

Hard Work beats Talent,

When Talent doesn't Work Hard!

RECURSION

INTRODUCTION:-

1) Make Input Smaller! But Why?

2) Recursion - Decision Space

3) Recursive Tree - Soul of Recursion] Imp.

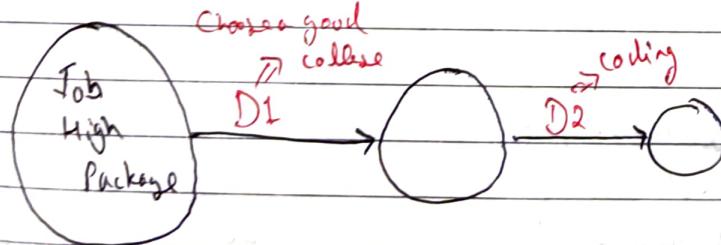
4) 2 steps to solve any recursive Problem

1) Make Input Smaller! But Why?

We don't deliberately smaller the input. The input becomes smaller automatically!

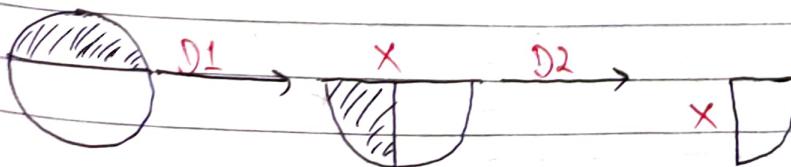
Example:-

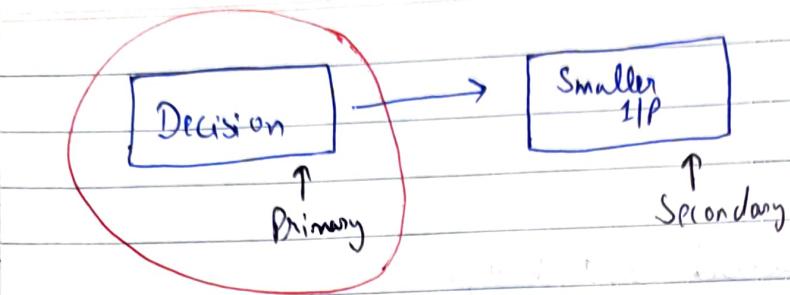
Let's understand it with an example. How the input is getting smaller.



To get a high package job. I have to take 1st decision of good college. By doing that our problem becomes small. Now our 2nd Decision is to learn coding. Our problem becomes more smaller.

Similar thing with our Input.



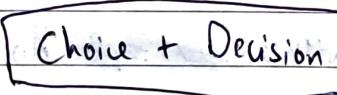


Our decision has to be a primary goal & on the basis of that Input becomes smaller automatically

2) Recursion - Decision Space

Whenever we have to think about Space. Recursion is a good choice. now, how can we identify that whether a problem statement uses recursion or not.

→ For that we can see that, the problem has given choices & some decision. So, it's using recursion

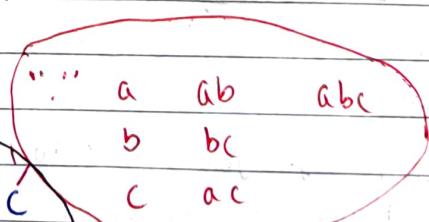


3) Recursion Tree

Let's understand with example.

→ Get subset of "abc"

So, its subset will be



In recursion we have choices

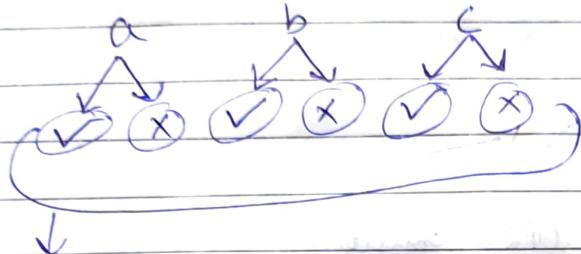
&
take decision
This is my choice.

+

These are my decisions

→ So in this what choices do I have.

I should consider or not



So on these choices whatever the decision I take will make my subset!

→ So if you see we taking choices whether to have or not. And by that choices we taking decision. And because of that our input is becoming smaller.

