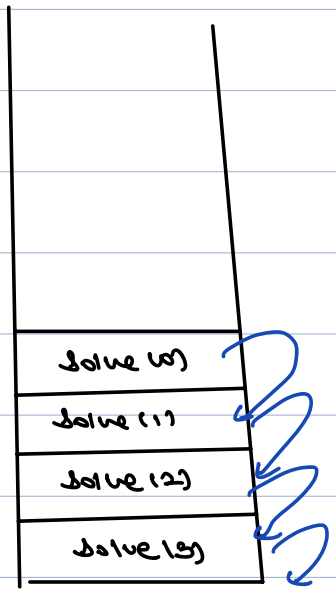
Quiz 1 :-

N = 3.

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

→ Postorder

1 2 3



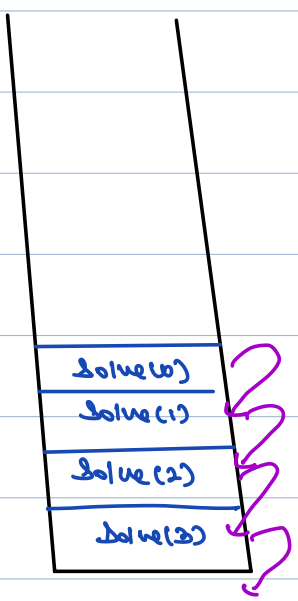
Quiz 2 :-

Solve (3)

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

← Pre order

3 2 1

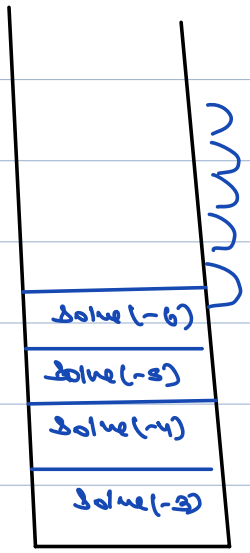


Quiz 2

N = -3

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

-3 -4 -5



TLE → Time Limit Exceeded.

Stack Overflow Error ✓

← Towers of Hanoi →

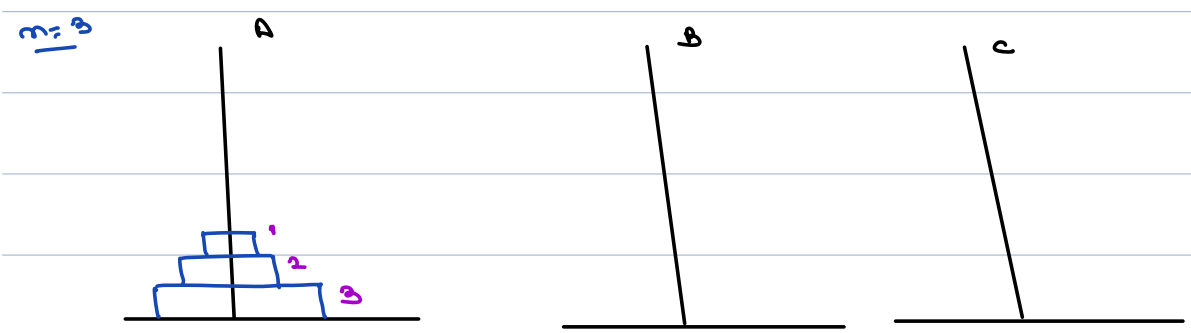
There are n disks placed on tower A of different sizes.

Goal

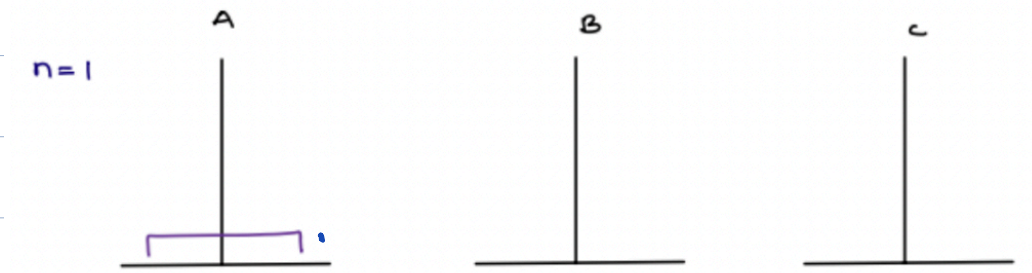
Move all disks from tower A to C using tower B if needed.

