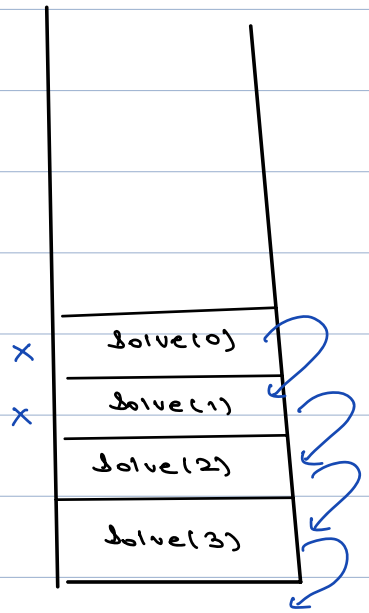
Quiz 1 :-

N = 3

```
1  void solve(int N){
2      if(N == 0)
3          return;
4      solve(N-1);
5      print(N);
6  }
```

1 2 3

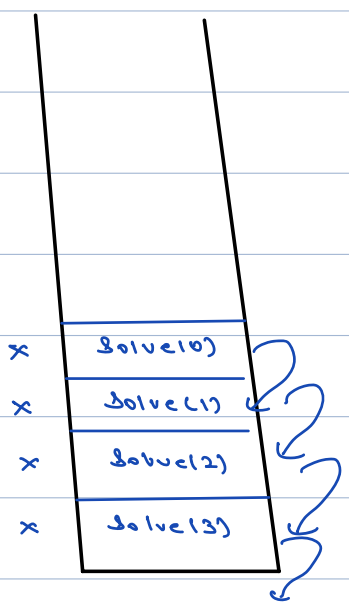


Quiz 2 :-

N = 3

```
1  void solve(int N){
2      if(N == 0)
3          return;
4      print(N);
5      solve(N-1);
6  }
```

3 2 1

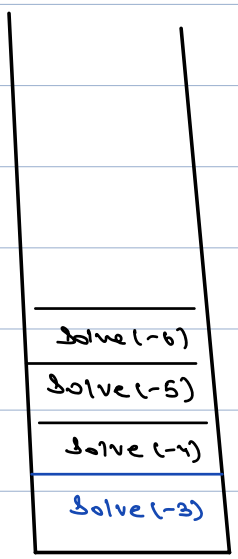


Quiz 9

N = -3

```
void solve(int N){
1   if(N == 0)
2       return;
3   print(N);
4   solve(N-1);
}
```

-3 -4 -5



TLE.

Stack Overflow Error

Towers of Hanoi

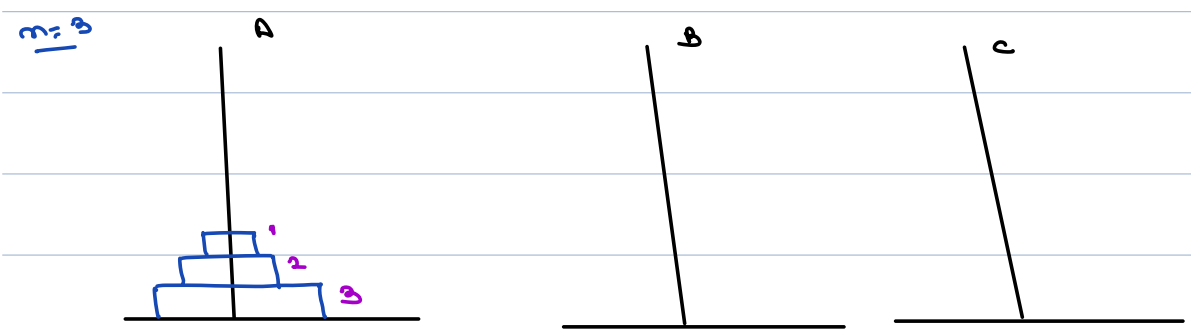
There are n disks placed on tower A of different sizes.