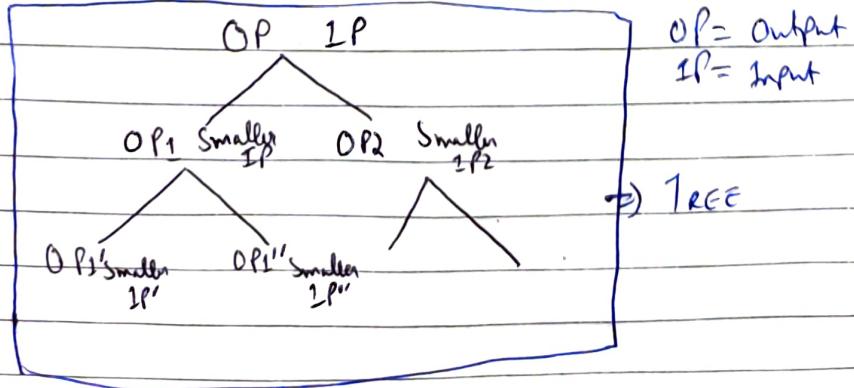
Now let's understand What is Recursive tree?

Again, let's understand with example we have to get subset of "ab"

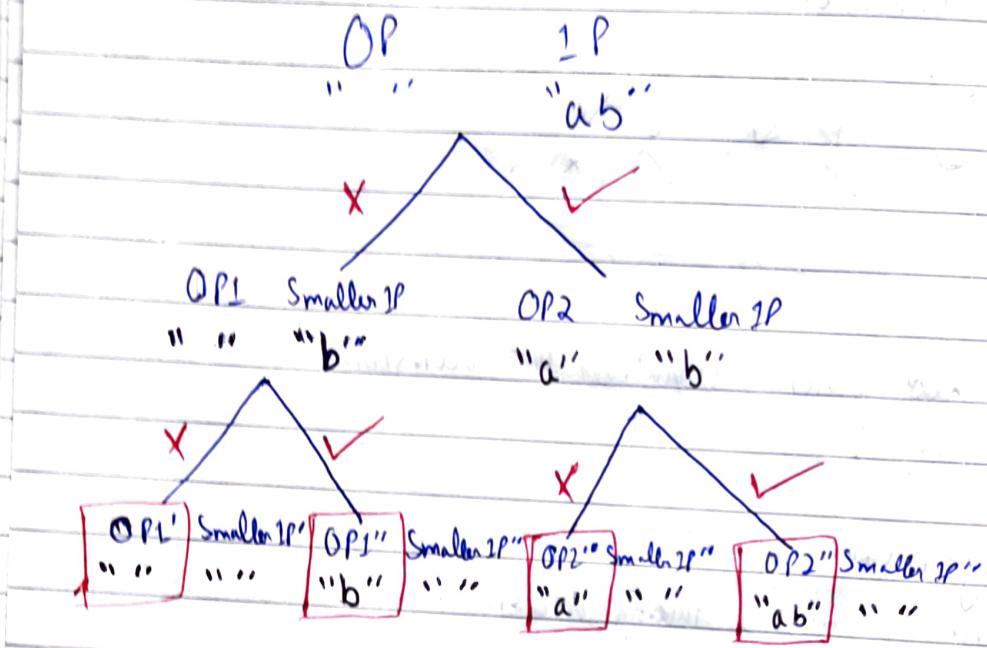
$\{\}$	$a \times b \times$
a	$a \checkmark b \times$
b	$a \times b \checkmark$
ab	$a \checkmark b \checkmark$

So instead of representing in $a \times b \times$ way. We want to represent in a good way. That way is called Recursive tree

For Recursive tree there is a method called "IP-OP Method"



Let's understand this tree with an example.



When Smaller "IP" becomes empty return the tree & gets answer

4) 2 Steps to Solve Recursive Problem

1) Design a recursive tree

2) Fuck The Problem aka write the code

RECURSION IS EVERY WHERE

We use recursion most of the places.

