

① Find the middle by slow fast approach
 ↳ add a corner case if (fast == null) { // even case }

↳ Prev will be one node before slow.

② Right Head = mid.next
 mid.next = null;

M.Sort(head) left half
 M.Sort(right Head) Right half] sorting

③ Merging

④ Make temp Linked List

temp → head [dummy node]
 temp.data = -1

if (Left.data < Right.data) {
 temp.add(Left.data); }

else { temp.add(RightHead.data); }

if elements are remaining add all them.

↳ in final list remove first the dummy node

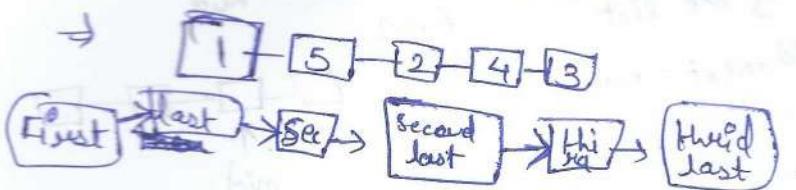
Zig Zag Linked List

→ We have given a Linked List

$$L(1) \rightarrow L(2) \rightarrow L(3) \rightarrow L(4) \rightarrow L(5) \rightarrow L(6)$$

We need to convert it in zigzag form

example - 



Approach

① find mid Node (mid = 1st half last node)

(slow → fast approach)

② Reverse second half (Because it's hard to travel backwards in LL)

③ divide the both LL

④ Alternate merging

↳ Node LH = 1st half head

Node RH = 2nd half head

Node NextL, NextR
(next left, next right)

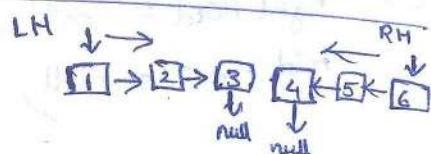
while ($LH \neq \text{null}$ & $RH \neq \text{null}$) {

 nextL = LH.next

 LH.next = RH

 nextR = RH.next

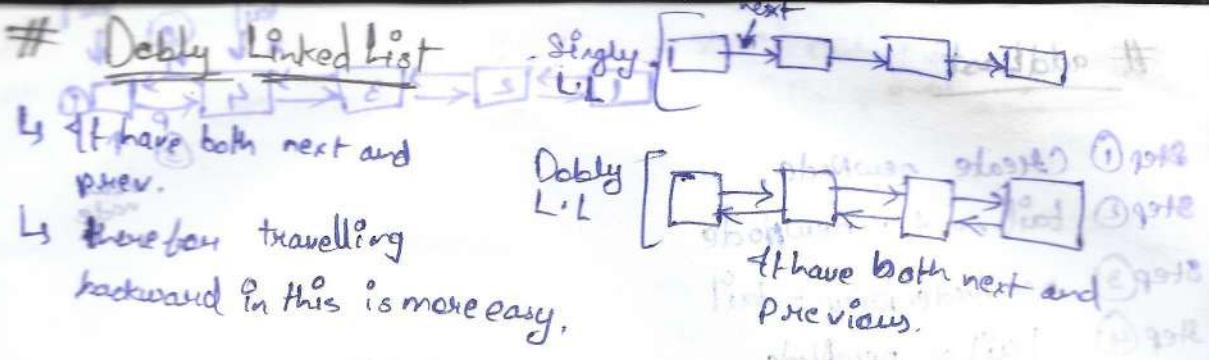
 RH.next = nextL



zigzag

update { $RH = \text{nextR}$
 $LH = \text{nextL}$ }

}



Code → Class DoubleLL {

 class Node {

 int data;
 Node next;
 Node prev;

 } constructor
 public Node (int data) {
 this.data = data;
 this.next = null;
 this.prev = null; }

 public static Node head;
 public static Node tail;
 public static int size;

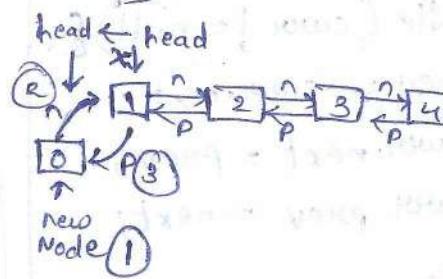
add First

Step 1) Create node;

Step 2) newnode.next = head;

Step 3) head.prev = newNode;

Step 4) head = &newNode;



Corner Cases

→ if (head == null) {
 head = tail = newNode;
 return; }

Remove First

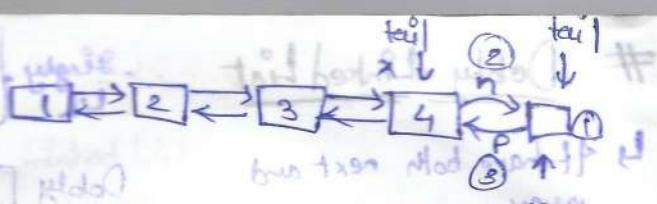
Step 1) Pass head = head.next;

Step 2) head.prev = null;

Step 3) size--;

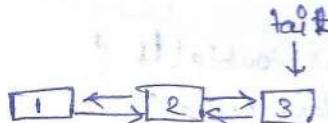
addLast

- Step ① Create newNode.
 Step ② tail.next = newNode
 Step ③ newNode.prev = tail
 Step ④ tail = newNode.



RemoveLast

- Node temp = tail;
 Step① tail = tail.prev;
 Step② tail.next = null;
 return temp.data;



Reverse a Doubly LL.

```
2 Node curr = head;
Node prev = null;
Node next;
```

```
while (curr != null) {
    next = curr.next;
    curr.next = prev;
    curr.prev = next;
    update {
        prev = curr
        curr = next
    }
}
```

here just link the head with tail and tail with head

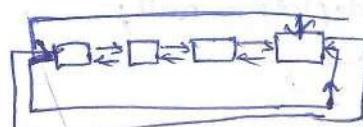
example → head.prev = tail;
 tail.next = head;

Circular Linked List

Circular Doubly LL



Circular Doubly LL



- Doubly list is asked rarely.

(*Citostomus elongatus* pd għor Vilgħi - 12/12/12)

30°C

10°C

5°C

2°C

1°C

0°C

-1°C

-2°C

-5°C

-10°C

-15°C

-20°C

-25°C

-30°C

-35°C

-40°C

-45°C

-50°C

-55°C

-60°C

-65°C

-70°C

-75°C

-80°C

-85°C

-90°C

-95°C

-100°C

-105°C

-110°C

-115°C

-120°C

-125°C

-130°C

-135°C

-140°C

-145°C

-150°C

-155°C

-160°C

-165°C

-170°C

-175°C

-180°C

-185°C

-190°C

-195°C

-200°C

-205°C

-210°C

-215°C

-220°C

-225°C

-230°C

-235°C

-240°C

-245°C

-250°C

-255°C

-260°C

-265°C

-270°C

-275°C

-280°C

-285°C

-290°C

-295°C

-300°C

-305°C

-310°C

-315°C

-320°C

-325°C

-330°C

-335°C

-340°C

-345°C

-350°C

-355°C

-360°C

-365°C

-370°C

-375°C

-380°C

-385°C

-390°C

-395°C

-400°C

-405°C

-410°C

-415°C

-420°C

-425°C

-430°C

-435°C

-440°C

-445°C

-450°C

-455°C

-460°C

-465°C

-470°C

-475°C

-480°C

-485°C

-490°C

-495°C

-500°C

-505°C

-510°C

-515°C

-520°C

-525°C

-530°C

-535°C

-540°C

-545°C

-550°C

-555°C

-560°C

-565°C

-570°C

-575°C

-580°C

-585°C

-590°C

-595°C

-600°C

-605°C

-610°C

-615°C

-620°C

-625°C

-630°C

-635°C

-640°C

-645°C

-650°C

-655°C

-660°C

-665°C

-670°C

-675°C

-680°C

-685°C

-690°C

-695°C

-700°C

-705°C

-710°C

-715°C

-720°C

-725°C

-730°C

-735°C

-740°C

-745°C

-750°C

-755°C

-760°C

-765°C

-770°C

-775°C

-780°C

-785°C

-790°C

-795°C

-800°C

-805°C

-810°C

-815°C

-820°C

-825°C

-830°C

-835°C

-840°C

-845°C

-850°C

-855°C

-860°C

-865°C

-870°C

-875°C

-880°C

-885°C

-890°C

-895°C

-900°C

-905°C

-910°C

-915°C

-920°C

-925°C

-930°C

-935°C

-940°C

-945°C

-950°C

-955°C

-960°C

-965°C

-970°C

-975°C

-980°C

-985°C

-990°C

-995°C

-1000°C

-1005°C

-1010°C

-1015°C

-1020°C

-1025°C

-1030°C

-1035°C

-1040°C

-1045°C

-1050°C

-1055°C

-1060°C

-1065°C

-1070°C

-1075°C

-1080°C

-1085°C

-1090°C

-1095°C

-1100°C

-1105°C

-1110°C

-1115°C

-1120°C

-1125°C

-1130°C

-1135°C

-1140°C

-1145°C

-1150°C

-1155°C

-1160°C

-1165°C

-1170°C

-1175°C

-1180°C

-1185°C

-1190°C

-1195°C

-1200°C

-1205°C

-1210°C

-1215°C

-1220°C

-1225°C

-1230°C

-1235°C

-1240°C

-1245°C

-1250°C

-1255°C

-1260°C

-1265°C

-1270°C

-1275°C

-1280°C

-1285°C

-1290°C

-1295°C

-1300°C

-1305°C

-1310°C

-1315°C

-1320°C

-1325°C

-1330°C

-1335°C

-1340°C

-1345°C

-1350°C

-1355°C

-1360°C

-1365°C

-1370°C

-1375°C

-1380°C

-1385°C

-1390°C

-1395°C

-1400°C

-1405°C

-1410°C

-1415°C

-1420°C

-1425°C

-1430°C

-1435°C

-1440°C

-1445°C

-1450°C

-1455°C

-1460°C

-1465°C

-1470°C

-1475°C

-1480°C

-1485°C

-1490°C

-1495°C

-1500°C

-1505°C

-1510°C

-1515°C

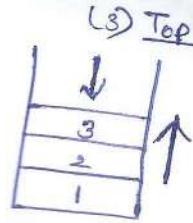
-1520°C

-1525°C

-1530°C

Stacks →
 Explicit (made by us)
 implicit (made by computer automatically)

- Operations
 - ① Push $O(1)$
 - ② Pop $O(1)$
 - ③ Peek $O(1)$
- Every operation
for only top element



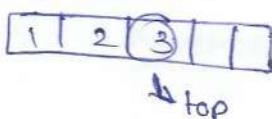
(LIFO)

Last In First Out

Implementation -

Arrays

fixed size



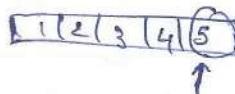
- stack == full

As array is of fixed size there we does not prefer this for Stack.

Stack → using ArrayList.

ArrayList

variable size



last element = top

LinkedList

variable size



Head = Top

here head is top because head is the only element on which we can perform the tasks with $O(1)$ time complexity.

Push → To add element

```
public static void push(int data)
{ list.add(data);
}
```

Pop → To remove

```
public static int pop()
{
    if list is empty
        if (isEmpty())
            return -1;
        int top = list.get(list.size() - 1);
        list.remove(list.get() - 1);
        return top;
}
```

Peek

```
public static int peek()
{
    if the list is empty
        if (isEmpty())
            return -1;
        return list.get(list.size() - 1);
}
```

IsEmpty

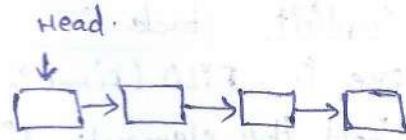
```
→ public static boolean isEmpty()
{
    return list.size() == 0;
}
```

Stack → Using Linked list

↳ Here top is considered as head.

1) IsEmpty → to check whether the L.L is empty or not.

```
→ public static boolean isEmpty () {  
    return head == null;  
}
```



2) push

↳ As the top is head always

so we would use add-

-First function.

