6, 1, 3.33}

Direct sort → ratio = {6, 3.33, 1}

↓

Now you don't know 6 is from val 60 & wt 10

∴ we created index array = {0, 1, 2}

which keeps index of val, wt, ratio in order

Then we sorted index array based on ratio ∴ index = {0, 2, 1} ∴ first pick index 0 then 2 from 1.

double totalval = 0;

int remCapacity = W;

for (i=0 to n){

int idx = index[i];

if (remCapacity >= wt[idx]){

totalval += val[idx];

remCapacity -= wt[idx];

}

else{

totalval += ratio[idx] * remCapacity;

break; → sack gets full here

}

}

print(totalval); ✓

Descending sort 1: Arrays.sort(index, (a,b) → Double.compare(ratio[b], ratio[a]));
(sort after comparing with other)

Descending sort 2: Arrays.sort(coins, Comparator.reverseOrder());
(direct)

Ajanta
Page No.
Date 30-06-25

Concept-3: Min. Abs. Diff. Pairs

→ * Find the pairs from two arrays such that their absolute difference is minimum.

→ e.g. * A = [1, 2, 3]
B = [2, 1, 3]
→ |1-2| + |2-1| + |3-3| = 2 ✗
|1-1| + |2-2| + |3-3| = 0 ✓

→ Approach: * nos. that are in pair if they're close to each other then uska abs. diff. utna hi kam hoga.
* Sort the arrays & get diff. at indices 0 to n.

→ Easy code: psvm {
[] A = ✓
[] B = ✓
Arrays.sort(A);
" " (B);
int minDiff = 0;
for(i=0 to A.length) {
minDiff += Math.abs(A[i] - B[i]);
}
print(minDiff);

Concept-4: Indian Coins Change (Khulla)

→ * If you have coins of [1, 2, 5, 10, 20, 50, 100, 500, 2000].
* Find min. no. of coins used to make change of value V.

→ e.g. V = 551 → ans. 3 (∵ 500, 50, 1)
V = 590 → ans. 4 (∵ 500, 50, 20, 20)

→ easy code:

```
import java.util.*;
psvm {
Integer[] coins = {1,2,5, ..... ,2000};
Arrays.sort(coins, reverseOrder());
```

(understand approach from code)

Integer[] object array for 2 comparator

Arrays.sort(coins, reverseOrder()); → Descending sorted ∵ pehla bada coin lo for min. change

Collections.

```
ArrayList<Integer> ans = new ArrayList<>();
int count = 0;
int amt = 590;
```

Concept → (Debugging way to sochlo if not clicked)

```
for (i=0 to coins.length){
if (coins[i] <= amt){
    while(coins[i] <= amt){
        count++;
        ans.add(coins[i]);
        amt -= coins[i];
    }
print(count);
```

to print coins used →

```
for (i=0 to ans.size){
print( ans.get(i) + " ");
}
```

read!

Concept-5 : Job Sequencing

* → Given Jobs with deadline and profit.
→ Every Job can be done one at a time.
→ (min. possible deadline for any Job is 1)
→ maximize total profit.

Deadline ↓ Profit ↓

* eg. Job A = {4, 20} → matlb Job A krne main 4 hrs lagega with profit 20
Job B = {1, 10} → " " B " " 1 hr " " " 10
Job C = {1, 40}
Job D = {1, 30}

Choices we can select → 1. If we select A (first), then it will take 4 hrs so we can't do any other job (∵ baaki ka 1 hr main khatam)
∴ Only A → ₹20

2. BA → can take both ∵ A ki deadline 4 hai aur B 1 hr me done ho jayega → BA → ₹30

3. CA → ₹60 → Max. ans.

4. DA → ₹50

5. BC → Not possible → ∵ Koi ek karne jaye to dusra ka deadline over.

* Approach → • sort jobs based on profit (descending)
∴ Job C, D, A, B.