Data Structures:-

- ARRAY / String
- TREE
- HEAP
- GRAPH [DFS]
- STACK
- Linked List

Recursion is backbone of :- DP / Backtracking / Divide & Conquer

- 1) Print 1 to n / N to 1
 - 2) Sort an array → Sort on Stack
 - 3) Delete middle element in a stack
 - 4) Remove Duplicate from a String
 - 5) Count no. of Occurrence using Recursion
 - 6) Permutation : - Space Variation
 - Case Change
 - And many More
 - 7) Tosses Problem
 - 8) Print N binary numbers having more 1's than 0's
 - 9) Generate all Balanced Parentheses
- } Easy
- } Medium
- } Hard

Hypothesis - Induction - Base Condition

→ Print 1 to N / N to 1

Flow:- ★ Methods for a Recursive Problem

★ Why making I/P smaller is Popular/Imp.

★ Print 1 to N

1) Methods :- There are 4 approaches to solve a Recursive problem:-

→ Recursive Tree - I/P/O/P Method [Works only if you know decision]

→ B.C - Induction - Hypothesis [Making I/P smaller]

→ Choice Diagram (DP)



Recursion Problem → Hypothesis [Work like recursive tree]

Recursion for ⇒

$$\text{Solve}(n) = \frac{1}{2} \text{ ton}$$

↳ Magic

Induction

Base Conditions:- 1) Smallest Valid I/P
2) Smallest Invalid I/P

void print(int n)

{

if (n == 1) → base condition
return 1

Output → 1
2
3
4
5

print(n-1) → Hypothesis

S.O.PLn(n) → Induction

3

print(7)



print(6)



print(5)



print(4)



print(3)



print(2)



print(1)



print(0)



print(7)

we will just look at till here we have to work

Coverup:- • Easy by LBN

• Medium by Recursion tree

• Hard by Choice diagram

void print(int n)

{

if (n == 1) return;

Output → 7

S.O.PLn(n);

6
5
4
3
2
1

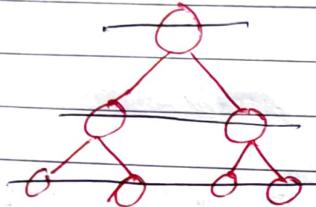
print(n-1);

3

HEIGHT OF A BINARY TREE

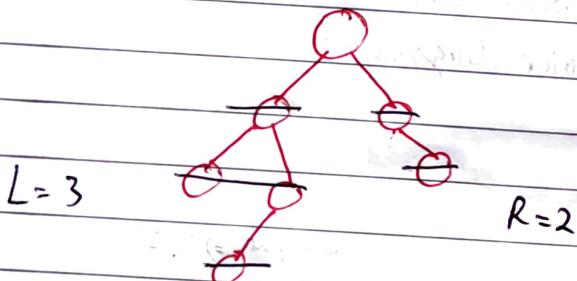
Let's understand this with an tree example.

Say, we have given something like this,



To get his height, we draw line. That is "3"

Now, by recursion how we can do this. So we know the root & its child will be find by hypothesis. So we go deep inside the left, then right & from where we get the max. deep. That will be count as height with +1 = bcz, root will count as well.



The code will be

```
int height( Node root){  
    if( root == null) return 0; //Base condition
```

```
    int left = height( root.left);  
    int right = height( root.right);
```

// hypothesis

```
    return Math.max( left, right)+1;
```

// induction

SORT AN ARRAY USING RECURSION.

Firstly your thought is, why we are doing this with recursion. As, we can do that using sorting or iterative way. I say, you will learn from this a lot. So, now without any further due let's start.

PS:- IP:- [2, 4, 3, 6, 9, 7]

OP:- [2, 3, 4, 6, 7, 9]

I want you to understand with more basic example

arr = [0, 1, 5, 2] Sort this array

our hypothesis will be make it smaller

So, Sort [0|1|5|2]

Sort (0|1|5) [2]

Now in induction

insert (0|1|5) [2])

Again in Hypothesis make it more smaller

insert (0|1), [2) → [0|1|2] [5])

Sort $\xrightarrow{\text{induction stop}}$ insert $\xrightarrow{\text{induction stop}}$ arr.array.push

Code on next Page:-

Public void List<Integer> Sort(List<Integer> arr) {
 if (arr.size() == 1) return arr; } Base condition,

Hypothesis [Integer temp = arr.get(arr.size() - 1); // [0, 1, 5, 2]
 arr.remove(arr.size() - 1); // [0, 1, 5] [2]
 arr = sort(arr); // [0, 1, 5]

Induction [return arr = insert(arr, temp); // [0, 1, 5] [2]
 3]