3) pop

```
if (isEmpty ()) {  
    return -1;  
}
```

```
int top = head.data  
head = head.next;  
return top;
```

4) peek

```
if (isEmpty ()) {  
    return -1;  
}  
return head.data;
```

Stack using ICF

Just ~~Wrote~~ write

import java.util.*;

Without Functions Stack < Integer > s = new Stack <>();

- ① s.push();
- ② s.pop();
- ③ s.peek();
- ④ s.isEmpty();

~~Push~~ at the bottom of the stack amazon.

→ we will solve with recursion

→ In recursion there have implicit stack in which there is FILO (First in last out) it will hold the elements till we push the desired element at the base and then we will add the elements one by one by our stack.

Reverse a String using stack

↳ Most easiest thing to do in stack is this

- ① Just add the elements.
- ② And then print the elements
- ③ Due to LIFO, stack will be automatically reversed.

Approach

- ① push all elements
- ② Peek - pop.

Reverse a Stack → We are told to make a stack that is reversed flat of original.

→ example Input Output

3
2
1

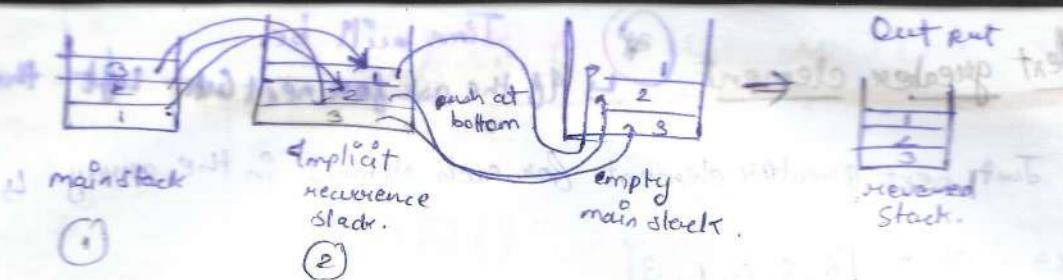
1
2
3

→ By brute force approach then will need extra memory so to save memory and time we will use recursion

Approach

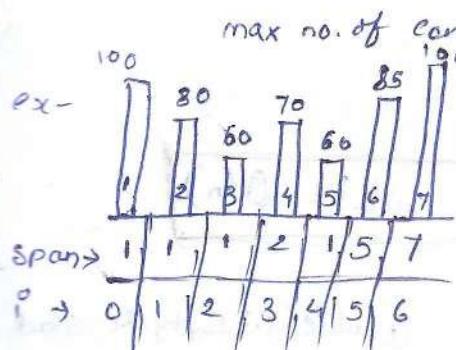
Step ① add all the elements in recursion stack
base case → until our stack become empty.

Step ② backtrack all the element in the main stack but use push at bottom so the stack will be empty.



Q# Stock span problem

$i = \text{index}$



max no. of consecutive days for which $\text{price} \leq \text{today's price}$
including the day itself

∴ For this we derive a formula
 $\text{Span} = i - \text{prevHigh}(i)$

Ex- for day 4

$$\Rightarrow 3 - 1 = 2$$

but this take so much time
 $O(n^2)$

Now [optimized solⁿ] using Stack ← our stack will store the indexes of the stock price

↳ we will make an stack which store the value of a stock

↳ If the current stock price is greater than previous stock price store the value of stock in the Stack.

↳ And in array stores $\left[\begin{array}{c} \text{Index of} \\ \text{current} \end{array} \right] - \left[\begin{array}{c} \text{Index of} \\ \text{Prev high} \end{array} \right]$

but if the curr stock price is greater than prev stock price remove the prev from stack and check again until find a stock price that is greater.

↳ If found the the greater price stock

and if not and the stack became empty then store in array $\rightarrow \left[\begin{array}{c} \text{Index of} \\ \text{curr} \end{array} + 1 \right]$

Next greater element Q Time will be O(n) if the ask for next Gttr left the array.

ex → arr = [6, 8, 0, 1, 3]

next Greater = [8, -1, 1, 3, -1]

If found no greater element then -1 or if no element after that then also -1.

Brute Force

↳ Nested Loop

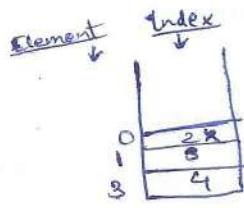
Time will be O(n)

- For Optimised with Stack ↗

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

next Greater [] = [8, -1, 1, 3, -1]

→ here we check from backwards.



→ for 3

stack is empty ∴ -1

→ ~~for~~ and also push the index of 3 in stack

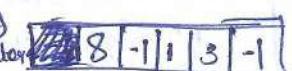
for 1

3 is greater

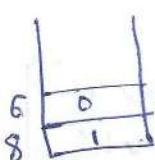
∴ 3 add in this

also push the index of 1 but add backwards

Ans →



arr ↙



→ for 0

1 is greater and next

∴ 1

push i of 0

→ for 8.

0 is smaller ∴ pop

1 is small ∴ pop

3 is small ∴ pop

New empty ∴ -1

push index of 8

For 6, 8 is next greater and further element will be greater because 8 has popped them all out who were smaller than him

∴ 8 add in nextGreater
push index of 6

Done return next greater array.

then about loop
from 0 to 1
they can ask for next smaller left or right

Valid Parentheses

Ex - String will be given

$s = "()" \checkmark$

$s = "({})" \checkmark$

$s = "([{}])" \checkmark$

$s = "([{}])" x$

- ① Opening will be first and each opening will have its own closing.

Ex -

\Rightarrow ① Opening bracket

s.push();

② Closing bracket

↳ check does its pair present at top

if not return False;

↳ if yes s.pop() and continue;

③ At last check if the string is empty or not

if empty \rightarrow return True;

if not \rightarrow return False;

Duplicate Parenthesis

↳ we have given an string

$((a+b)+(c+d)) \checkmark$ this does not contain any duplicate parenthesis.

$((((a+b)+(c+d))) \times$ this contains a duplicate parenthesis.

$((a)+(b)) \checkmark$

* logic

\rightarrow If there is no information bet' these parenthesis then it is duplicate, means they are of no use.

Approach (stack)



① if found and

opening bracket ~~operator~~ ^{good bracket} or a mark
operator (+, -, *, /)
or . (a, b, c)

Push ();

② if found closing

int count = 0;

+ try to find is closing by
popping elements.

count++; for the no. of
elements popped.

if ~~count < 1~~ means no other
data bet' parenthesis & means
duplicate. if ~~count > 1~~ continue

Code for this

→ int count = 0

while (S.Peek () == '(') {

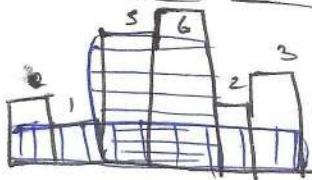
S.Pop ()

Count++ }

[count < 1 → duplicate] else (c)

S.Pop () ← c;

Max Area in Histogram



Approach

For these we will use
the concepts used in nextaller

two arrays
of nsl (next smaller left)

[-1 -1 1 5 1 2 0]

(4) then we will written the max area

nsl [next smaller right] * when no smaller left put -1

1 1 6 1 2 2 1 6 1 6 (j) * when no smaller right put n

maximum area in rectangle

it can be any rectangle

① area = height x width.

② height → this will be the height of bar

③ width →

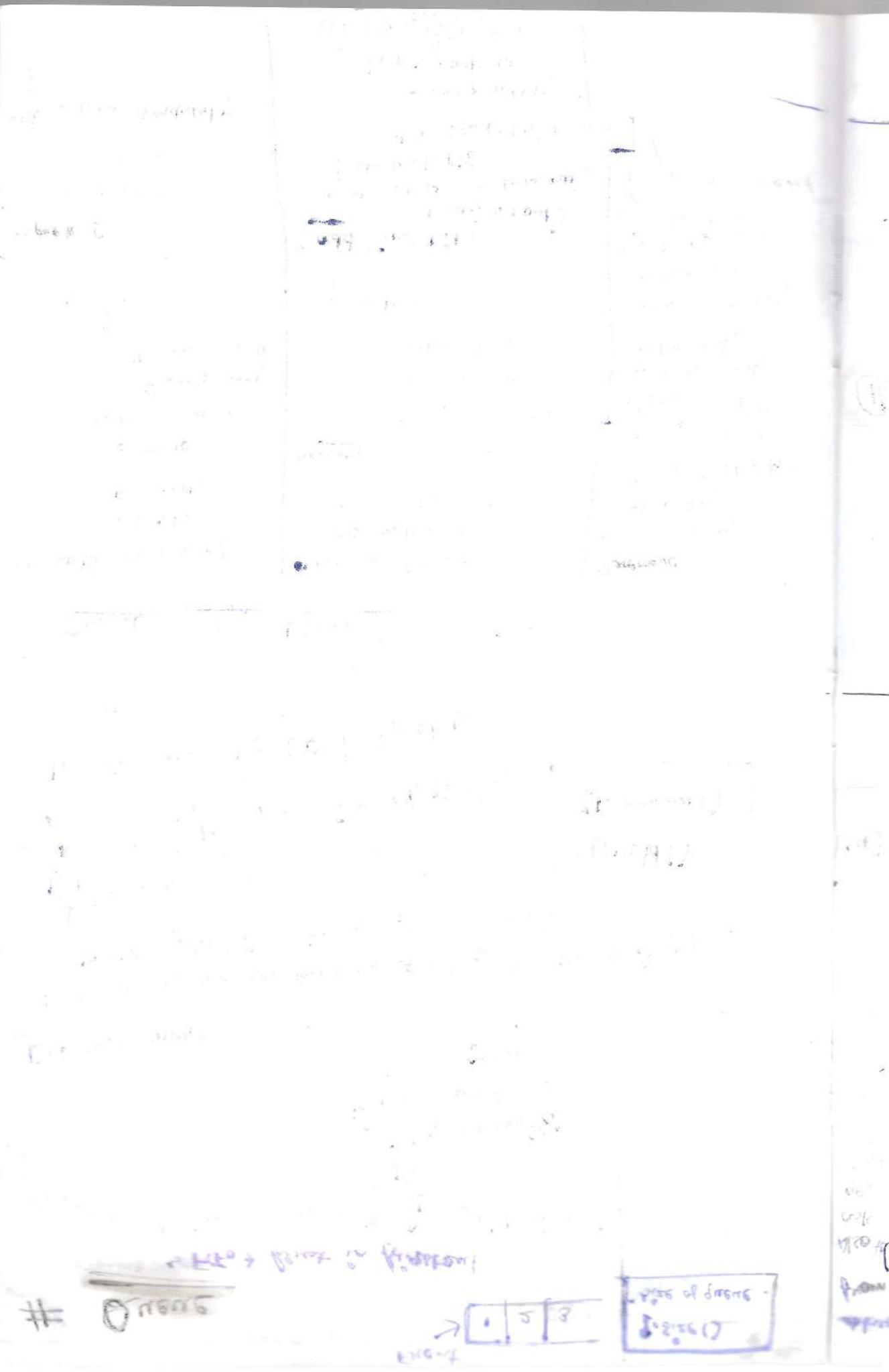
For this

Next Smaller Left

Next Smaller Right

width = j - l - 1

width of bar between two smaller bar.



Queue

Front



q.size()

- size of queue.

↳ FIFO → first in first out

↳ Can be made with array, L.L, Stack?

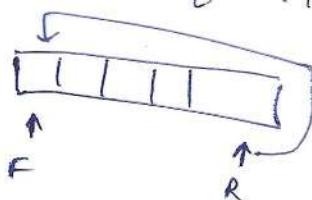
i) fixed size

ii) TC of remove($O(n)$)

iii) But we use this for
circular queue.

Circular queue

↳ Here as we get towards end of the array it get to the first index by a special formula.



$$\text{Rear} = (\text{Rear} + 1) \% \text{size}$$

↳ So here the TC is ~~$O(n)$~~ $O(1)$.