Constraint

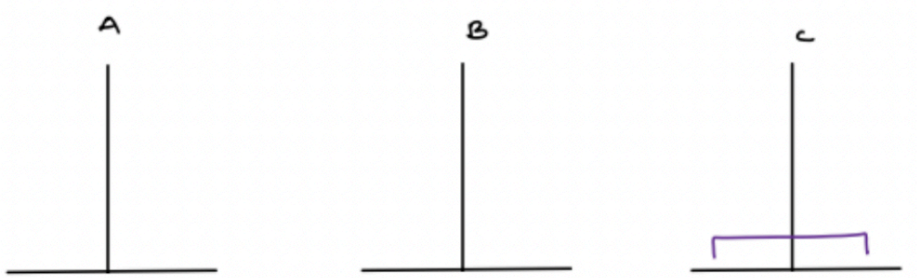
- Only 1 disk can be moved at a time.
- Larger disk can not be placed on a small disk at any step.



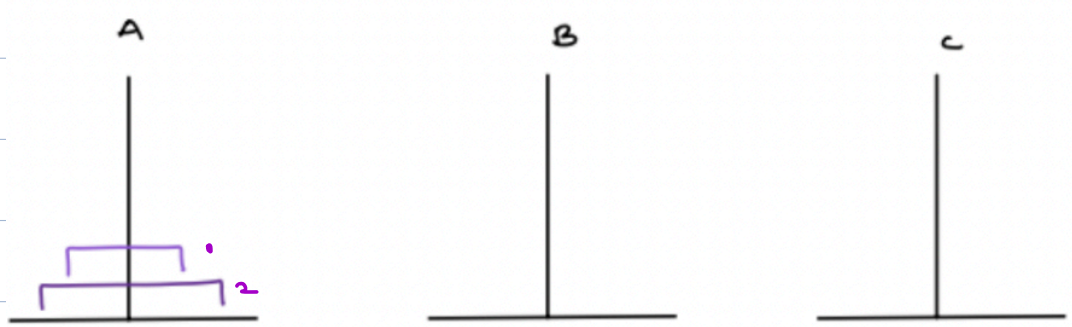