Public void List<Integer> insert(List<Integer> arr, int i) {
 if (arr.size() == 0 || arr[i] >= arr.get(arr.size() - 1))
 {
 arr.add(i); // let's say in last we have 6
 return arr; // [0, 1, 5] [6] => [0, 1, 5, 6]
 }
 Base condition

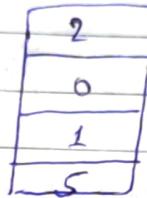
Hypothesis [int val = arr.get(arr.size() - 1); // [0, 1, 5]
 arr.remove(arr.size() - 1); // [0, 1, 2] [5]
 arr = insert(arr, i); // [0, 1, 2]

Induction [arr.add(val); // [0, 1, 2, 5]
 return arr;
 3]

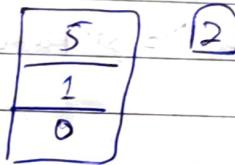
SORT A STACK

This is much similar to "sort an array".
I mean its almost to that

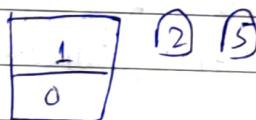
first what we do ; let's take an example:-



Take the top element & sort the stack!

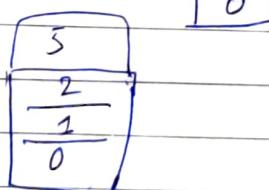


Now insert 2 into stack but before that lets make our input more smaller



Now add 2 in stack = & then finally add

5 in stack =



Let's code it!

public void Stack<Integer> sortStack(Stack<Integer> st) {

 if (st.size() == 2) return st; } Base Condition

 Integer temp = st.pop();
 sortStack(st); } Hypothesis

 return insertElementAt(st, temp); } Induction

}

public void Stack<Integer> insertElementAt(Stack<Integer> st, Integer temp) {

{

 if (st.size() == 0 || temp >= st.peek()) { } Base Condition
 st.push(temp);
 return st; }

}

 Integer val = st.pop();
 insertElementAt(st, temp); } Hypothesis

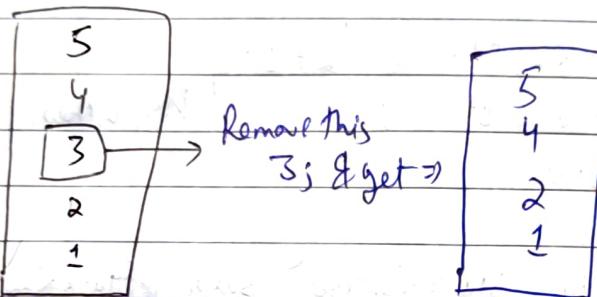
 st.push(val); } Induction
 return st;

3

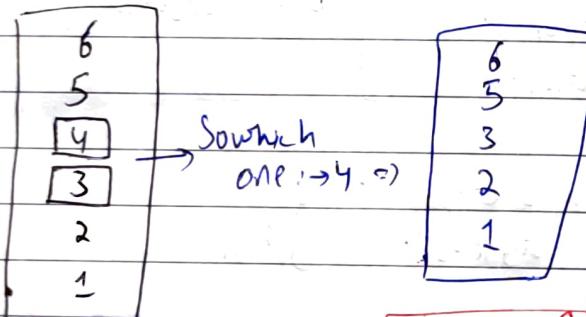
DELETE MIDDLE ELEMENT OF A STACK

So, in this we have to delete the middle element from the stack.

If Stack is odd, middle element will be



If Stack is even, middle element will be

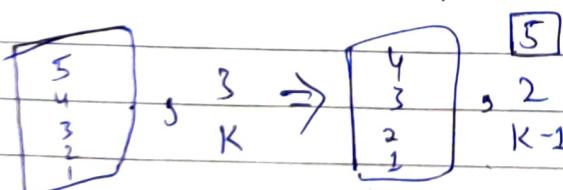


Now we have formula;

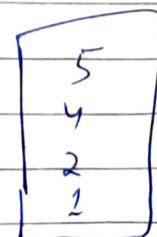
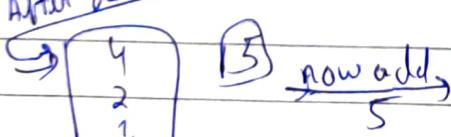
$$\text{Stack} = \frac{\text{Mid}}{2} + 1$$

So, how we do this we have choice ~~means~~ [1 BN], if 1 can take decision then [Tree].