~~q1.add()~~

~~q1.remove()~~

Queue using Linked List

```
→ static class Node {
    int data;
    Node next;
}

// constructor
Node (int data) {
    this.data = data;
    this.next = null;
}
```

```
# peek {
    ① check isEmpty()
    → return -1;
}
else
    return head.data;
```

```
static class Queue {
    static Node head=null;
    static Node tail=null;
```

IsEmpty

```
public static boolean isEmpty () {
    if (head==tail==null)
        return true;
    else
        return false;
```

Add (use FIFO)

FIFO (first in last out)

Node newNode = new Node(data);

① check isEmpty

then head=tail=newNode

② tail.next = newNode;

tail = newNode

remove

① check isEmpty
→ return -1;

② int val = head.data;

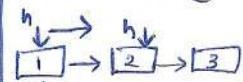
③ if single element

if (tail==head){}

head=tail=null;

return val;

④ head = head.next;
return val



This will be removed

Queue using JCV

→ import java.util.*;

public class Queue_RS {

 public static void main (String args[]) {

 functions

 Queue<Integer> q = new LinkedList<>();

- ① add
- ② remove
- ③ isEmpty
- ④ peek

→ Queue is not written here because
Queue is not a class. (Note)
→ Queue is a interface so we cannot make a
class of queue. we can use → `LinkedList<>()`
or → `ArrayDeque<>()`

Queue using Two Stacks

by keeping ↓

push O(n) or pop O(n)

```
static class Queue {
    static Stack<Integer> s1 = new Stack<>();
    static Stack<Integer> s2 = new Stack<>();
}
```

// empty

public static boolean isEmpty () {
 return s1.isEmpty();

// add

public static void add (int data) {

// transferring elements from s1 to s2
 while (!s1.isEmpty ()) {

int temp = s1.pop();
 s2.push (temp);
 }

// adding new element
 s1.push (data);

// transferring the elements back

while (!s2.isEmpty ()) {

int temp = s2.pop();
 s1.push (temp);

}

// remove

public static int remove () {
 if (isEmpty ()) {

System.out.println ("Queue is empty");

} return -1;

return s1.pop();



① add

- ② S1 empty add here ✓
- ③ S1 if not S2
transfer all elements

② S1 push

- ④ S2 S1
put all elements back

Use the other bucket
to store the elements
while adding element in
first stack.

③ peek

```
if (isEmpty ())  
    System.out.println ("Queue is empty");  
    return -1;
```

return s1.peek();

}

Stack using 2 Queues

// add (push)

- ↳ add the 1st element in Q₁
- ↳ then add the elements in the queue that have already elements.

Pop

- ↳ put the elements one by one in another queue until we get to the last element.
- ↳ then at last element return the last element.
- ↳ For another pop use the filled queue as the primary one and other as secondary and and repeat the above process.

static class Stack {

```
static Queue<Integer> q1 = new LinkedList<>();
static Queue<Integer> q2 = new LinkedList<>();
```

1 2 3 4

Q₂

Stack

//empty

```
public static boolean isEmpty() {
```

```
    return q1.isEmpty() && q2.isEmpty();
```

//add

```
public static void push(int data) {
```

```
    if (!q1.isEmpty()) {
```

```
        q1.add(data);
```

```
    } else { q2.add(data); }
```

```
}
```

//remove - O(n)

```
public static int pop() {
```

```
    if (isEmpty()) {
```

```
        System.out.println("Stack is Empty");
```

```
    return -1;
```

there are two cases

int top = -1;

```
if (!q1.isEmpty()) {
```

```
    while (!q1.isEmpty()) {
```

```
        top = q1.remove();
```

```
        if (q1.isEmpty()) {
```

```
            break;
```

```
        q2.add(top);
```

```
}
```

```
else { // case 2
```

```
    while (!q2.isEmpty()) {
```

```
        top = q2.remove();
```

```
        if (q2.isEmpty()) {
```

```
            break;
```

```
    q1.
```

First non-repeating letter in a stream of characters

Flipkart

ex- aabccxb T.C \rightarrow Big O(n)

① freq [] = [26] - for 'a' to 'z' storing the frequency of each character.

② Queue <character>

```
for (int i=0; i<str.length(); i++) {
    ch = str.charAt(i);
    q.add(ch);
}
```

output

a -> b bb X

```
freq [ch - 'a'] ++;
```

```
while (freq [q.peek()]) {
```

```
    q.remove();
}
```

```
if q.empty()  $\rightarrow$  -1
```

```
else print (q.peek());
```

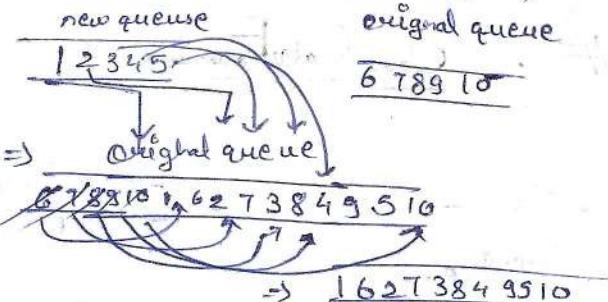
}

* Almost in all questions in which it is written "stream of characters" Queue is used. * Almost all !

Intercleave or shuffle 2 halves of Queue (evenLength)

ex- 1 2 3 4 5 6 7 8 9 10

\Rightarrow 1 6 2 7 3 8 4 9 5 10



Step ① - size of q.size()

② Queue first \rightarrow size/2 // make a new queue with first half elements original que.

③ remove first element from new queue and add at back of original que.

④ then transfer the first element of original at last

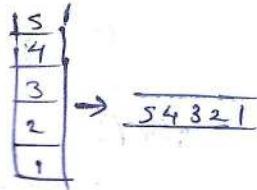
Repeat this until the Queue first became empty.

Queue Reversal

ex - 1 2 3 4 5

final → 5, 4, 3, 2, 1.

- ① Put the elements in stack 1 2 3 4 5 →
- ② put back in queue



Deque (Double ended Queue)

* This is different
from Dequeue

→ Queue of two mouth.

This means removing
elements from
queue.

Methods

- 1) addFirst();
- 2) addLast();
- 3) ~~removeFirst();~~
- 4) removeLast();
- 5) getFirst();
- 6) getLast()

Meaning as you
understand by
name.

Implement using JCF (Java collection Framework)

Java Collection Framework

Class -



---> Implements

Interface -



→ Extends.

ArrayList -

Vector - - -

↑

Stack

→ List → Collection

Linked Hash set -

Hash set - - -

Set ← Sorted set ←

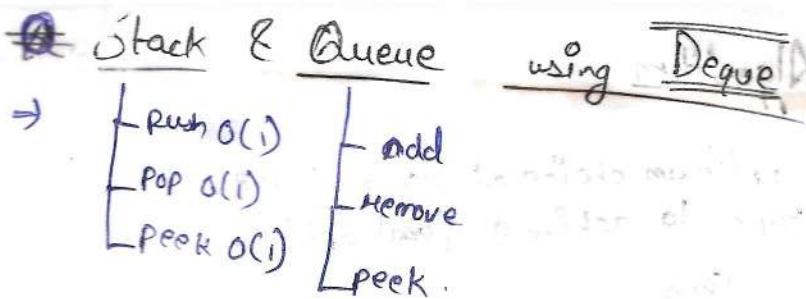
Tree set

↓

Queue ← - - - Priority Queue

Deque ← - - - Array Deque

LinkedList -



⇒ Stack.

Push → addLast()

Pop → removeLast()

Peek → getLast()

Queue

add → addLast();

remove → removeFirst();

Peek → getFirst();

Sout.

Greedy Algorithm

↳ locally optimum choice at each stage.
and hope to achieve a global optimum.

Pros

- ↳ simple and easy
- ↳ good enough T.C

Cons

- ↳ A lot of time
- ↳ last solⁿ is not optimum.



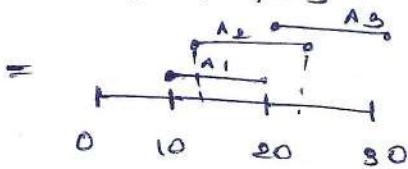
Activity Selection

- ↳ start time and end time of a activity will be given
- one man can do only one work at a time (No multitask)
- find maximum activities that can be performed ?

Ex -

$$\text{start} = [10, 12, 20]$$

$$\text{end} = [20, 25, 30]$$



so max activities = 2

A1 and A3

→ because if we start
A2 the start time of
A3 will be missed and
we will be able to do
only one activity.

→ So the question is sorted with end time
if end time basis sort
A1 first activity

↳ end time

Max no. of activities → Non overlapping (disjoint)

↳ start time >= last chosen action

Count ++

end time

index	start time	End time
0		
1		
2		
3		
4		

Steps

→ ① sort on end time basis

→ ② first end time will be the end time of last task.

③ for ($i=0$; $i < \text{end.length}$)

if ($\text{start time} \geq \text{end time}$) {

 maxAct++;

 ans.add(that activity index)

 last end = activities[i][2];

Sorting in 2D Array

→ Arrays, sort(activities, Comparator, comparingDouble(0 → 0[2]))

0x - 0	1	2	3
Row 1	.	.	.
Row 2	.	.	.
Row 3	.	.	.

Column 0 1 2 → on the basis of this column
the table will sort.

enrich
column the
data should
be sorted

Fractional KnapSack *

→ You are given a bag which can hold limited Kg of things, You have different things in different kg's and their sale value is different choose the things so that you have max value.

$$\text{Value} = [60, 100, 120]$$

$$\text{weight} = [10, 20, 30]$$

$$\text{bag can hold} = W = 50$$

for (int i = n-1; i >= 0; i--) {

if (capacity >= weight[i]) {

total include → capacity - weight[i]

$$\rightarrow \frac{\text{value}}{\text{weight}} + \text{val}[i]$$

else {

$$\text{value} = \text{val} + \left(\frac{\text{ratio}}{\text{capacity}} \right) \cdot 2^D$$

break; }

}

}

① steps → find the rate ratio

② → make a 2D array having
[index, weight/price]

③ → if bag can't hold every thing
add every thing.

④ → rather than add that
much that the bag can
hold.

⑤ → priority with max rate
object first.

⑥ → if not sorted sort the
2D array descending of rate
columns.

Min Absolute Difference Pairs

$2-1 = 1$ (Even)

$1-2 = -1 \rightarrow |1-1| = 1$

So here absolute difference is 1.

↳ We are given two arrays we need to make pairs betⁿ their element as the sum of their absolute difference is minimum.

Ex- $A = [1, 2, 3]$
 $B = [2, 1, 3]$

ans = 0

→ So to get minimum sum the difference bet the elements should be ~~as~~ minimum.

→ take the difference betⁿ the elements as much closer to each other.

→ For this sort both arrays and take difference as their index wise.

→ This will give you minimum difference.

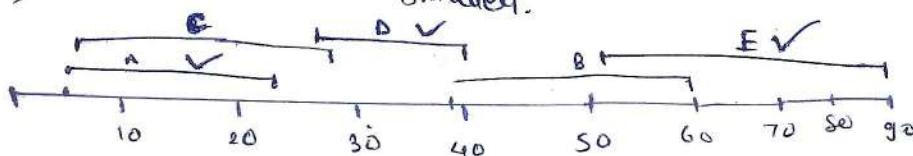
Max Length Chain of Pairs

↳ Here some pairs of elements are given you need made a chain with them but if (a, b) and (c, d) are a pair and in chain then $b < c$ → next.

Ex- Pairs =
 $(5, 24)$
 $(39, 60)$
 $(5, 28)$
 $(27, 40)$
 $(50, 90)$

Approach → this question is same as activity selection question.

→ If we arrange them in a line then the next pair will only come there if the earlier pair right element is smaller.



→ So we can pick A D E
 Max chain is 3 pairs.

Approach (Make a 2D array of [index, element, 1st element, 2nd element])

- ① Sort on the basis of second element.
- ② first pair will be the first indexed pair.
- ③ Last element = 2nd element of first pair.
- ④ for ($i=1; i < n; i++$) {

 if ($\text{pair}[i](\text{start}) > \text{last end}$) {

 ans++; // increasing element in chain.

 last selected end update
 }

Indian coins

If you are given a mount say 590 you need to give this much to some one in minimum no. of notes

ex - You have $[1, 2, 5, 10, 20, 50, 100, 500, 2000]$.

So you will give 500×1

50×1

20×2 .

→ So here we have used Greedy Approach to solve this - because we have used minimum no.

Off notes:-

but this approach will not work on every note system
India has Canonical coin system so it works here.

Approach-

① Sort descending $[2000, 500, 100, 50, 20, 10, 5, 2, 1]$

② count = 0.

amount = 590

for ($\text{int } i=0; i < n; i++$) {

if ($\text{coin}[i] \leq \text{amount}$) {

while ($\text{coin}[i] \leq \text{amount}$) {

count ++

amount -= $\text{coin}[i]$; } else

}

* Average sort (coins, Comparator reverse Order())

to sort the array in reverse order.