Goal

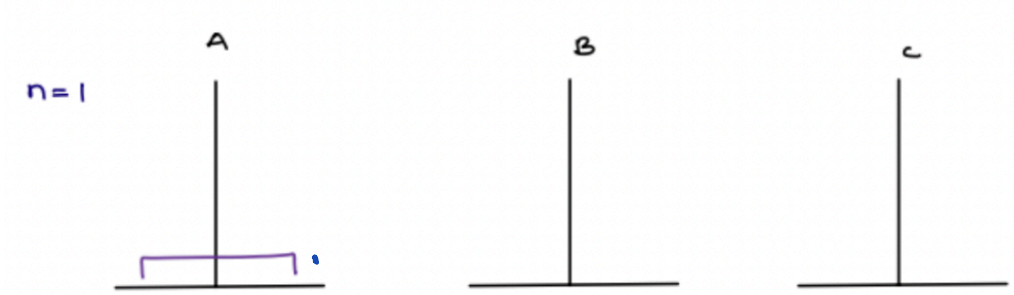
Move all disks from tower A to C using tower B if needed.

Constraint

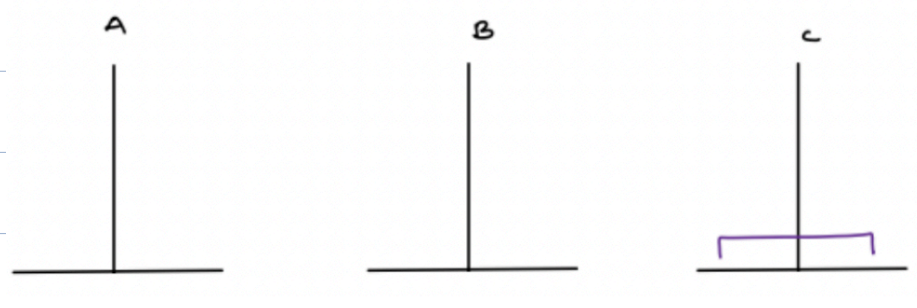
- Only 1 disk can be moved at a time.
- Larger disk can not be placed on a small disk at any step.