But I have ~~can't~~ smaller input



After deleting 3



Let's Code it;

public static Stack<Integer> StmidDel (Stack<Integer> st) {

} if (st.size() == 0) return st; } Base Condition

int k = st.size() / 2 + 1;

Solve (st, k);

return;

} Hypothesis

3

public static Stack<Integer> Solve (Stack<Integer> st, int k) {

} if (k == 1) {

st.pop();

return;

} Base Condition

3

int temp = st.pop();

Solve (st, k - 1);

} Hypothesis

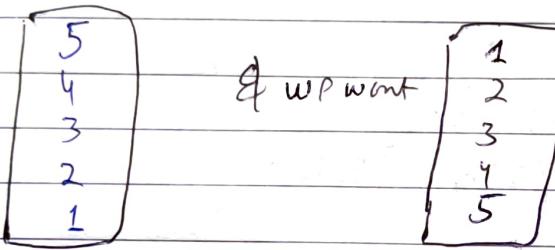
return st.push(temp); } Induction

3

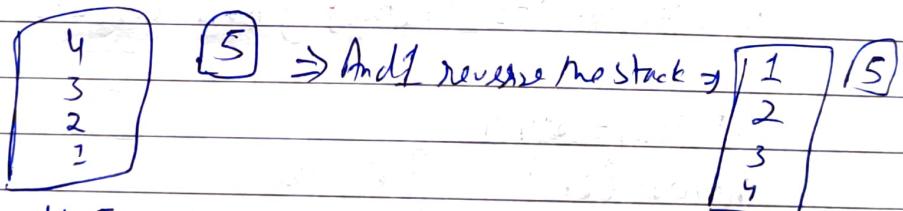
REVERSE A STACK

So, we have given a stack & without using extra space we have to reverse the stack.

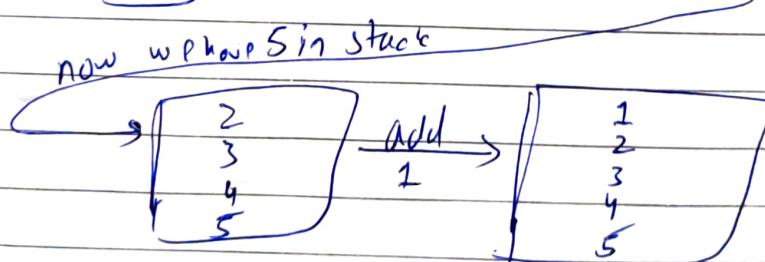
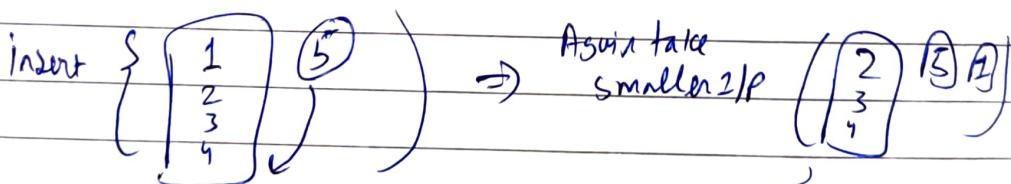
So, let's say we have something like this:-



first let me take a smaller Input That's my hypothesis.



Now to add 5 in stack. I have to empty my stack first, then add 5 & fill the stack after that.



Let's Code it

public void insert (Stack<Integer> st, int element) {

if (st.size() == 0) {
 st.push(element);
 return;
}

int temp = st.pop();
insert(st, ~~temp~~^{element});] Hypothesis
return st.push(temp);] Induction

}

Public void reverse (Stack<Integer> st) {

if (st.size() == 1) return;] Base Condition

int temp = st.pop();
reverse(st);] Hypothesis
insert(st, temp);] Induction
return;

}

K^m Symbol In Grammar

So, what it saying we have given two integer n & k . We have to return K^m (1-indexed) symbol in the n th row of table n rows.