n = 1



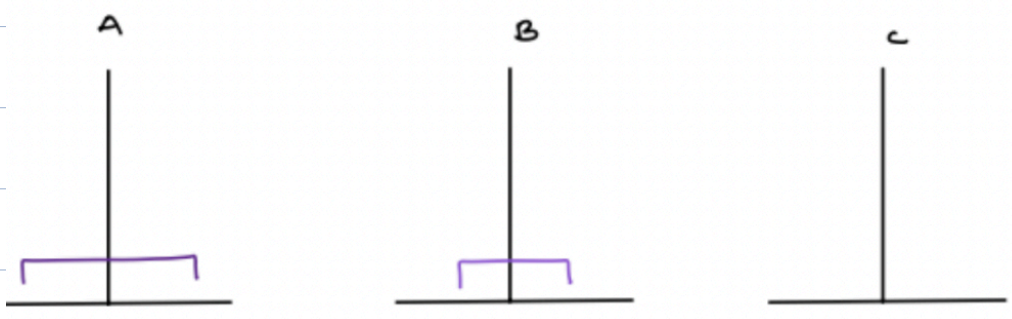
1 : A → C



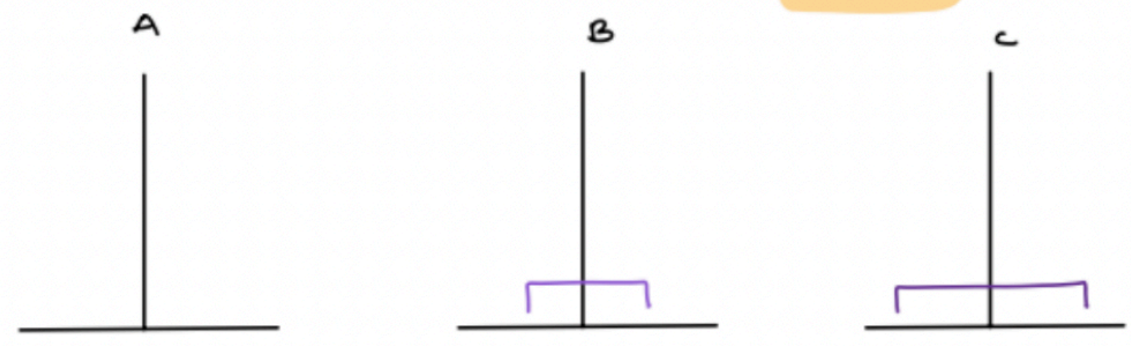
n = 2



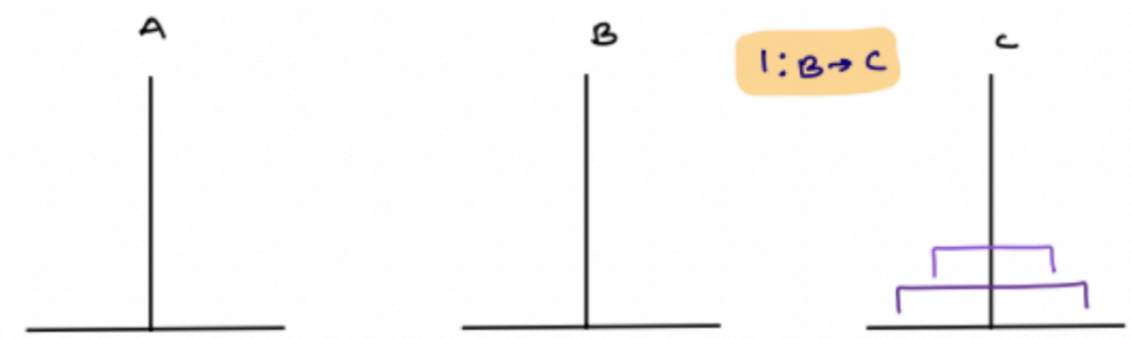
1 : A → B



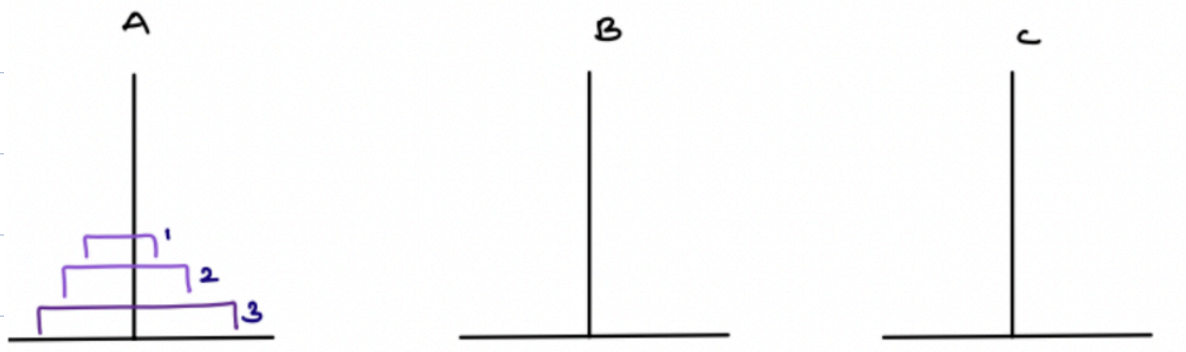