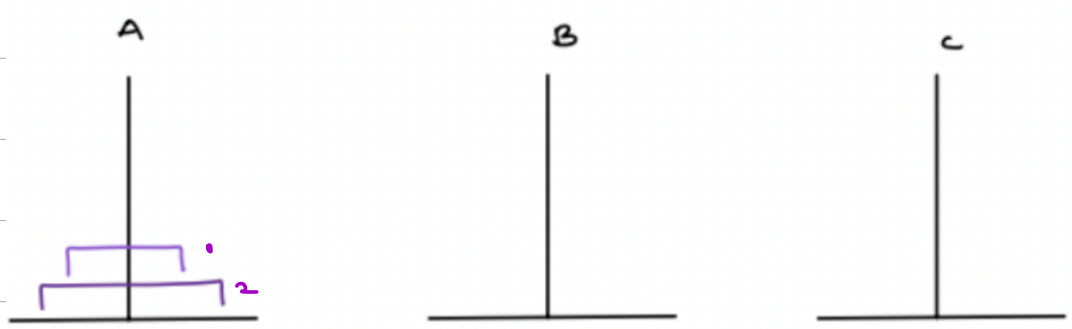
n = 1

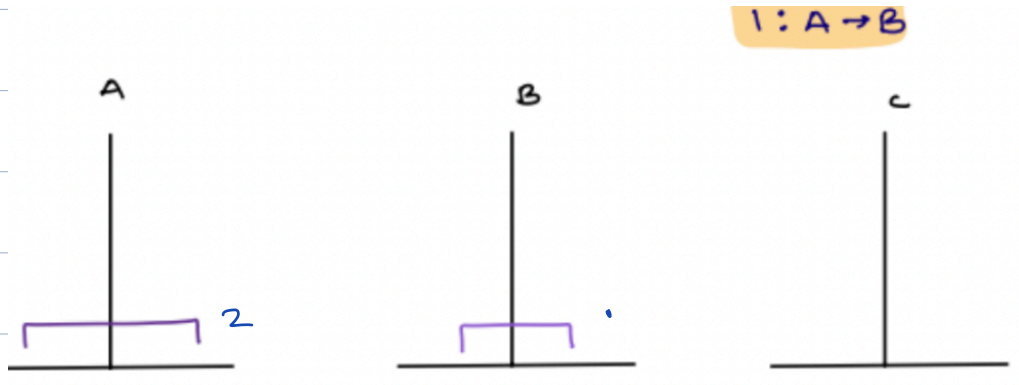


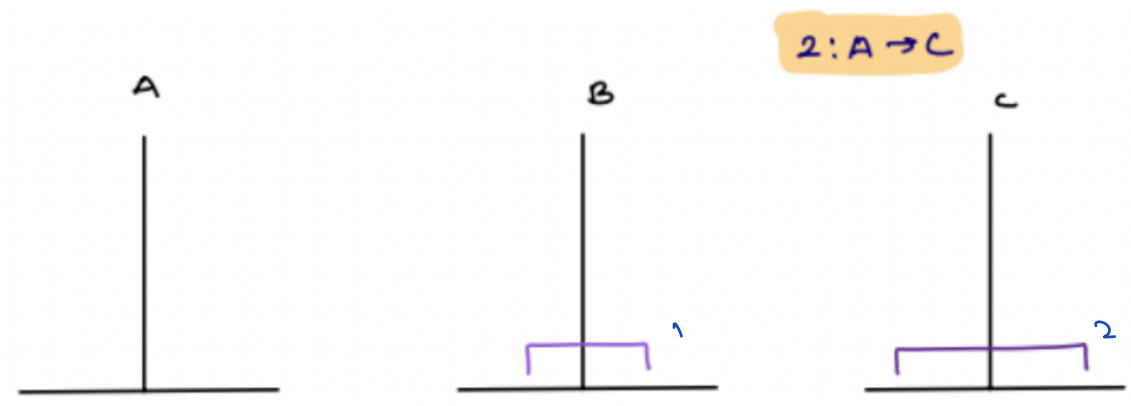
∴ A → C

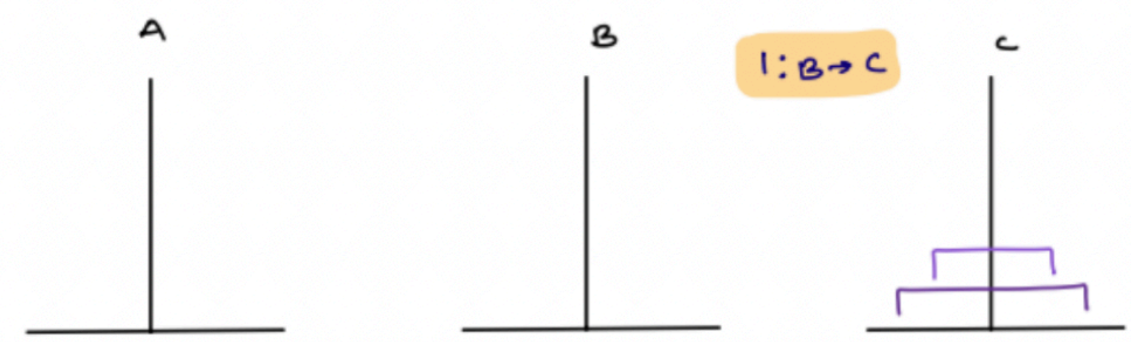


n = 2

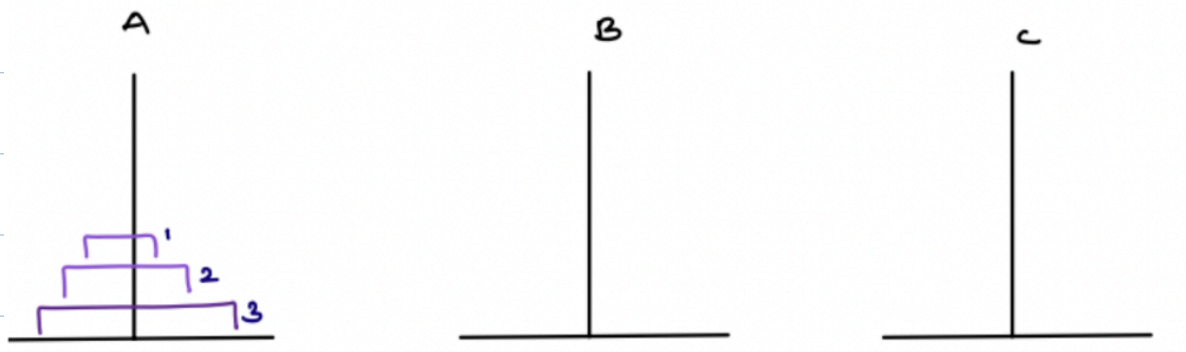



1 : A → B
A
B
C
2
1