But for this Integer this must be used to make array not primitive.

Job Sequencing Problem

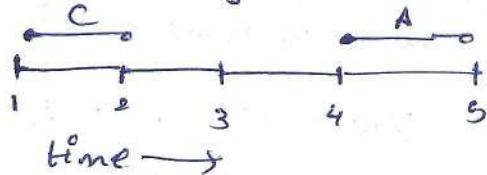
→ We are given some jobs, their deadlines and the payment we will get after doing them. each job takes 1 hour.

ex-	deadline	Payment
Job A =	4 ,	20
Job B =	1 ,	10
Job C =	1 ,	40
Job D =	1 ,	30

and each job only can be started before their deadline

So use greedy to get max profit

→ we will do job C = 40



Approach

→ ① Sort Jobs (based on profit)

② time = 0

for (int i=0; i<jobs; i++) {

if (job(deadline) > time) {

add job in ans;

time++;

}

}

→ now only job A can be done because it had dead line

4.

$$\Rightarrow \text{profit} = 40 + 10 \\ = 50$$



~~Collections.sort(jobs, (obj1, obj2) → obj2.profit - obj1.profit);~~

↳ code to sort objects.

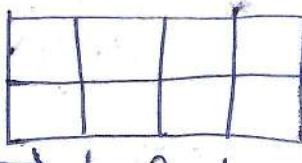
Chocola Problem

Microsoft
Google

04 Min Cost to cut board into squares.

→ We have give a chocolate bar.

Ex -



we need to break this into small pieces such that there should be a single piece.

but the twist is here is that each cut cost something and the cost differ for each cut.

So we have to cut the chocolate in such a way that it cost minimum for cutting it.

4 imp things

- i) We have to perform each cut
- ii) After each cut chocolate breaks in pieces & then for each cut we need to pay separately
- iii) So we will do
 - a) expensive cut first
 - b) cheap cut at last.

iv) Cost of vertical → $\frac{\text{no. of horizontal pieces}}{\text{cut price}}$.

Cost of horizontal cut → $\frac{\text{no. of vertical pieces}}{\text{cut price}}$.

We will make two array

$$\text{vertical cut} = [4, 3, 2, 1, 1]$$

$$\text{horizontal cut} = [4, 2, 1]$$

→ Move the pointer on both

→ perform the cut first which is costlier

{ sort hor, verti in decreasing order

int h=0, v=0;

int hori = 1, verti = 1

while (h < hori.length & v < verti.length)

 h < v ? verti++ : else ? hori++;

At last remain hori or verti cuts.

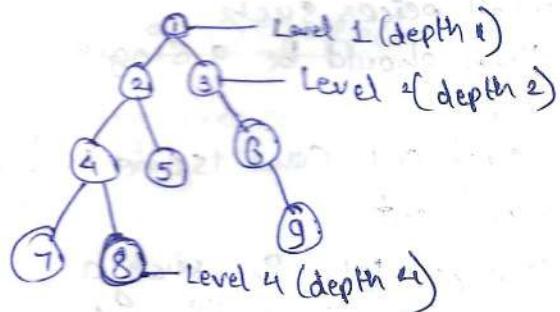
Pseudo code

and add
cost++;

Binary Tree

→ This tree consist of max 2 nodes/no.

• Levels and Subtree



No. of leaves $\rightarrow 4$

④ ⑧ ⑤ ⑨

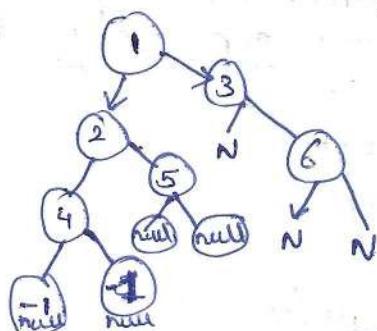
Ancestors of 8

$\rightarrow \{4, 2, 1\}$

• Build Tree Preorder

1, 2, 4, -1, -1, 5, -1, 7, 3, -1, 6, -1, -1

→



-1 = Null:

- First take left node then right.
- Null has no further node.
- So then choose right

DFS → Depth first search

BFS → Breadth First Search

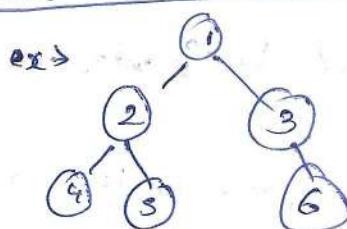
Tree Traversals

A) Preorder → Prebecause root is printed first.

- i) Root
- ii) Left Subtree
- iii) Right subtree

T.C) = $O(n)$

$\Rightarrow 1, 2, 4, 5, 3, 6$



DFS

public static void preOrder(Node root) {

// base case

if (root == null) {

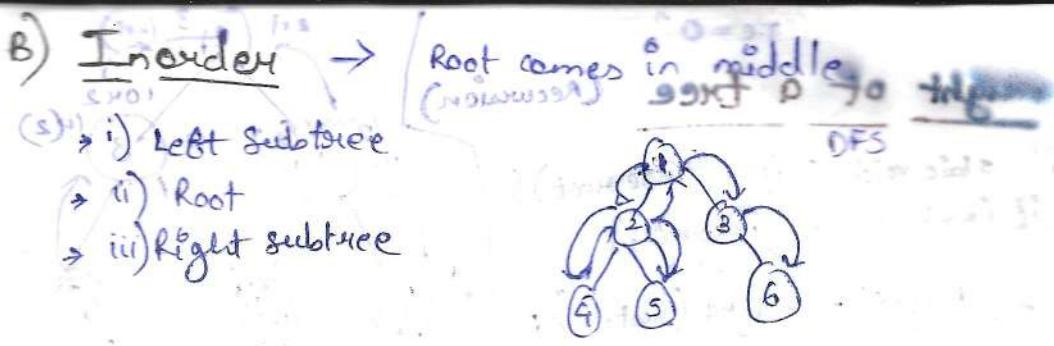
System.out.print(" ");

return;

System.out.print(root.data + " ");

preOrder(root.left);

preOrder(root.right);

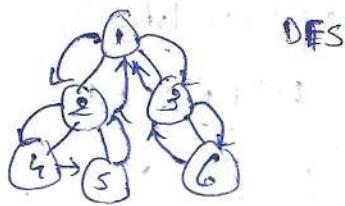


⇒ 4 2 5 1 3 6

```
public static void InOrder(Node root){  
    if (root == null){  
        return;  
    }  
    InOrder (root.left);  
    System.out.print(root.data + " ");  
    InOrder (root.right);  
}
```

c) Postorder → root will be at last

- i) Left subtree
- ii) Right subtree
- iii) root.



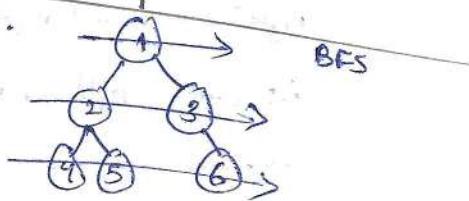
```
public static void postOrder(Node root){  
    // base case  
    if (root == null){  
        return;  
    }  
    postOrder (root.left);  
    postOrder (root.right);  
    System.out.print(root.data + " ");  
}
```

d) Level Order → print level wise.

$T.C = O(n)$

In this we will use a Queue

- 1) put root in queue
- 2) print element from queue (pop Q)
- 3) add left and right element of root to Q of last element
- 4) also add null after adding both left and right leaves so to know about next line.
- 5) another null will be added when the first null will be out of queue



⇒ 1, 2, 3, 4, 5, 6

out of queue

Height of a tree (Recursion) T.C = O^n

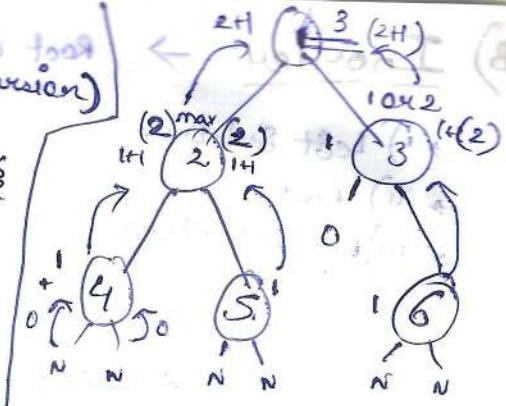
```

    → static void height (Node root) {
        if (root == null) { return 0; }

        leftHeight = height (root.left);
        rightHeight = height (root.right);

        height = max (lh, rh) + 1;
    }

```



Count of Nodes of tree (Recursive)

```

    static void Count {
        if (root == null) {
            return 0;

        leftCount = count (root.left);
        rightCount = count (root.right);

        int treeCount = L.C + R.C + 1;
    }
}

```

Sum of Nodes (Recursive)

```

    static int sum {
        if (root == null) {
            return 0;

        leftSum = sum (root.left);
        rightSum = sum (root.right);

        int sum = leftSum + rightSum + root.data;
    }
}

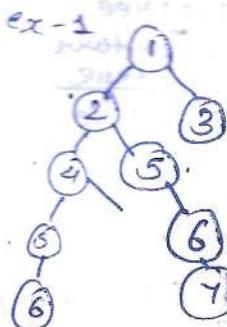
```

Diameter of a Tree

T.C = $O(n^2)$

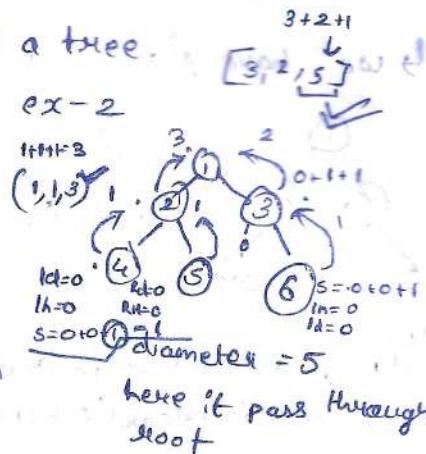
(Approach no. 1)

→ largest distance betⁿ two node of a tree.



diameter = 7

here it does not pass through root node.



here it pass through root

• diameter of that Node = $\left(\text{left diameter} + \text{right diameter} \right) + 1$

→ static int diameter(Node root){
if (root == null) {
return 0;
}

int left Dia = diameter (root.left);
int right Dia = diameter (root.right);
int left HT = height (root.left);
int right HT = height (root.right);

int selfDiameter = leftheight + Right height + 1

return Math.max (selfDiameter, Math.max (left Dia, right Diameter))

Diameter (Approach 2)

class Info {
int diam
int height
}

↳ to save the time we will return both height + diameter using a class.

⇒ left Info = diam (root.left) → includes [diam, height]
right Info = diam (root.right)

newInfo [final Diam = max (lInfo.dia, rInfo.dia, lh + ht + 1)
final HT = max (lh, rh) + 1]

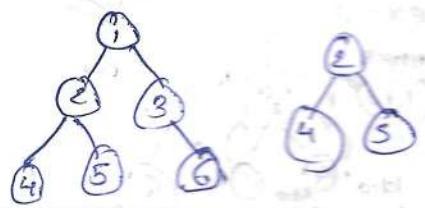
return newInfo (Diam, ht);

Subtree of another tree

↳ we have given a full tree and a sub tree.

we have to find out if the subtree is a subtree of Main tree.

→ Node and position both should be same.



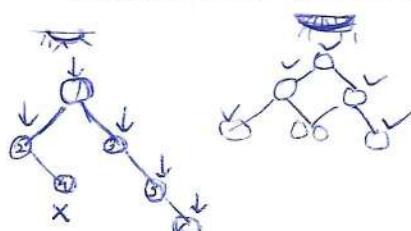
Step

- ∴ ① find subroot in tree ↳ using any traversal
- ② check identical (subtree, node subtree)

Return false if

- ① node.data != subroot.data
- ② node == null || subroot == null
- ③ left subtree → non identical
- ④ right subtree → non identical

Top view of a tree



Output : 2, 1, 3, 5, 6

Hash Map →

map(key, value);

① Create

HashMap<Integer, Integer> map = new HashMap<

② Add (put)

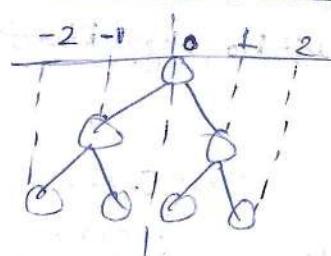
map.put(key, value)

③ Get

map.get(key).