2 : A → C



1 : B → C

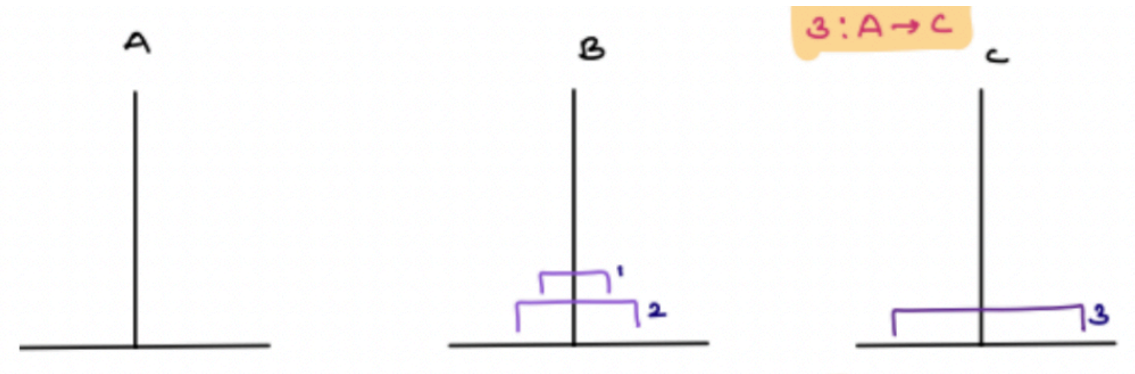


N : 3

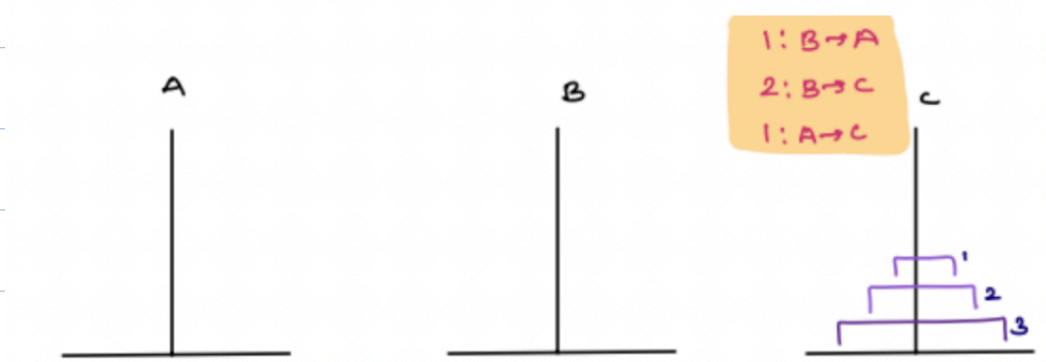


current problem,
move 2 disks from A to B
with help of C.