2 : A → C
A
B
C
1
2


A
B
1 : B → C
C

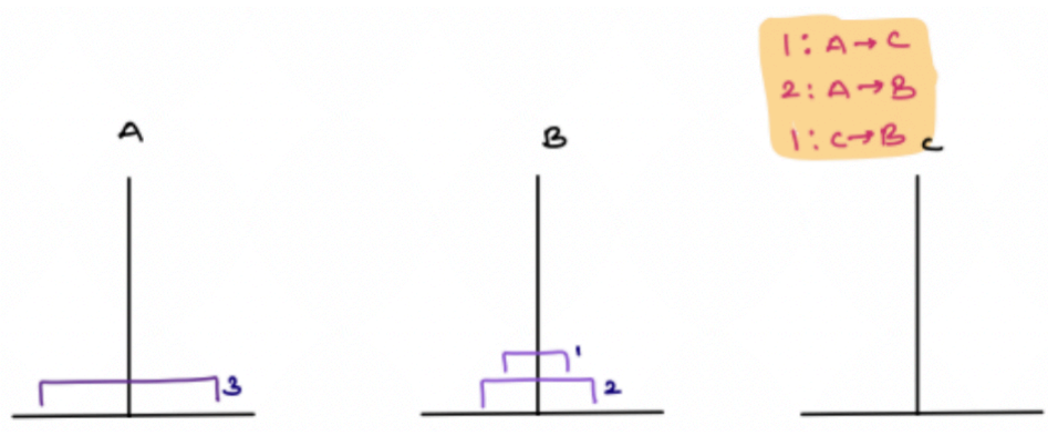
N : 3

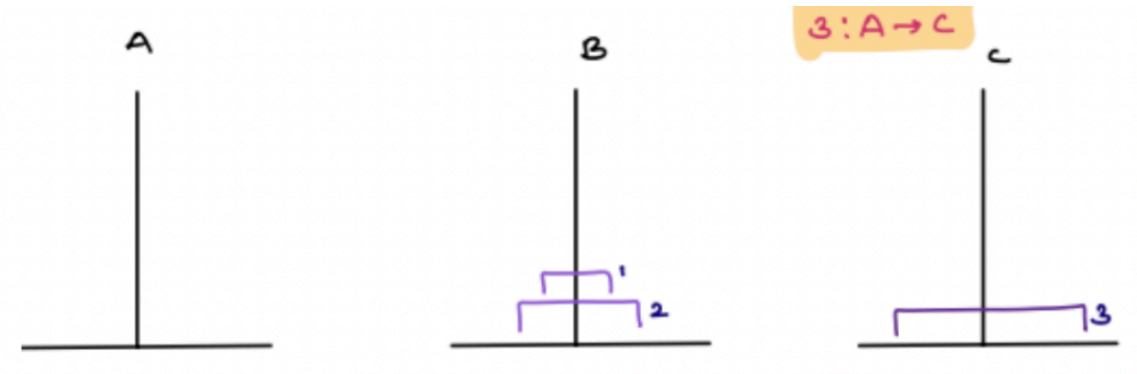


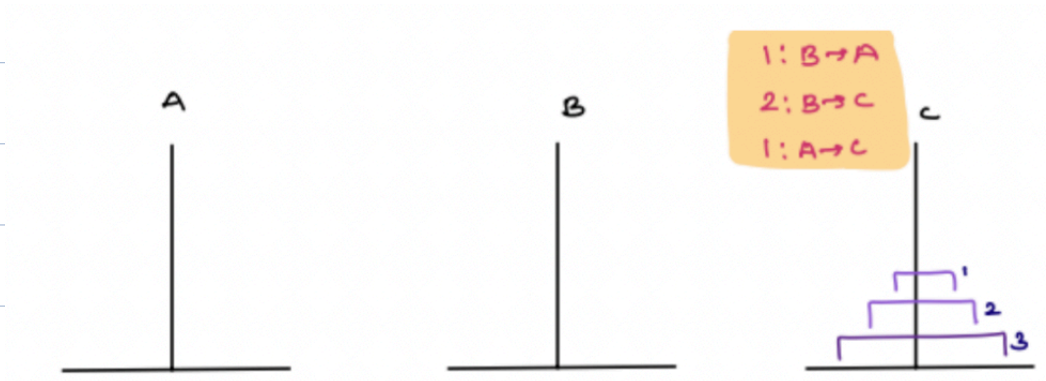
current problem . 1) 2 disks A → B

2) 3 , A → C

3) 2 disks B → C







Output:

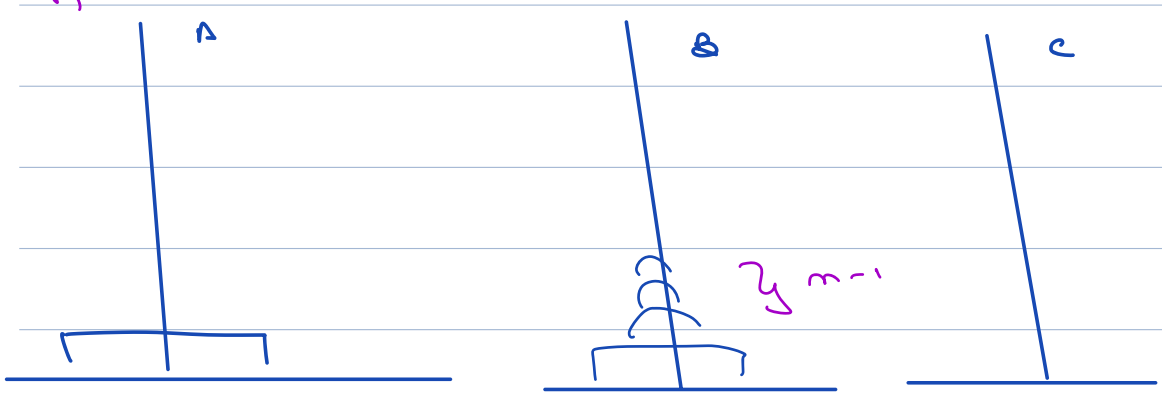
1: A -> C
2: A -> B
1: C -> B
3: A -> C
1: B -> A
2: B -> C
1: A -> C

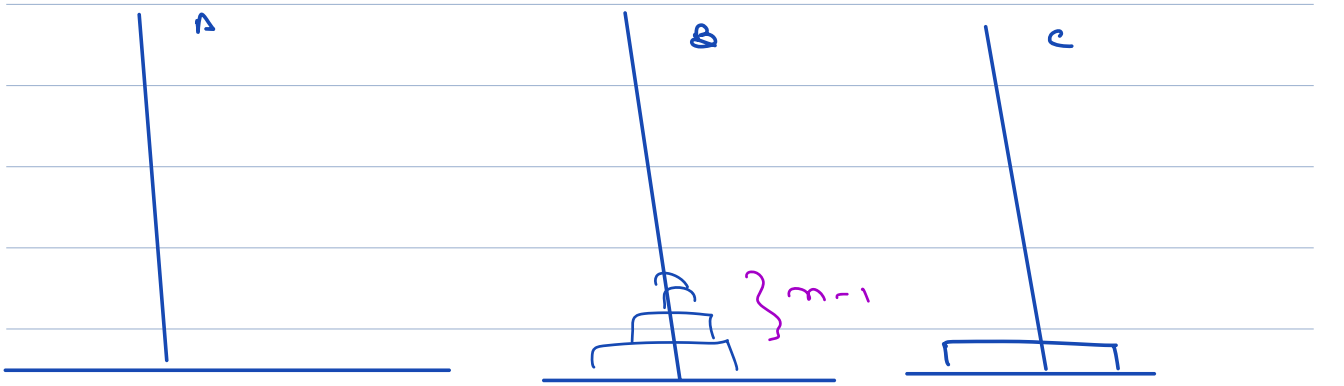
1) 3 disks from A - B
2) 4th disk from A - C
3) 3 disks from B - C.

n = 4

e.g. n disk

1)




A
B
C
n-1


A
B
C
n

Picking helper drop

```
void TOH ( int n, char s, char h, char d) {
    if (n == 0) return;
    TOH ( n-1, s, d, h)
    Print ( n: s → d );
    TOH( n-1, h, s, d );
}
```