Example:- ($n=4, k=3$)

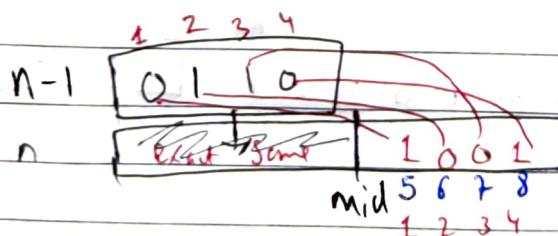
	k=1							
$n=1$	0							
$n=2$	0	1						
$n=3$	0	1	1	0				
$n=4$	0	1	1	0	1	0	0	1
		1	2	3				
					↓			
						OTP		

Now, this question is all about to check your observation skill!

So if we carefully look at, & we have to find induction:-

0
0 1
 $n-1$ 0 1 1 0
 n 0 1 1 0 1 0 0 1

If we see then we find out that (1) 1st half is equals to $(n-1)$. And the remaining half is just the complement of (1).



So for the complement part, how we can do is if we check the indexing of $n-1 = k - \text{mid}$.

Let's get its induction part \Rightarrow

if ($k \leq \text{mid}$) { } // get values of k before mid

Hypothesis $\text{solve}(n, k) \rightarrow \text{solve}(n-1, k)$

else { } // get value of k after mid

$\text{solve}(n, k) \rightarrow \text{solve}(n-1, k - \text{mid})$; // induction

Base condition

if ($n = 2 \ \& \ k = 1$) return 0;

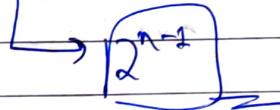
Grammar to get mid

$1 = 2^0 \leftarrow n=1 \quad 0$

$2 = 2^1 \leftarrow n=2 \quad 0 \ 1$

$4 = 2^2 \leftarrow n=3 \quad 0 \ 1 \ 1 \ 0$

$8 = [2^3] \leftarrow n=4 \quad 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 1$



int mid = (int) Math.floor (2, n-1) / 2;

Main logic

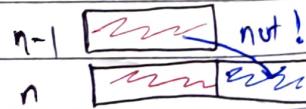
if ($k \leq \text{mid}$)

return & solve(n-1, k)

3

else

return 1 - solve(n-1, k-1)



OBSERVATION + 1BM to solve this problem
70% 30%

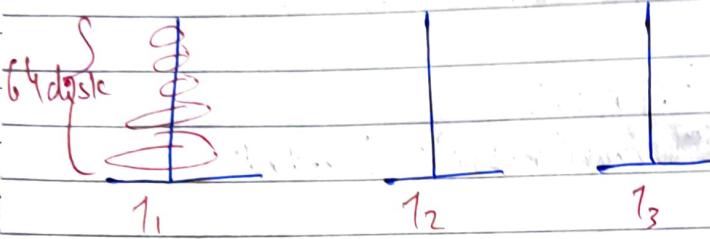
Let's Code it:-

```
public int kmbinamman(int n, int k){  
    if(n == 1 && k == 1) return 0; // Base Condition  
    int mid = (int) Math.pow(2, n-1)/2;  
    if(k <= mid) // Hypothesis  
        return kmbinamman(n-1, k); // Induction  
    else  
        return 1 - kmbinamman(n-1, k-mid mid);  
}
```

TOWER OF HANOI

Before solving the problem, I have one interesting story to share with you.

So, its original name is Tower of Brahma



Put on T_3 all 64 disk :-

But condition :- 1 disk at a time

• And has to be in large to small order

If, 1 disk taking 1 sec then 64 will

$$\left(2^{30} = 1B\right) + \left(2^{30}\right) + \left(2^4 = 16\right) \Rightarrow [16BB \text{ second}] \rightarrow 585 \text{ billion years}$$

age of universe

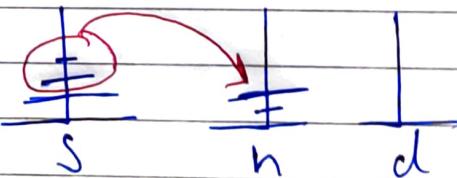
Problem :- Given 3 disk & 3 tower. Return the steps of putting from T_1 to T_3

Hypothesis

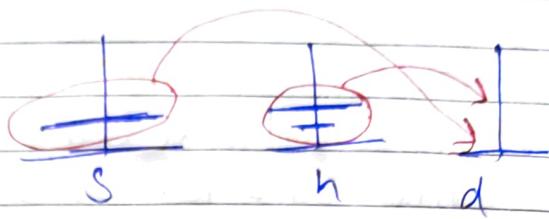
Solve($n, S \rightarrow d, h$)