1 : A → C
2 : A → B
1 → C → B



1 : B → A
2 : B → C
1 : A → C

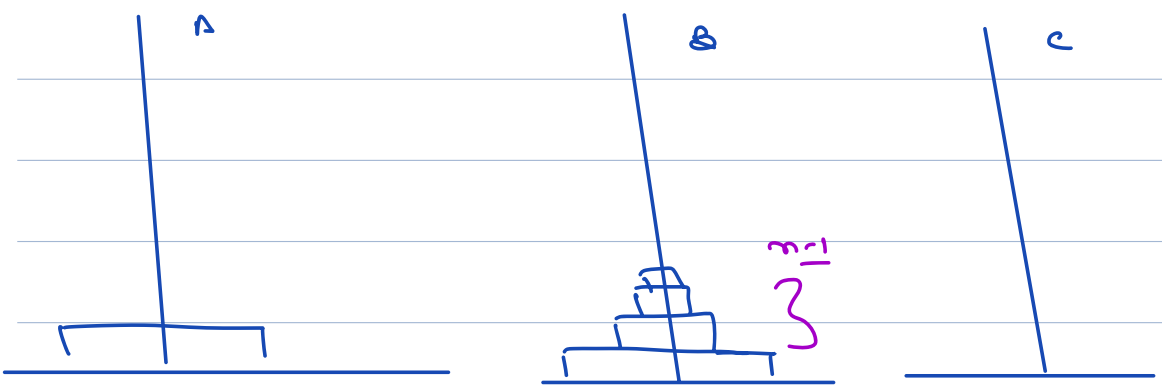


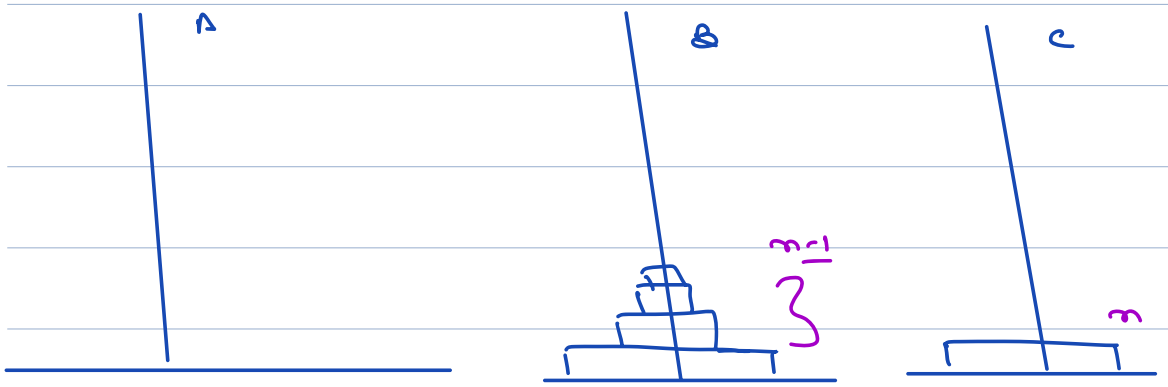
Output:

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

$n = 4$

e.g. n disk





Recurrence Relation

$T(n) = 2T(n-1) + 1$ → Solve.

picking → helper → drop

```
void TOH ( int n, char S, char H, char D) {
    if (n==0) { return 3
    TOH ( n-1, S, D , H) ;
    Print ( n: S ⟶ D ) ;
    TOH ( n-1, H, S , D ) ;
}
```

3

Recursive code :-

1) High level understanding. ✓

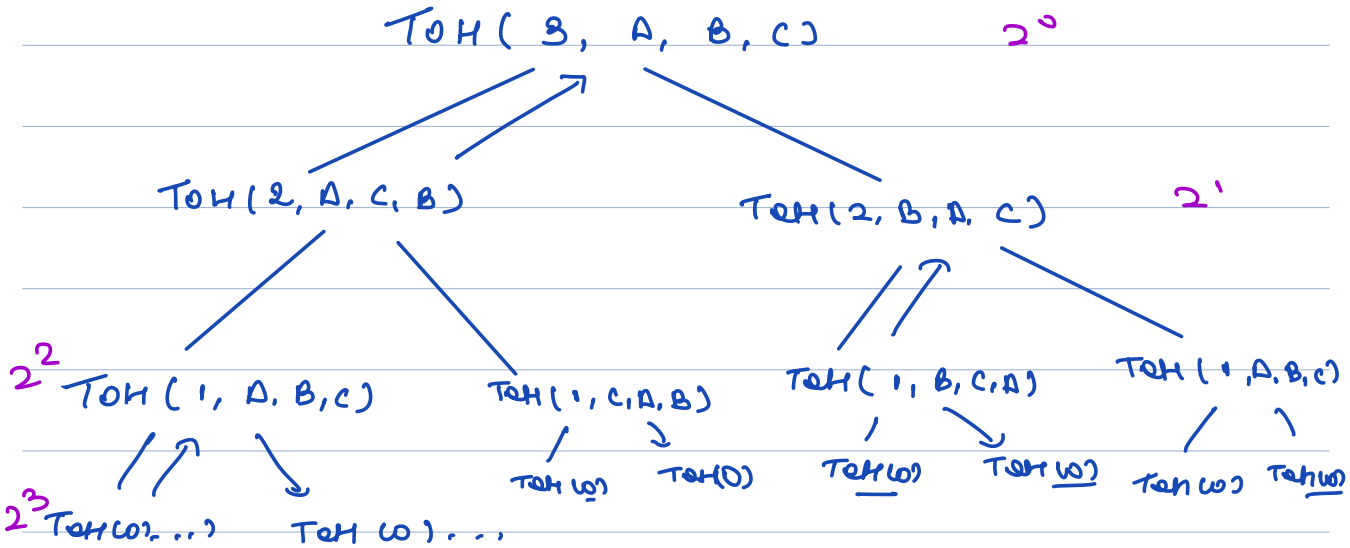
2) Low level understanding ✗.