* Key is always unique

Horizontal Distance



we will use level order traversal

↳ if we move root.right → Hd + 1

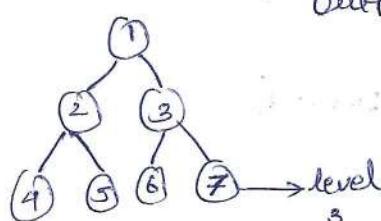
↳ if we move root.left → Hd - 1

Kth Level of a tree

• time Complexity = O(n)

$$2^x - 1 = 3$$

Output : 4 5 6 7



* first level will be zero.

Pseudo code:

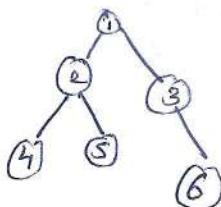
```
k level (root, level, k) {
    if (root == null)
        return;
    if (level == k) {
        print (root.data) } as we don't
        need to
    } return; go further
    klevel (root.left, level+1, k)
    klevel (root.right, level+1, k) }
```

Lowest Common Ancestor and ~~find smallest node~~

(LCA)

↳ the node at (first node) where the meet if we go up in tree

example -



$$n_1 = 4, n_2 = 6$$

$$\text{output} = 1$$

$$n_1 = 4, n_2 = 5$$

$$\text{output} = 2$$

HashMap<>();

Approach

Make array of path from leaf to root

$$a) 0x-6 \rightarrow [1 \ 3 \ 6]$$

$$b) 4 \rightarrow [1 \ 2 \ 4]$$

Now traverse both array and the last common element will be the LCA

Path finder function

- 1) check if the node node = value
if yes add to array list return
- 2) if no add to array list and check left and right of that node.
- 3) if both right and left return null means node is not available in that direction.
- 4) Remove that node and return false
- 5) if got false from the ahead node remove the current node and return false.
- 6) Return true only if get the key

Optimized approach $\xrightarrow{\text{TC}} O(n)$

- 1) $\text{root} = \text{null}$ }
 $\text{root} = n_1$ }
 $\text{root} = n_2$ }
 return root

$\text{leftLca} = (\text{Node.left}, n_1, n_2);$

$\text{RightLca} = (\text{Node.right}, n_1, n_2);$

② $\text{rightLca} = \text{null} \rightarrow \text{return leftLca};$

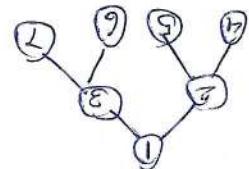
$\text{leftLca} = \text{null} \rightarrow \text{return RightLca};$

$\text{right & Left} \neq \text{null} \rightarrow \text{return root};$

Steps ① Find my node

① OutPut = Node ①

$$\begin{aligned} k &= 2 \\ n &= 5 \\ \text{etc.} &\end{aligned}$$



Kth Descendant of Node

~~if left = -1, right = -1~~
~~else if (right = -1)~~
~~return left + 1~~
~~right += 1~~
 return right + 1

~~if (left = -1, right = -1)~~
 return 0;
 if (right = -1, right = -1)

left n search
right n search

return 0;
if (root == null)

return -1;
if (root == null)

minDistance = dist1 + dist2

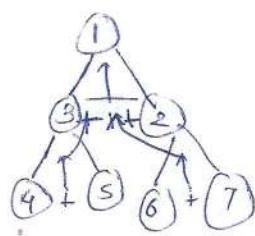
② dist2 lca to n2
① dist1 lca to n1

applicable \rightarrow

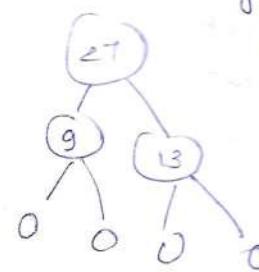
Min distance between two nodes

Transform to Sum Tree

Each node = sum of left and right subtree.



→



approach

```

transform(node root)
{
    if (root == null) {
        return 0;
    }
    int leftChild = transform(root.left);
    int rightChild = transform(root.right);
    int data = root.data;
    if (root.left == null) {
        root.left = newLeft;
    }
    if (root.right == null) {
        root.right = newRight;
    }
    root.data = newLeft + leftChild + newRight + rightChild;
    return data;
}
  
```

Time Complexity: O(n^2)

Space Complexity: O(n)

Time Complexity: O(n)

Space Complexity: O(n)

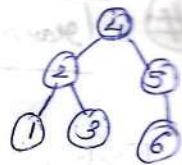
Time Complexity: O(n)

Space Complexity: O(n)

Binary Search tree

↳ Same as binary tree with some extra properties.

- i) Left Subtree Nodes < Root
- ii) Right Subtree Nodes > Root
- iii) Left and right Subtree are also BST with no duplicates.



* So here Inorder Traversal → give sorted sequence

BST search Time Complexity = $O(n)$

→ if ($k = \text{root}$) { return; }
if ($k > \text{root}$) Search at right side
if ($k < \text{root}$) Search at left side

→ But in skewed tree ex-
Worst case

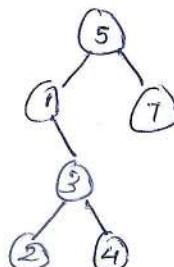
Common Strategy → Most problems will be solved using recursion
i.e. by dividing into sub problems and making recursive calls on subtrees.

Build a BST

→ Values [] = {5, 1, 3, 4, 2, 7}

- Inorder traversal

→ 1, 2, 3, 4, 5, 7

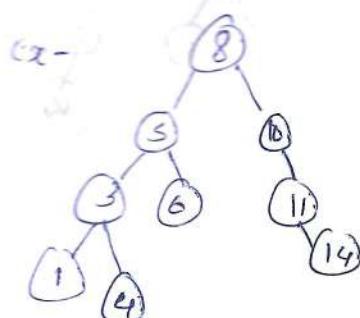


Approach → values[i] > root → put in right subtree
values[i] < root → put in left subtree.

Search in BST

T.C = $O(H)$

$H = \text{height}$ of tree



key = 1

$\begin{cases} \text{① } \text{Root} > \text{key} \end{cases}$	{ search in left tree }
$\begin{cases} \text{② } \text{Root} < \text{key} \end{cases}$	{ search in right tree }
$\begin{cases} \text{③ } \text{Root} = \text{key} \end{cases}$	↳ return true
$\begin{cases} \text{④ } \text{Else if root == null} \end{cases}$	{ return false }

Delete a Node :

Approach

- ① find the node
- ② if case (1) return null to the parent.
- ③ and the node will be deleted automatically.

Case II

- ↳ Replace the node with child node

Case III

- ↳ If the node have 2 children

then the node should be deleted

and the order of BST should also not be disturbed.

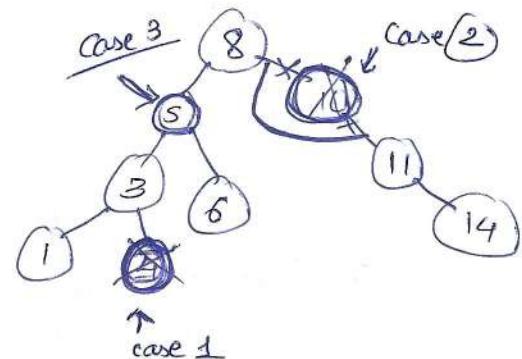
∴ we need to replace that node with inorder successor

- ① Replace value with inorder successor
- ② Delete the node for inorder successor

Inorder successor → Left most node in right subtree.

Cases

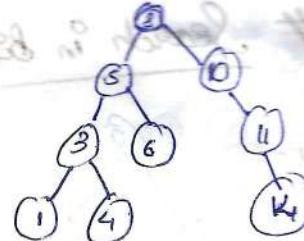
- 1) No child (Leaf Node)
- 2) One child
- 3) Two child.



Print in Range

Example from $k_1=5$ to $k_2=12$

output should be - 5, 6, 8, 10, 14.



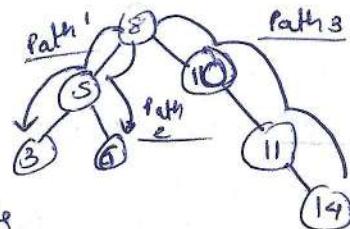
Case I if $k_1 \leq \text{root} \leq k_2$

$\begin{cases} L, R \\ R, R \end{cases}$ If the value of node lies between the range print it.

Case II

$\begin{cases} \text{root} > k_2 \\ \text{root} < k_1 \end{cases}$ else $\begin{cases} R, L \\ L, R \end{cases}$ only.

Root to Leaf Paths



so we will store this path in array lists.

Approach :

- ① Add node to array list
- ② traverse left
- ③ if found null \rightarrow print path. base case.
- ④ backtrack to parent node and remove the node from array list
- ⑤ then traverse right.

Validate a BST

\Rightarrow BST will be given and validate it.?

• Approach 1

If inorder is sorted
then \rightarrow BST valid ✓

Left subtree \rightarrow min = min
max = root/parent

Right subtree \rightarrow min = root/parent
max = max

• Approach 2

- ① take out max of left side = ml
- ② take out min of right side = mr

if $(ml < \text{root} < mr) \{ \text{then BST valid} \} \checkmark$

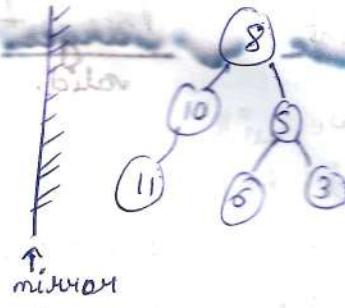
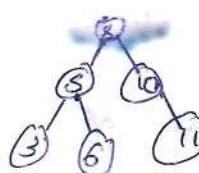
Mirror a BST

Approach

① take out mirror of left subtree

② take out mirror of right subtree

③ swap them.



→ Mirror → ~~root <--> root~~ root.left <--> root.right

Sorted Array → Balanced BST

arr = {3, 5, 6, 8, 10, 11, 12}

To get minimum height → find mid

recursively → take it as root

→ half elements at left and half elements at right

Pseudo code

→ CreateBST(arr, st, end) {
if (st > end)

 int mid = (st + end) / 2
 return null;

 Node root = newNode(arr[mid])

 root.left = CreateBST(arr, st, mid - 1)

 root.right = CreateBST(arr, mid + 1, end)

 return root;

Convert a BST to Balanced BST

Approach → ① take out Inorder sequence of BST

② convert the arr to Balanced BST

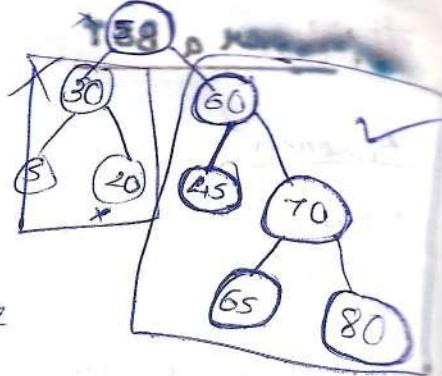
Print

largest BST valid.

↳ Here we will use the logic of isBST valid

↳ But we will make class of "Info"

This whole thing will be calculated for each node $\left\{ \begin{array}{l} \text{① } \text{IsBST (boolean) True or false} \\ \text{② } \text{size} = \text{left size} + \text{right size} + 1 \\ \text{③ } \text{min} = \text{math.min}(\text{root.data}, \text{Left min}, \text{right min}) \\ \text{④ } \text{Max} = \text{Math.max}(\text{root.data}, \text{Left Max}, \text{right Max}) \end{array} \right.$

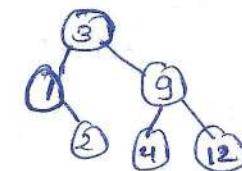
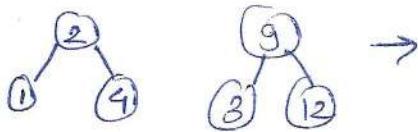


↳ we will make a static variable (max size) when the isBST \rightarrow true

$\text{max} = \text{Math.max}(\text{max}, \text{curr Info});$

↳ And we will return max size.