3

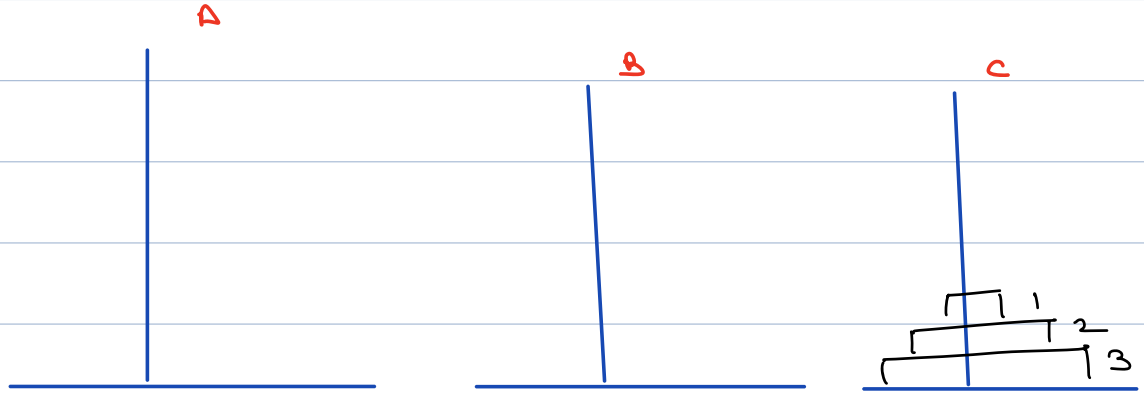
Recursive Problem :-

1) High level understanding. ✓

2) Low level Understanding ✓

TOH (3, A, B, C)

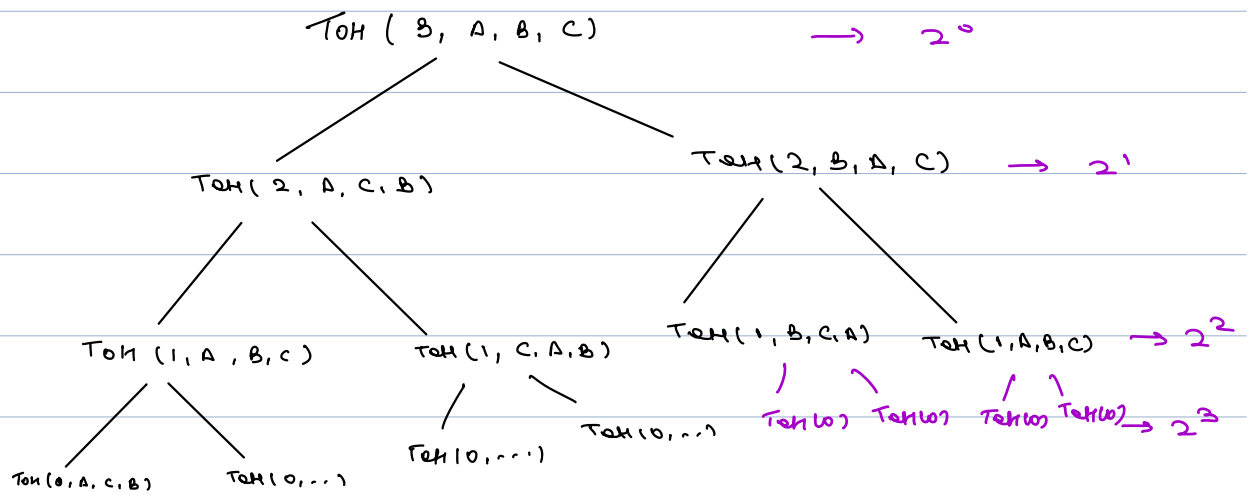
Move 1 A → C
Move 2 A → B
Move 1 C → B
Move 3 A → C
Move 1 B → A
Move 2 B → C
Move 1 A → C



T(n)

picking helper drop

```
void TOH ( int n, char S, char H, char D) {
    1  if (n == 0) Statement 3
    2  TOH ( n-1, S, D, H)   → T(n-1)
    3  Print ( n: S → D );
    4  TOH( n-1, H, S, D );   → T(n-1)
}
3
```



$n = 3, \quad 2^0 + 2^1 + 2^2 + 2^3$

input 4,

$2^0 + 2^1 + 2^2 + 2^3 + 2^4,$

input n, → g.p.

$2^0 + 2^1 + 2^2 + 2^3 + \dots 2^n \Rightarrow 2^{n+1}$

$O(2^n)$.