1 : A → C
2 : A → B
1 : C → B
3 : A → C
1 : B → A
2 : B → C
1 : A → C

picking → s, helper → h, drop → d

```
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



input ⇒ 3

$2^0 + 2^1 + 2^2 + 2^3$ ✓

input = 4

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

input = n,

$2^0 + 2^1 + 2^2 + 2^3 + 2^4 \cdots 2^n$ → g.p.

→ 2^{n+1} ⇒ T.C → $O(2^n)$.

S.C → O(n).

↳ recursive stack space.

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

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

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

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

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

⋮

// generalized expression:-

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

$T(0) = 1$

when $k = n$,

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

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

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

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

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

S.C → O(n).

Indians → ⊗

Breaks 8:36 Am – 8:46 Am

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

e.g.:- n=1, () ,)(✗

n=2, ()(), (()),

n=3, ((())), ()()(),)))(((✗

)(((()) ✗

(()()(), ()(()) , (()()).

n=4, ()()()(), (((())))

()((())), ((()))(), ---

string,

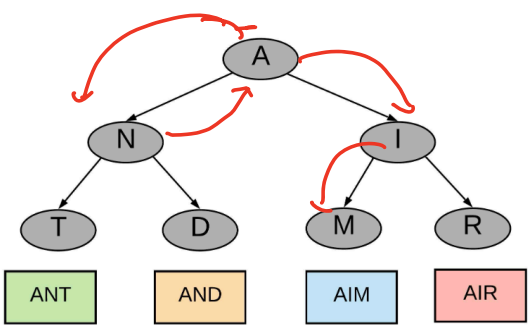
2n , ( ), check their validity,

n=3,

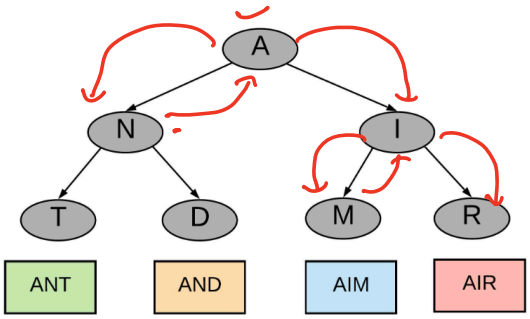
()) ...

Backtracking → recursively trying all the paths.

words

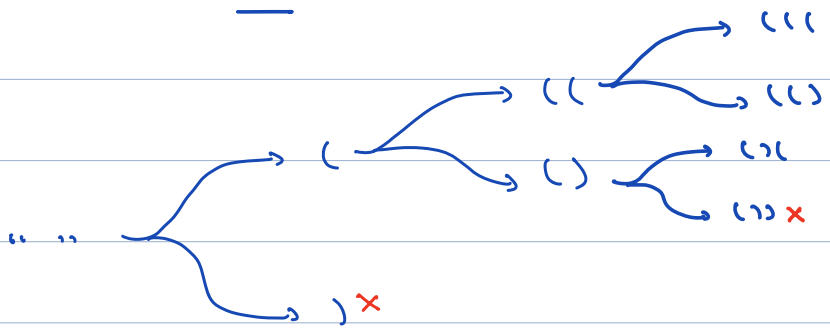


AIM



AIR

n = 2



open_br , closing_br

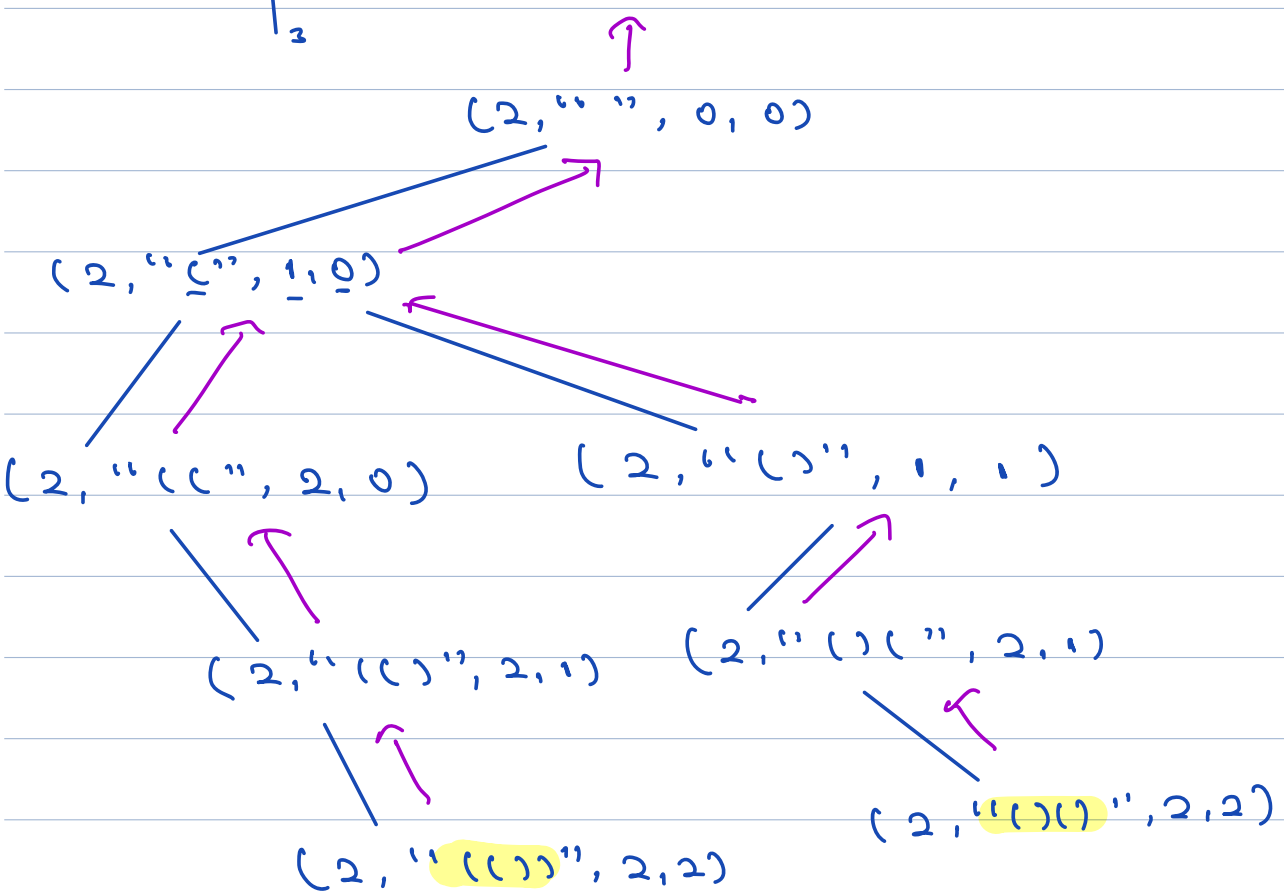
closing_br > openbr

① open_bracket < n → '('.

② open_bracket > close_bracket → ')'.

n = 2

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



T.C → $O(2^n)$.

S.C → $O(n)$ (actual stack space $2*n$).