Merge 2 BSTs



Balanced BST

Approach →

- ① BST 1 inorder sequence (sorted)
- ② BST 2 inorder sequence (sorted)
- ③ Merge both arrays.
- ④ Sorted array \rightarrow Balanced BST

AVL Trees

→ only concept asked

Self-Balancing BST

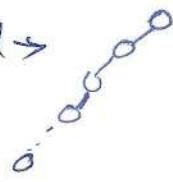
↳ we required Balanced tree to reduce T.C

for ex - 10^5 nodes if it's skewed \rightarrow tree

then $T.C = 10^5$

but if balanced $\rightarrow 5 \log 10$
then

And $5 \log 10 \ll 10^5$



Heaps / Priority Queues

↳ give first priority to the element you want.

• Priority Queue using ICF

↳ Priority Queue < Integer > pq = new Priority Queue < >();

Functions

At default the priority in integer is from lowest \rightarrow highest number.

- ① add() TC: $O(\log n)$
- ② remove() TC: $O(\log n)$
- ③ peek() TC: $O(1)$
- ④ isEmpty()

To travel through queue for each loop

```
→ for(int x : q) {  
    if(x == k) return true;  
}  
return false;
```

To change the priority (descending type) add this

Priority Queue < Integer > pq = new Priority Queue < > (Comparator. reverse

Priority Queue of objects

→ For this we need to make our class comparable

(Means give it the power so that it can compare)

```
static class Student implements Comparable<Student> {
    string name;
    int rank;

    public Student(string name, int rank) {
        this.name = name;
        this.rank = rank;
    }
}
```

④ Override

```
public int compareTo(Student s2) {
    return this.rank - s2.rank;
```

this rank - S₂ rank

$$S_1 \begin{bmatrix} 15 \\ 15 \end{bmatrix}$$

$$S_2 \begin{bmatrix} 15 \end{bmatrix}$$

if $S_1 < S_2$ - ve $\rightarrow S_2$ is greater

$S_1 > S_2$ + ve $\rightarrow S_1$ is greater

$S_1 = S_2$ \Rightarrow equal.

Output: Output will be sorted on the basis of rank

Ex - A - 2

C - 3

D - 4

B - 8.

* To make the order reverse

we (Comparator. reverseOrder())

in side Priority Queue argument section

4. Reverse Order();

- o Heap
 - Max heap \rightarrow PQ max. \uparrow priority element
 - Min heap \rightarrow PQ min. \uparrow priority element

- Properties

① Binary tree

↳ Atmost 2 children

② Complete binary Tree (CBT)

↳ CBT is a binary tree in which all levels are completely

filled except, possibly the last one

↳ It is filled from left to right, before new leaf.

↳ Means a level should be completely filled before starting to put the elements in new level

Ex -



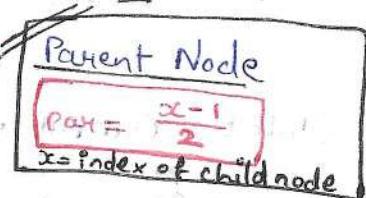
③ Heap Order Property

↳ Children node \geq Parent (**Min Heap**)

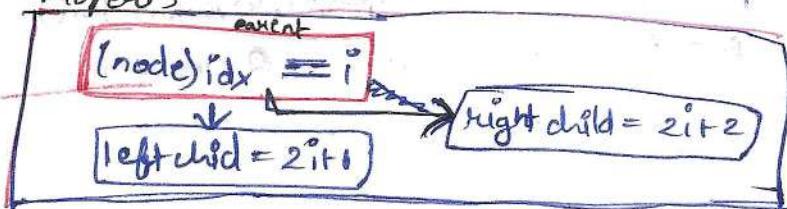
↳ Children node \leq Parent (**Max Heap**)

Heap Implementation ← we do it with the help of arrays

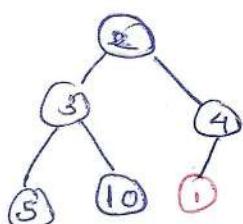
- We cannot use class using class because then the add function will take more than $\text{Log}(n)$ time.
- we use Array or ArrayList.



Properties



Insert in Heap T.C = $\log(n)$



2 | 3 | 4 | 5 | 10

→ 0 1 2 3 4 5 6 7 8 9 10
2, 3, 4, 5, 10, 1

parent swap

$$\Rightarrow \text{par} = \frac{5-1}{2} = 2$$

→ 0 1 2 3 4 5 6 7 8 9 10
2, 3, 1, 5, 10, 4

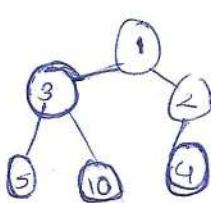
$$\text{par} = \frac{2-1}{2} = \frac{1}{2} = 0$$

Step ① - add at last index

Step ② - fix heap.

Step ③ - par = $\frac{x-1}{2}$ while

{Step ④ - if (child val < par val)
swap (child, par)}



(child < par) = swap

→ 1 3 2 5 10 4

Get min in Heap.

↳ we know that in Min Heap minimum element of the Tree is root

`Peek()`

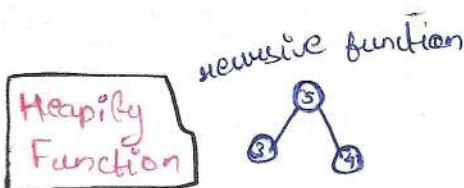
```
public int peek () {
```

```
    return arr.get(0);
```

}

Delete in Heap $\rightarrow TC = O(log n)$

Q. Remove min element



→ ① It compares root, leftchild, rightchild

→ ② put the min at root = i

→ 2nd min at left root = $2i+1$

→ and large at Right root = $2i+2$

Steps 1) swap 1^s and last node

2) remove last index in array list
 $arr.remove(arr.size() - 1)$.

3) fix heap
 ↓ using

heapify

Heap Sort \rightarrow Accending
 ↓
 $arr = 1, 2, 3, 4, 5$ (Max heap) Depending
 ↓
 (Min heap)

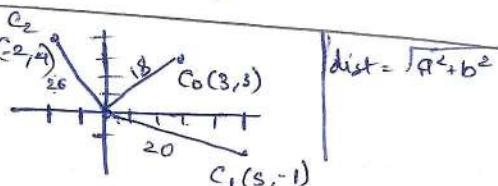
Step 1) $arr \rightarrow$ Max heap
 ↓
 extract level
 Non-leaf Nodes

heapify()
 ↓ (maxheap)

Step 2) \rightarrow Swap (1^s and last)
 \rightarrow heapify ~~remove last~~
 ↓
 for (0 to n-1)

- Max heap will bring the largest element at top (root)
- then exchange it with last
- heapify (size 1 se kam karke)
- first apna ~~for heap~~ array sorted

Nearby Cars \rightarrow we are at origin $(2, 2)$
 Q. find K nearest cars? $K=2$



distance from origin \rightarrow

Approach

① → find the distance

② → Put that value in Priority Queue

③ → take out first K values as your ans

$C_0 \rightarrow 18$

$C_1 \rightarrow 26$

$C_2 \rightarrow 20$

④ @Override

public int compareTo (Point p2) {

return this.distSq - p2.distSq; }

Comparing on the basis of dist Sq

Solve

- Connect N Ropes → but each time when 2 ropes connected they have charge. charge = length of rope \times length of another rope
- we want to connect all rope in minimum charge
- As the charge is added each time so to have a minimum charge we will connect smallest hoop first. Doing this the charge will be lowest

	Charge
2 + 3	5
5 + 4	9
9 + 6	15
	29 //

- Approach
- ① Add all elements in PQ
 - ② Remove first element
 - ③ add them and push in PQ
 - ④ do above (3 and 4) until PQ gets ~~empty~~ 1 element
 - ⑤ last removed is answer.

* This question looks like solved by Greedy algorithm as just sort and start adding them but No it gives WRONG Answer

Weakest Soldier → 1 represent soldier 0 represent civilians.
 ex. $m=4, n=4, k=2$ Row is weak if
 → ① It is at top
 ② It has less no. of soldier.

Row is weak if
 → ① It is at top
 ② It has less no. of soldier.

Q. Find weakest (first two row) ?

ans → R0, R2

Approach:

static class Row implements Comparable<Row> {

int soldiers;
int idx;

public Row(int soldiers, int idx) {

this.soldiers = soldiers;

this.idx = idx;

else compare on
the basis of
soldiers.

else {

return this.soldiers - st2.soldiers;

② Override

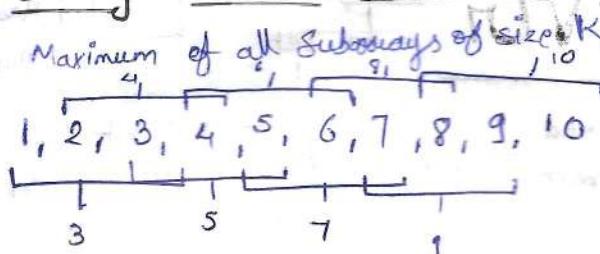
public int compareTo(Row st2) {

if(this.soldiers == st2.soldiers) {

return this.idx - st2.idx;

If soldiers are
same then compare
on the basis
of index

Sliding Window Maximum



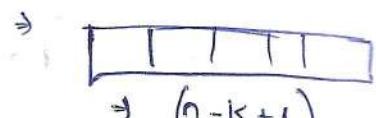
$k=3$
window of
length 3

we need to take out maximum of three element

Approach

we will make max heap

length of result array



① add K nums to PQ (1st window)

② ① PQ.peek() \rightarrow window[0]

② ② while ($PQ.peek().idr \leq (i-K)$)

{ remove () }

③ PQ.add (element next from end)

④ window[i] = PQ.peek()

⑤ Adding many elements

in PQ will do no harm

at the top there always be
the max element.

⑥ but check whether the
element is from that
current window or not

→ After making Priority Queue

here as in the
PQ the top element
will be maximum
then check whether
the element at top
is of that current
window or Not

→ if not then remove
it

→ otherwise keep it
and then add the next
element.

// 1st window

for (int i=0; i<k; i++) {

| PQ.add (new Pair (arr[i], i));

} res[0] = PQ.peek().val;

- adding elements of 1st window

for (int i=k; i<arr.length; i++) {

| while (PQ.size() > k && PQ.peek().idr <= (i-k)) {

| | PQ.remove();

| PQ.add (new Pair (arr[i], i));

| res[i-k+1] = PQ.peek().val;

}

if the
element in
top is not
of current
window

HASHING

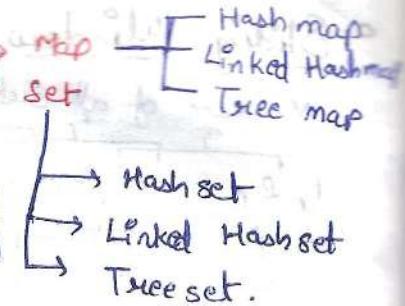
HashMap

(key, value)

↓
always
unique

↳ unordered
[we get data back in random form]

import java.util.HashMap;



HashMap < String, Integer > hm = new HashMap<>

columns ←
one datatype columns ←
2 datatype

- * ① Put(key, value) → to update value
→ hp.put(key, value)
 - * ② get(key) → hp.get() → or to make new node
 - * ③ containsKey(key)
 - * ④ remove(key) → hp.remove(key)
 - hp.remove()
- To get the value
hm.get(key);
if (key) is not present then
you will get null value.
- To check if the data with that
key is present or not

→ To remove (key and value)
Node.

if passed a [key] ← Not existing

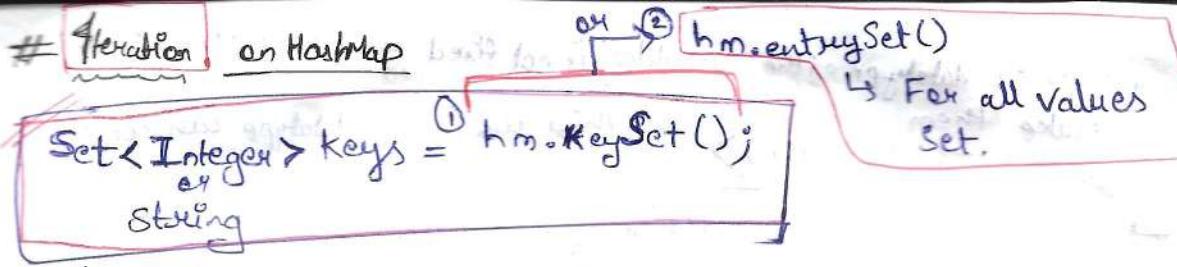
→ Null keys values

	keys	values
"one"	:	1
"two"	:	2
"three"	:	3
"four"	:	4

* ⑤ Size
→ hp.size();

* ⑥ Empty → hp.isEmpty();

* ⑦ Clear → hp.clear();



↳ this stores all the keys in set

ex - from last keys = [one, two, three, four]

• foreach →

```
for (String k : keys) {
    System.out.println("key=" + k + ", value=" + hm.get(k));
}
```

here → k will like iterator.
→ it will iterate till the end.