S.C → $O(n)$.

$T(n) = 2T(n-1) + 1$

$T(n) = 2T(n-1)+1 \Rightarrow 2^1 T(n-1) + 2^1 - 1$

$T(n-1) = 2T(n-2)+1$

$T(n) = 4T(n-2) + 3 \Rightarrow 2^2 T(n-2) + 2^2 - 1$

$T(n-2) = 2T(n-3)+1$

$T(n) = 8T(n-3) + 7 \Rightarrow 2^3 T(n-3) + 2^3 - 1$

$\vdots$

// generalized expression :-

$T(n) = 2^k T(n-k) + 2^k - 1$

$T(0) = 1$

when $k = n$,

$T(n) = 2^n T(0) + 2^n - 1$

$T(n) = 2^n + 2^n - 1$

$T(n) = 2 * 2^n - 1$

$T(n) = 2^{n+1} - 1$

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

T.C $\rightarrow O(n)$

$2^{n+1} \rightarrow 2 \cdot 2^n + 1$

Ques: Print all valid parenthesis of len 2N, for given N. '(' ')'

e.g.:

N = 1, → ()

N = 2, → ()(), (())

N = 3, ((())), ()()(), ~~)))(((~~

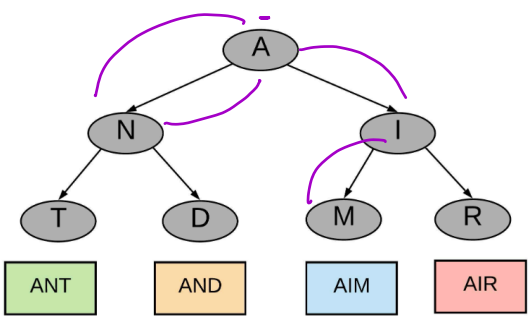
(())(), ()(()), (()())

N = 4, → ()()()(), (())(()) ...

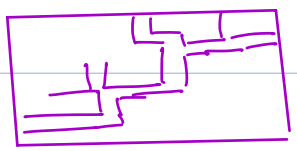
String of 2N, (), check their validity.

N = 3, ()) ...

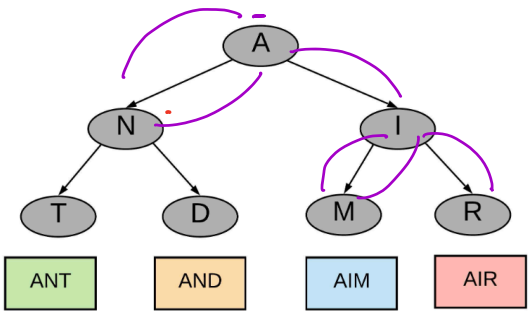
Backtracking → Trying all possible options using recursion.

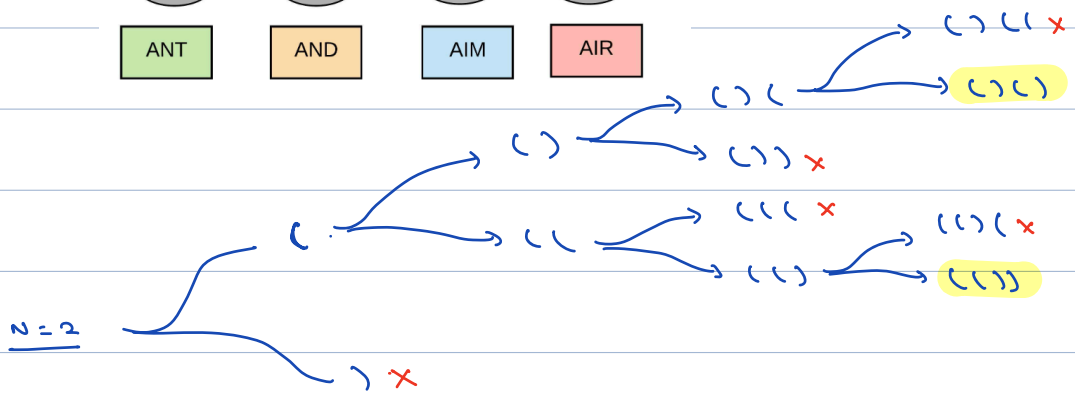


AIM



AIR





open_br, closing_br.