Smaller

Solve($n, S \rightarrow h, d$)

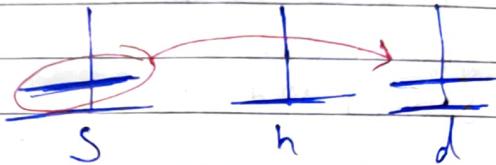


Induction



Base Condition

If there only 1 disk then
simply put on destination



```
int main() {
    input(n);
    int s=1, h=2, d=3;
    solve(n, s, d, h);
}
```

Pseudo Code

```
void solve (n, s, d, h) {
    if (n == 1) {
        print(moving plate n from s to d);
        return;
    }
    solve(n-1, s, h, d);
    print(moving plate n from s to d);
    solve(n-1, h, d, s);
}
```

If you interviewer ask you to print no. of steps:-

Then in main func. create

```
int count = 0;
& in void func. increment count
    count++;
```

By doing that whenever func. call takes place count will increment.

Optional

Let's Code it:-

```
public static void main (String [] args) {  
    Scanner scn = new Scanner (System.in);  
    int n = scn.nextInt();  
    int s=1, d=2, h=3;  
    solve (n, s, d, h);  
}
```

} Main for

3
public static void solve (int n, int s, int d, int h) {

if ($n = 1$) {

Base condition

S. O. P. ln (Moving plate from s to d)

return;

3

solve ($n-1$, s, h);] Hypothesis

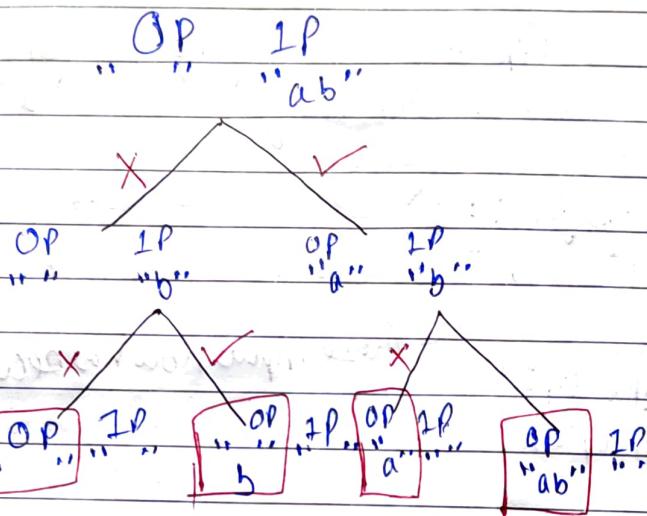
S. O. P. ln (Moving plate n from s to d);

solve ($n-1$, h, d, s);] ~~Hypothesis~~ Induction

3

PRINT SUBSETS

Let's first recap how we build a recursion tree of subsets.



First let's create main fnc.

```
int main() {
```

str = ip;

 String op = ""

 solve(ip, op);

}

PSEUDO CODE

```
void solve(String ip, String op){
```

 if(ip.length() == 0)

 {

 print(op + "");

 return;

}

 Base Condition

So what this B.C is saying that if input length becomes empty return output!

If you look at above diag we are only returning when input becomes empty.

String op1 = op // will be same as op.

String op2 = op // But for op2.

// We consider 0^m index value.

OP2.push_back(ip[0])

// adding 0^m value to OP2 string

ip.erase(ip.begin() + 0)

// we erase [remove] 0^m value index for next consideration

Solve(ip, op1)

Solve(ip, op2)

These inputs now has become smaller

Let's Code it:-

```
public static void main(String[] args) {
```

```
    Scanner scn = new Scanner(System.in);
```

```
    String ip = scn.nextLine();
```

```
    String op = " ";
```

```
    solve(ip, op);
```

```
}
```

```
public static void solve(String ip, String op) {
```

```
    if (ip.length() == 0) {
```

```
        System.out.println(op + " ");
```

```
        return;
```

```
}
```

```
    String op1 = op;
```

```
    String op2 = op;
```

```
    op2 += ip.charAt(0);
```

```
    ip = ip.substring(1);
```

```
    solve(ip, op1);
```

```
    solve(ip, op2);
```

```
}
```