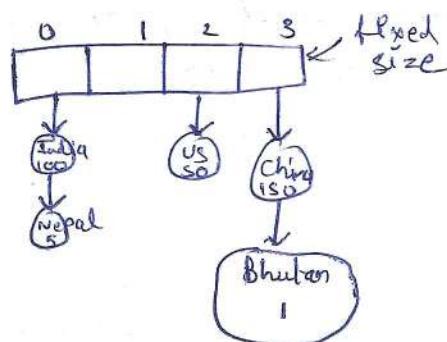
Implementation HashMap

Was asked in Uber to implement hash

Put(), get(), containsKey(), remove(), size() $O(1)$

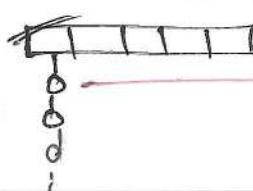
→ this all works in the Java in the form Array of Linked List

Map	
key	value
"India"	100
"China"	150
"US"	50
"Nepal"	5



special Note

Rare case
Worst time complexity will be $O(n)$. if we have such data whose bucket index giving by hash function is same



* When the datatype of the variables is not fixed we use generic to make them
→ ex $\langle K, V \rangle$ this way any datatype can come
→ used in implementation of hashmap.

Linked Hash Map → everything same just it maintains the order in which the elements are inserted in it
Keys are insertion order → It uses Doubly Linked List to maintain the order.

Tree Map → put, get, remove are $O(\log n)$ (BST)
→ Keys are sorted → Here in implementation Red Black trees are used which are self-balancing trees

Q. Majority Element → $Tc = O(n)$

given an arr of n , size find the elements that appeared more than $[n/3]$ times.

⇒ $\text{nums}[] = \{1, 3, 2, 5, 1, 3, 1, 5, 1\}; \quad n = 9$

output = 1

1 is $[n/3 = 3]$ greater

ex $\text{nums}[]$
 $\{1, 2\}$

for ($i=0$ to arr.length)

output $\{1, 2\}$

$\text{num} = \text{arr}[i]$

if ($\text{map.containskey}(\text{num})$)

$\text{map.put}(\text{num}, \underline{\text{map.get}(\text{num}) + 1})$;
adding with increasing freqcy.

else $\text{map.put}(\text{num}, 1)$;

To traverse on a Map we need **KeySet**

→ Set<Integer> keys = map.keySet();

for (Integer key : keys) {
 }

* if the key exist then
it will return it's
frequency or and if
not then 0.

Shortcut for increasing frequency

⇒ map.put(arr[i], map.getOrDefault(arr[i], 0) + 1);

Shortcut for "No need to store key set in any other place"

for (Integer key : map.keySet()) {
 if (map.get(key) > arr.length / 3) {
 System.out.println(key);
 }
}

Valid Anagram.

T.C = O(n)

ex - "race" = "care"

same no. of letters →

same value of letter, with same frequency.

Approach :

① HashMap HashMap<Character, Frequency>
 → we will ↑ char ↑ int

② then when we will traverse
the second string we will delete
or remove the elements from
map :

③ if Map is empty return True or False.
if found such a element which
is not available then return False

Hashset → implements using hash map.
↳ Collections of unique elements.

- No duplicates
- Unordered
- Null is allowed

1) add(key)
2) contains(key)
3) remove(key) } O(1)
4) set.size();
5) set.clear();
6) set.isEmpty();

HashSet < Integer > hs = new HashSet <>();

Iteration on Hashset

① Using Iterators

→ Iterator it = set.iterator();
while (it.hasNext()) {
 print(it.next());
}

→ Provides a iterator
if → null
points towards null.

→ Checks if there is
such an element which
is not traversed till now.

→ Then automatically
it shifts to next it.

② Using Advance loop

→ for each.

→ for (key keys
for (String city : cities) {
 System.out.print(city + " ");
}

ex - HashSet < String > cities = new HashSet <>();
cities.add("Nagpur");
cities.add("Mumbai");

Linked HasSet → implemented using linked Hash Map.
using doubly LL.

- Ordered elements.

T.C → remains same
just little performance
drop.

→ Linked HashSet < Integer > lhs = new LinkedHashSet <>();

Tree Set → tree map (Red Black) (self Balancing trees) T.C = O(log N)

- Sorted in ascending order.
- Null is not allowed.
- No duplicates.

TreeSet<String> ts = new TreeSet<>();

Other properties are same.

Q Count Distinct elements

- use set as it contains no duplicates.
- return set size.

Q Union and Intersection of arrays T.C = O(m+n)

$$\text{arr1} = \{7, 3, 9\}$$

$$\text{arr2} = \{6, 3, 9, 2, 9, 4\}$$

a) Union → take a set of all elements.

b) Intersection →
i) add elements of one set
ii) traverse on the array of second.
iii) if the element present in set

↳ a) count it;

b) remove that so can't be count again.
i) point count.

Q. Find Itinerary from Tickets

Journey.

→ print all the places one wise you visited.

i) Find the starting point

↳ a) Make a reverse map.

b) find the place which is the start

c) if the place exists in key

of main map but

not in to

key	value
chennai	Bengaluru
Mumbai	delhi
Goa	chennai
Delhi	Goa

to	from
B	C
D	M
C	G
C	D

→ Then just go from that key to value until last value is not a key.

largest subarray with sum = ~~target~~ 0

ex: $\boxed{\text{target} = 0}$

this much has no
value : 0

brute force → nested loops

$\leftarrow \rightarrow \{15, -2, 2, -8, 1, 7, 10, 23\}$

$\boxed{\text{ans} = 5}$ largest subarray with

if (found and same sum)
then

(Key)	(Value)	Index
15	0	
-2	1	
2	3	
-8	4	
1	5	
7	6	
10		
23		

steps:

MaxSubarray = Math.Max(index)

because as we want sum = 0

so if any key is same then we can
subtract the front and get sum as zero

Count no. of subarrays with sum k

T.C $\Rightarrow O(n)$

Approach \rightarrow $i \boxed{j}$

$\text{arr} = \{10, 2, -2, -20, 10\}$

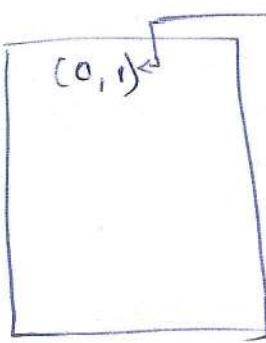
$k = -10$

$i \rightarrow i^{\text{th}} \text{ index}$
 $j \rightarrow j^{\text{th}} \text{ index}$

So $\boxed{\text{Sum}[0, j] - \text{sum}[0, i-1] = \text{sum}(i, j)}$

we will make Hash Map

$$\text{Sum}[j] - \text{Sum}[i] = k$$



$(\text{sum}, \text{count})$

$$\boxed{\text{Sum}[j] - k = \text{Sum}[i]}$$

↑
cumulative sum
↓
given

→ ① First we will add $(0, 1)$ in hash map $\text{sum} = -10$

② We want K_{sum} as a subarray.

So if $\text{current_sum} - K = 0$ then its equal to K

③ Each time we will check the

④ Take sum = ~~$10 \neq 0$~~ did 0 also frequent increase.
 $\text{ans} = 0$

$\rightarrow \text{ans} + 1$

$\rightarrow \text{ans} = 1$

$\text{ans} + \text{get}(10)$

$= (\text{ans} + 2)$

$\rightarrow \text{ans} = 3$

$$12 - (-10) = 22$$

$$10 - (-10) = 20$$

does not exist

does not exist

does not exist

exists

$\text{ans} + (\text{get}(10))$

$$0 - (-10) = 10 \text{ exists}$$

and

$[10, 2, -2, -20, 10]$

↑ ↑ ↑ ↑ ↑

$(0, 1) \leftarrow \text{already}$

$(10, 12)$

$(12, 1)$

$(-10, 1)$

trie

Tries

→ Prefix tree

or
Retrieval tree]

[Retrieval tree]

ex - There

The

The

↑ prefix

Questions
are directly
asked in
Competitions
like O-star
various
universities.

- ↳ same prefix are saved only once
- ↳ used in string question

• What is Trie?

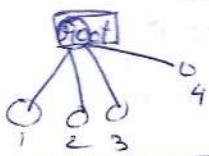
→ It is a (K-way) tree.

→ Saves time as height is less

ex- K=4

→

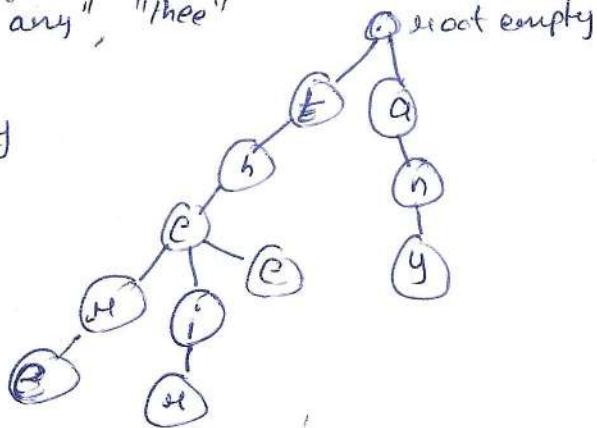
① root (empty)



words [] =

⇒ "the", "a", "there", "their", "any", "thee"

→ As prefix is stored only once so time is saved.



⇒ Class Node {

 Node [] children; 26 letters
 boolean endOfWord;

- Each node has 2 info
- ① the letter it's holding
- ② end of word
- ③ to show which letter is it holding.
- ④ → to check whether the word ends here or not.

Making trie

Static class Node {

 Node children [] = new Node [26];
 boolean eow = false;

and also a empty root node

 Node () {

 for (int i=0; i<26; i++) {
 children [i] = null;
 }

 } public static Node root = new

 ↓ This root
 it is always empty

 just stores which elements are used in tree

Insert in tree

T.C $\rightarrow O(L)$

\hookrightarrow level of letters

when we miss

$t-a \rightarrow$ we will get idx

- ① After the root we will check level by level
- ② like this each letter by letter will be stored.
- ③ And the place where word ends there we will store EOW = true
end of word

Sudocode

```

→ curr = root
for (level = 0 to word.length())
    if (curr.children[ch - 'a'] == null) {
        ↳ new node create
    } else x
curr = curr.children[ch - 'a']

```

```

public static void insert(string word) {
    Node curr = root;
    for (int level = 0; level < word.length(); level++) {
        int idx = word.charAt(level) - 'a';
        if (curr.children[idx] == null) {
            curr.children[idx] = new Node();
        }
        curr = curr.children[idx];
        curr.eow = true
    }
}

```

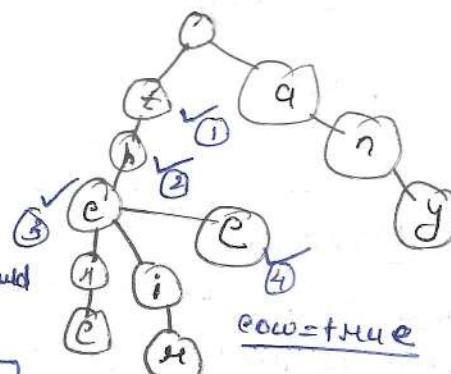
* Can be asked as a separate question

Search in a Tree

ex-1

Thee

- ① check at each level
- ② so at each level char should present.
- ③ And last/node eow = true



word exists

curr = root

for (level = 0 to key)

{ idx = key.charAt(level) - 'a'; }

if (curr.children[idx] == null) { return false; }

curr = curr.children[idx]; }

return curr.eow == true;

This should also be the end of word.