• time = 0
for (i=0 to jobs){
if (job(deadline) > time){
add job in ans
}time++

* code → import java.util.*;

public class JobSeq. {
static class Job{
∵ no need for declaring objects wala jhanjhat
int id;
int profit;
int deadline;
Constr of all (id, deadline, profit)
}

psvm{
int[][] jobInfo = {{4,20},{1,10},{1,40},{1,30}};
ArrayList<Job> jobs = new ArrayList<>();
for (i=0 to jobInfo.length){
jobs.add(new Job(i, jobInfo[i][0], jobInfo[0][i])
}
deadline profit

Descending order sort for profit → Collections.sort(jobs, (obj1, obj2) → obj2.profit - obj1.profit);

ArrayList<Integer> seq = new ArrayList<>();
int time = 0;

Socho (easy hai bahut)
Jiska profit high hai wo select ho gya + kisi aur ka deadline bacha hai to add it based on profit sorting

for (i=0 to jobs.size()){
Job current = jobs.get(i);
if (current.deadline > time){
seq.add(current.id);
time++;
}}

print(seq.size() + " = max Jobs");
for (i=0 to seq.size()){
print(seq.get(i));
}

Ajanta
Page No.
Date 20-06-25

Hashing

* HashSet :-

* contains all unique elements & removes duplicate.
* elements in the set are not ordered (jumbled)
* Complexity :- Insert/Add → O(1) ; Search/contain → O(1); Delete/remove → O(1);

Array (O(1))	BST (O(H))

* HashSet<Integer> set = new HashSet<>(); → same as arrayList

```
Set.add(1);
set.add(2);
set.add(1); → no need (won't consider duplicate)
Set.contains(1) // true
```

* Iterator : * import iterator first
  hasNext, next * does'nt guarentee order of items when iterated will be same next time.

* Use case :-

```
Iterator it = Set.iterator();   // for iteration in set HashSet
while (it.hasNext()) {
    print(it.next());
}
```

O/P
1
2

* HashMap

* Must have key, value pair where key should be unique
* HashMap<String, Integer> map = new HashMap<>();
  (String: key (countries); Integer: value (population))
* Inserting →

```
map.put("India", 70);
map.put("USA", 90);
map.put("China", 70);
print(map); → O/P
```

{China=70, US=90, India=70} → order not guarenteed.

* Updating → new value with same key name ∴ map.put("China", 200);
~~(value only)~~

Note :- in Updating → exist : updates value of key respectively
→ not exist : adds new key, value pair.

* Searching (key) → map.containsKey("China"); //true
* Get (value) → map.get("China"); //200
from key map.get("Indonesia"); //null

Two types of 'for' loop :- 1. int[] arr = {12, 15, 18};
for (int i=0; i<arr.length; i++){
print(arr[i] + " ");
}

for each
2. for (int val : arr){
print (val + " ");
}

*
1. for (Map.Entry<String, Integer> e : map.entrySet()) {
print(e.getKey());
print(e.getValue());

2. keyset → Set<String> keys = map.keySet();
for(String key : keys){
print (key + " " + map.get(key));
gives value

* Deletion → map.remove("China");
print(map);

* Implementation (in next Page)
V.V.V. imp for interviews

Page No. 4
Date 02-07-25

* Implementation

* functions → put, get, containsKey, remove, ~~size~~ search, keyset
  ↓
  all in O(1) complexities
* Implemented as array of LL.

* **Basic Declaration:-**

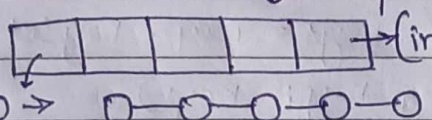
```
import java.util.*;
public class HashCode {
    static class HashMap<K,V> {
        private class Node {
            K key;
            V value;

            public Node(K key, V val){
                this.key = key;
                this.value = val;
            }
        }

        private int n;
        private int N;
        private LinkedList<Node> [] buckets;

        @SuppressWarnings("unchecked")
        public HashMap() {
            this.N = 4;
            this.buckets = new LinkedList[4];
            for(i=0 to 4){
                this.buckets[i] = new LinkedList<>();
            }
        }
```

generics (data type is dependable)
static (no need to instantiate)
check
datatype of key is K
& value is V
dependable on input
const^r of both
no. of nodes inside bucket
bucket.length = N
:(int[] arr)
(index are buckets)
since nodes are stored
index 0 →
nodes in LinkedList
n
~~size~~
due to generics, Java main type nhi dejine karo chalega but not in L.L :. used
initialised bucket size with 4
initialised bucket with LL size 4
saare indices pe new L.L liya diye