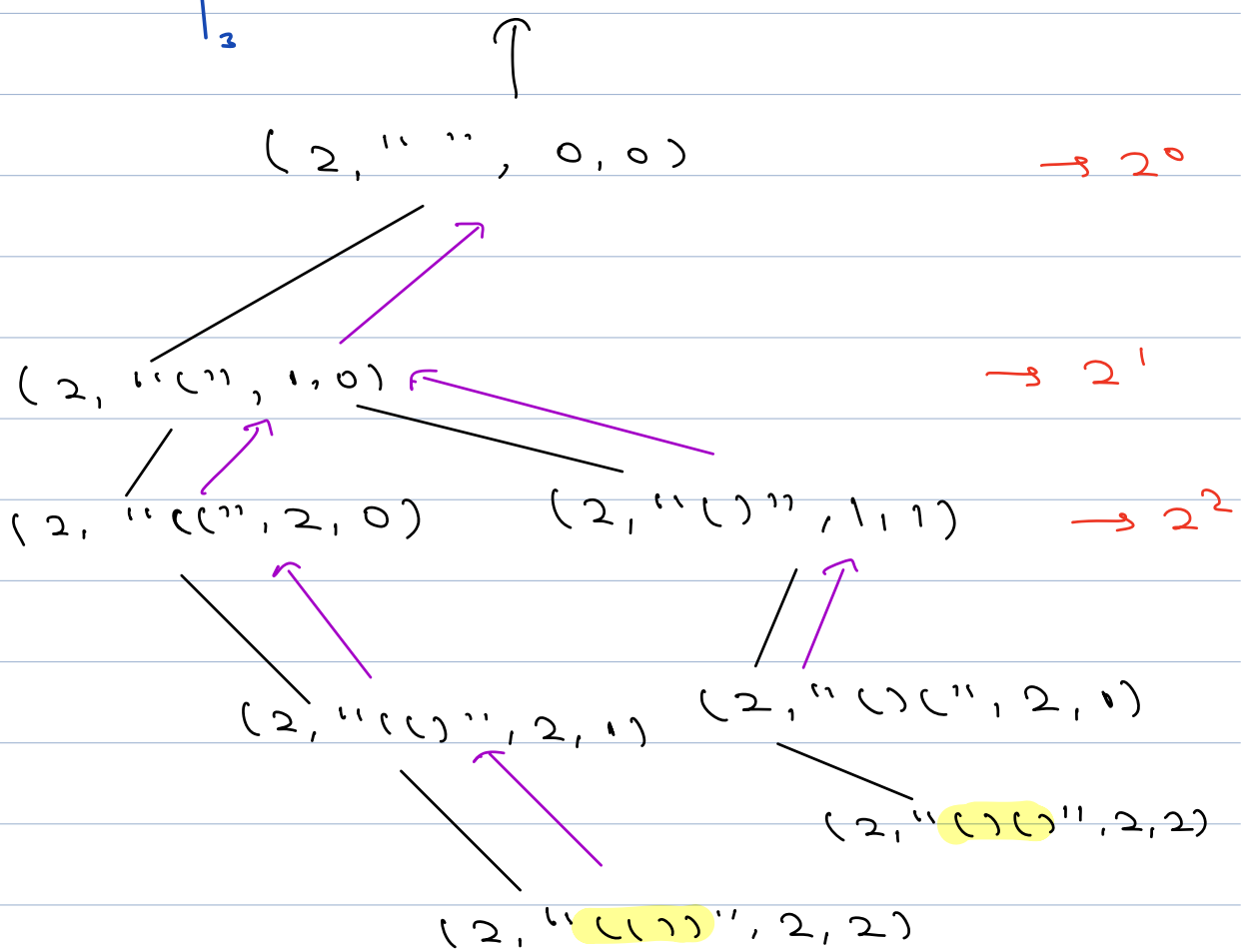
closing_br ≯ open_br

① open_bracket < n → '('

② open_brket > closingbrket → ')'

n = 2

```
void solve (n, str, open_br, close_br) {
1   if (str.len == 2*n) { print(str); return; }
2   if (open_br < n ) {
3       solve (n, str + "(", open_br+1, close_br);
    }
4   if (close_br < open_br) {
5       solve (n, str + ")", open_br, close_br+1);
    }
}
```



T.C → $O(2^n)$

S.C → O(n)