Word Break Problem

T.C
⇒ O(n).

Google

key = "like samsung"

words[] = { like, Sam, Samsung, mobile, ice }

Q. Can we break the key such that each part is present
in the dictionary / words[] as a word?

→ Like samsung → return true,
✓ ✓ ✓

Approach. - ① create a tree from string.

public static boolean wordBreak(String key) {

 // base case

 if (key.length() == 0) return true;

 for (i=1 to i <= key.length())

 As the substring function
 is inclusive

 if (search(key.substring(0, i)) && wordBreak(key.substring(i)))
 return true;

 }

 return false

Prefix Problem

30-40 min

Microsoft, Google

Find shortest unique prefix for every word in a given list.

Assume no word is prefix of another. → ex- apple

apple

X

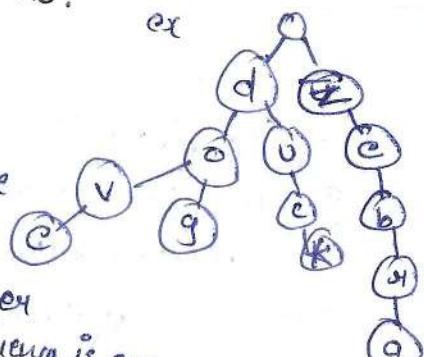
no such
case

→ Uska short name banao (nickname)

ekki sabka (nickname) unique ho.

Approach = ① tree create.

ex



→ ex- dov,
dog,
du,
z.

② also add frequency feature in tree.

③ if the letter/char repeated increase
the frequency.

④ the traverses each word letter by letter

till you find a letter whose frequency is one.

⑤ and then you will find unique prefix for every word

RE) check root to freq = -1) T.C = O(L)

Q. Starts With Problem

→ creates a function boolean which find the prefix is present or not

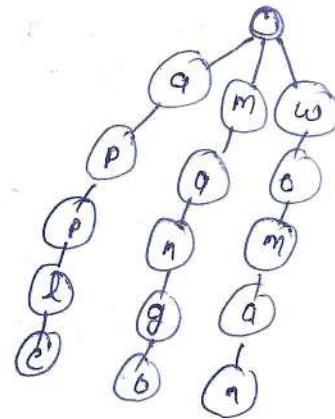
ex- words [] = { "apple", "app", "mango", "man", "woman" }
Prefix "app" output true, as in apple, app
Prefix "moon" output false.

→ Approach → ① create tree.

② check one by one character
in tree.

in
de/ear startwith(string prefix){

Node curr = root;
for (int i=0 to prefix.length();)
idx = prefix.charAt(i) - 'a';
if (curr.children[idx] == null)
return false;
else (curr = curr.children[idx])
return true;



Q. Count Unique Substrings

(Microsoft Google)

↳ a string is given count unique number of substrings.

ex- str = "ababa" → ① ② ③ ④ ⑤
ans = 10

① ab aba abab ababa
⑥ ④ ⑧ ⑨ " " ← empty string.

If you want all unique strings

→ unique substring = all prefix of all suffix
unique

~~Time~~ ~~Space~~
• Trie stores unique prefix \rightarrow count of Node of trie

Approach

① find all suffix of string

② Create trie and insert all suffix

③ Count nodes of trie

$$\text{root} \Rightarrow \begin{matrix} \text{Unique} \\ \text{prefix} \end{matrix} = \begin{matrix} \text{Unique} \\ \text{Suff-String} \end{matrix}$$

Suffix

\rightarrow for (int i=0 to n) {

str = str.substring(i)

 insert trie (str)

}

\rightarrow Count nodes as current in BST

int CountNodes (root) {

 if root==null;

 Count=0; return 0;

 for (int i=0 to 26) {

 As we have 26 letters

 if (root.leftchild[i]==null) {

 so we will check if further node is present or not each time.

 Count = CountNodes (root.leftchild[i]);

 return Count+1;

Longest Word with all Prefixes

\Rightarrow ex- words = ["a", "banana", "app", "appl", "ap", "apply", "apple"]

ans = "apple"

\rightarrow because a ap app, appl apple

apply

is also an answer

but apple < apply
lexicographically

[If any question has ~~goes~~ the solution that goes with prefix remember
Trie]

- So we can solve this question using Trie
- we want a longest word from a Trie who's all eof at each letter is true
 - this means all prefixes of word is present in the dictionary

Approach →

~~Approach~~ count the no. of root → children

↳ ① ! null

② eow = true.

Pseudo Code

→ static String ans = "" // final ans.

void longest word (root, temp) {
 if (root == null) ← // base case
 return;

for (int i=0; i<26)

if (root.child[i] != null & child[i].eow == temp)

temp + add char (root)

if (temp len > ans len)

ans = temp

largestword (root, child[i], temp)

temp → delete last char of temp.

Here we did not want to sort the answer as

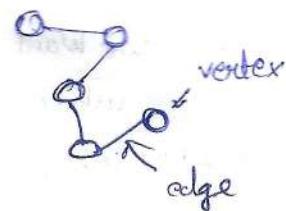
in Trie the letters are stored from $a \rightarrow z$

lexicographical order.

∴ it is already

Graph

network nodes

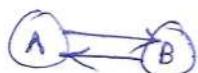


Edges

① Uni. Direction

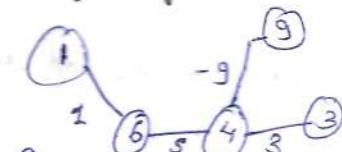


② Bidirectional / Un-Directed

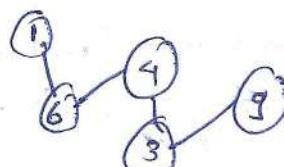


sometime
• weight of edge

ex -



i) weighted undirected



i) unweighted undirected.

Storing a Graph

① Adjacency List → List of List

② Adjacency Matrix

③ Edge List

④ 2D Matrix (Implicit Blraph)

→ Adjacency List → List of List

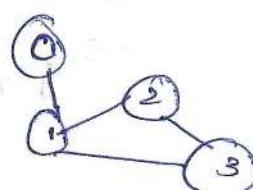
→ example -

→ $0 \rightarrow (1)$ list[0]

$1 \rightarrow (0, 2, 3)$

$2 \rightarrow (1, 3)$

$3 \rightarrow (1, 2)$



→ Data structures we can use

1) ArrayList<ArrayList>

2) Array <ArrayList> ✓ we will go with this

3) hash map <int lists>

List of Lists

- ① takes no extra space
- ② optimized

Array of ArrayLists <Edge>

src distination wt
(weighted) standard = 1

Type 2

Adjacency Matrix.

space $\rightarrow O(V^2)$

$V = \text{vertices}$.

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

(i, j)

$i \rightarrow j$ if vertex $\rightarrow 1$

$i \rightarrow j$ if not $\rightarrow 0$

Disadvantages

\hookrightarrow In time complexity
can go till so big.

Type 3

Edge List \rightarrow It stores the information of all the edges

$\rightarrow \{\{2, 1\}, \{1, 2\}, \{1, 3\}, \{2, 3\}\}$

\rightarrow we use this for some question

if with edges $\rightarrow \{\{S, D, w\}\}$

source desti weight

Type 4

Graph

Implicit Graph

\hookrightarrow In some questions if it is given that you
need to go from one point to another
in 2D Matrix then it is the use of Implicit
graph.

Application of Graph

i. Research (map neurons)

ii. Machine learning (Computational graph)

iii. Dependency Graph

ex - servers,

iv. Computer

vision \rightarrow Image segmentation.

v. Graph Databases ex - nebula.

1) MAPS (sorted network).

2) Social Networks.

3) Delivery network.

4) Physics and Chemistry (molecules).

5) Routing of algorithm.

Creating a graph (Adjacency list)

ArrayList<Edge> graph

Edge = (src, dest, weight)

code

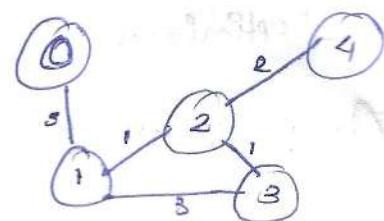
```
static class Edge {
```

```
    int src;  
    int dest;  
    int wt;
```

```
    public Edge(int s, int d, int w) {
```

```
        this.src = s;  
        this.dest = d;  
        this.wt = w;
```

ex:



0 → {4, 5}

1 → {1, 5} {1, 2, 1} {1, 3, 3}

2 → {2, 1, 1} {2, 3, 1} {2, 4, 2}

3 → {3, 1, 3} {3, 2, 1}

4 → {4, 2, 2}

3

```
public static void main() {
```

```
    int v = 5;
```

no. of nodes

ArrayList<Edge> [] graph = new ArrayList[v]; // null → empty array list

```
for (int i=0; i<v; i++) {  
    graph[i] = new ArrayList<>();
```

// 0-vertex

```
graph[0].add(new Edge(0, 1, 5));
```

// 1-vertex

```
graph[1].add(new Edge(1, 0, 5));  
graph[1].add(new Edge(1, 2, 1));  
graph[1].add(new Edge(1, 3, 3));
```

Same of 2, 3, 4 vertex.

now printing 12's neighbours

```
> for (int i=0; i<graph[2].size(); i++) {  
    |  
    |   Edge e = graph[2].get(i);  
    |  
    |   System.out.print(e.dest);  
|  
|}
```

Graph Traversals

can be asked directly
as a question

- ① Breadth First Search (BFS)
- ② Depth First Search (DFS)

Breadth F S → go to immediate neighbour first,

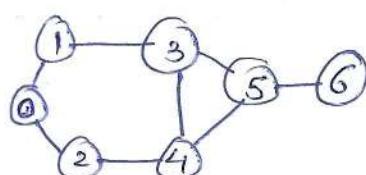
↳ we need to have a boolean array of visited
↳ because we have cycle.
↳ so that each node is visited once.

- ① get an any node
- ② add its neighbours in Queue.
- ③ print that node and mark visited.
- ④ ~~get~~ pop element from Queue

if visited ✓
if it is
not visited
then step
a) → remove from Queue
b) → ④. Pop another element
a) → ②③④

- ⑤^{while} the queue is empty.

ex-



← this code is written in C++.

T.C ⇒ $O(V+E)$ → because we travel all vertices and edges.

Output :

1, 2, 3, 4, 5, 6

But if we used matrix here then the T.C will be $O(V^2)$ so we used adjacency list.

DFS → keep going to first neighbour.

(depth first search)

→ can be done using stack or recursion
(implicit stack)

Approach

① ~~dfs (graph, curr)~~ (Recursion)
curr → visit

then

② for (int i=0 to k)

(!vis[adj])

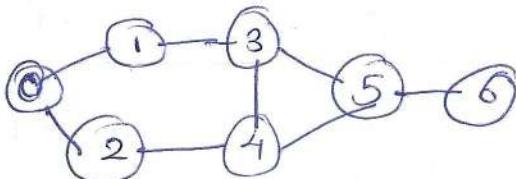
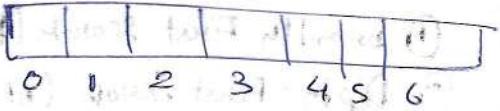
dfs (neighbours)

(i)

}

vis[curr] = true

& array of visited



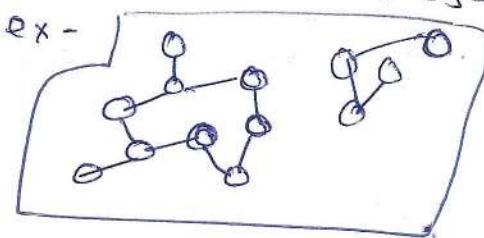
$$T.C = O(V+E)$$

(vertices + edges)

Output : 0, 1, 3, 4, 2, 5, 6

Q. Has Path ? $\begin{cases} \text{src} = 0 \\ \text{dest} = 5 \end{cases}$] Is there any path from zero to 5?

→ there can also be disjoint graphs



this is a one graph.

$$T.C = O(V+E)$$

Approach

haspath(C, src, dest, vis) {

① src=dest return true,

② if (!vis[neighbour]) && haspath(C, neighbour, dest)

return true;

{ return false;

@signature