Ajanta
Page No.
Date

PUT

T.C. = O(1), worst = O(n)

→ Get the bucket index "bi" i.e. index of buckets array.
→ " " data " "di" i.e. at bi jo L.L. stored hai uska index
∴ we need bi and key to get di.
→ If di gives -1 ∴ key doesn't exist ∴ add new node at that bi and update the size of 'n'
→ Else if key exist ∴ update value at di.

```
return type = void,  arg = K key, V val
    int bi = hashFunction(key);
    int di = searchinLL(key, bi);
    if (di == -1) {   // key doesn't exist → add new node + increase n
    buckets[bi].add(new Node(key, val));
                property of LL
    n++;
    }
    else {   // key exists → get the value then place new one
    Node data = buckets[bi].get(di);
    data.value = val;
    }
```

Just to make sure ek hi index ← `double lambda = (double) n/N;`
& bada LL banta na chala jaye

```
if(lambda > 2.0){        (let K=2)
                          ↓ threshold
    rehash();
}
```

↳ written like this ∵ double (must)

*hashFunction → return type = int, arg = K key.

```
int bi = key.hashCode();
return Math.abs(bi) % N;
```

- hashCode generates unique code, here it will generate based on key (ranges too like 12765 or -314411)
- We get bi positive values only, that too in range of N.

* searchInLL → return type = int, arg = K key, int bi.

provide LL inside bi → LinkedList<Node> ll = buckets[bi];

```
for (i=0 to ll.size()) {
  * if (ll.get(i).key == key) {
      return i; → 'di' found
    }
}
return -1; → 'di' does not exist
```

* rehash function → return type = void, no args

- firstly jo buckets me tha use oldbucket me daalo
- buckets ka size double karo
- saare naye buckets me linked list banao
- aur har old bucket ke index pe jo ll hai use get karo aur put kar do ~~naye~~ waha pe old wale me.

```
① LinkedList<Node>[] oldBucket = buckets;
② buckets = new LinkedList[N*2];
③ for (i=0 to buckets.length or N*2) {
      bucket[i] = new LinkedList<>();
④ for (i=0 to oldBucket.length) {
      LinkedList<Node> ll = oldBucket[i];
      for (j=0 to ll.size()) {
          Node node = ll.get(j);
          put(node.key, node.value);
```

⑤ main fn :- psvm{ → HashMap<String, Integer> map = new HashMap<>();
generics will take the datatypes.

```
map.put("India", 190); ---- more put
ArrayList<String> keys = map.keySet();
for (i=0 to keys.size()) {
* → print(keys.get(i) + " " + map.get(keys.get(i)));
map.remove("India");
print(map.get("India")); → null.
```

GET return type = "V", arg = K key

```
int bi = hashFunction(key);
int di = searchInLL(key, bi);
if (di == -1) {
    return null;   → key doesnot exist
}
else {
Node node = buckets[bi].get(di);   ← get the node from LL if key exists
    return node.value;   → get the value.
}
```

Contains key return type = boolean, arg = K key

```
int bi = ✓
int di = ✓
if (di == -1) {
    return false;
}
else {
    return true;
}
```

Remove return type = V, arg = K key.

```
int bi = ✓
int di = ✓
if (di == -1) { return null; }   → "DNE" ∴ no remove
else {
Node node = buckets[bi].remove(di);
n--;
return node.value;   → viewing purposes.
```

IsEmpty boolean, no arg

return n == 0; → if no nodes in hashmap then Empty.

KeySet return = ArrayList<K>, no arg

```
ArrayList<K> keys = new ArrayList<>();
for (i=0 to buckets.length) {
    LL<Node> ll = buckets[i];
    for (j=0 to ll.size)
    Node node = ll.get(j);
    keys.add(node.key)
}
return keys;
```

Loops add krdo keyList ka aur return krdo.