Lecture: Advanced Recursion

Qu1 Given no. n, find sum of natural no from 1 to n.

sum(5) = 1 + 2 + 3 + 4 + 5
sum(4)

sum(5) = sum(4) + 5.
↑ problem ↑ sub-problem.

```
int sum(int n) {
    if (n==1) {
        return 1;
    }
    sa = sum(n-1);
    return sa + n;
}
```

Assumption:
Given n, find and return sum from 1 to n.

main logic

sum(n)
↑ ans = sa + n.
sa = sum(n-1)

Base case
Smallest subproblem that we can solve

n==1 return 1.
n==0 return 0

TC:

sum(5) ans = 5 + 10 = 15
↓ ↑ 10
sum(4) ans = 4 + 6 = 10
↓ ↑ 6
sum(3) ans = 3 + 3 = 6
↓ ↑ 3
sum(2) ans = 2 + 1 = 3
↓ ↑ 1
sum(1) ⟶ 1

SC: h|w

Qu2 fibonacci number : Given n. find and return nth fibonacci number

Eg: 0 , 1 , 1 , 2 , 3 , 5 , 8 , 13 , 21 , 34 , 55 , 89 - - - -

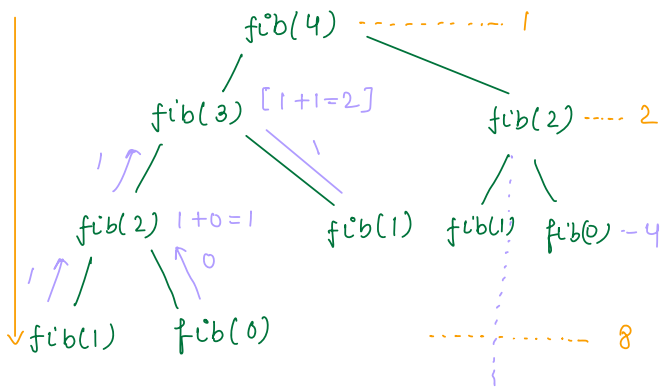
0th 1st 2nd 3rd - - - - - -

fib(3) = fib(1) + fib(2)

2 1 1

fib(n) = fib(n-1) + fib(n-2).

```
int fib(int n) {
    if(n==0 || n==1) {
        return n;
    }
    int sa1 = fib(n-1);
    int sa2 = fib(n-2);
    return sa1 + sa2;
}
```



$1 + 2 + 4 + 8 \cdots 2^n$

$2^0 + 2^1 + 2^2 + 2^3 \cdots 2^n$

GP

$\simeq 2^n$

TC: 2^n

SC: h | w

Qu3

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

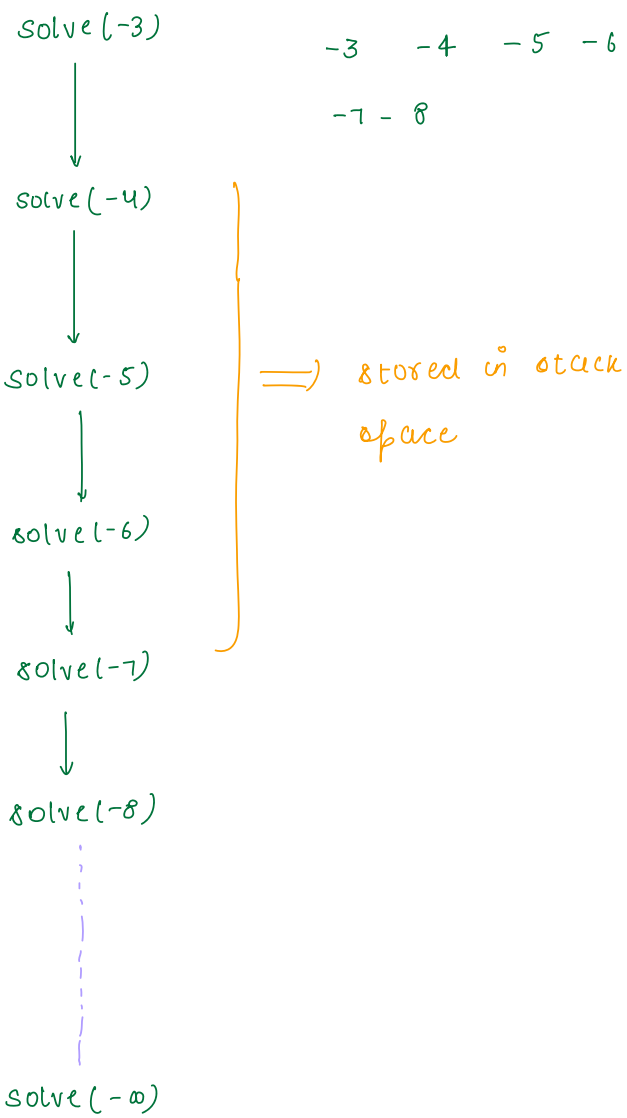
solve(-3) :

a) -3 -2 -1

b) -1 -2 -3

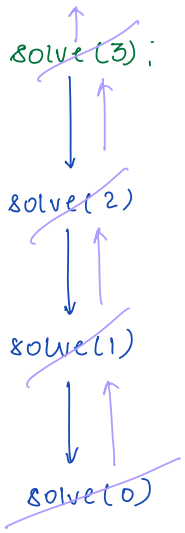
c) stack overflow error

ans) -3 -2 -1 0



Qu4:

```
void solve(int n) {
1.   if (n == 0) {
         return;
     }
2.   print(n);
3.   Solve(n-1);
}
```



a) 1 2 3

b) 1 2 3 0

c) 3 2 1 0

d) 3 2 1

3 2 1

Qu5:

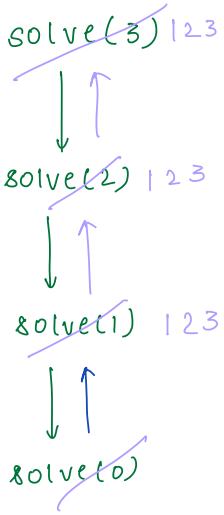
```
void solve(int n) {
1.   if (n == 0) {
         return;
     }
2.   Solve(n-1);
3.   print(n)
}
```

a) 1 2 3

b) 1 2 3 0

c) 3 2 1 0

d) 3 2 1



1 2 3

Qu6 **Tower of Hanoi**

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

Goal: Move all disks from tower A to tower B using tower C.

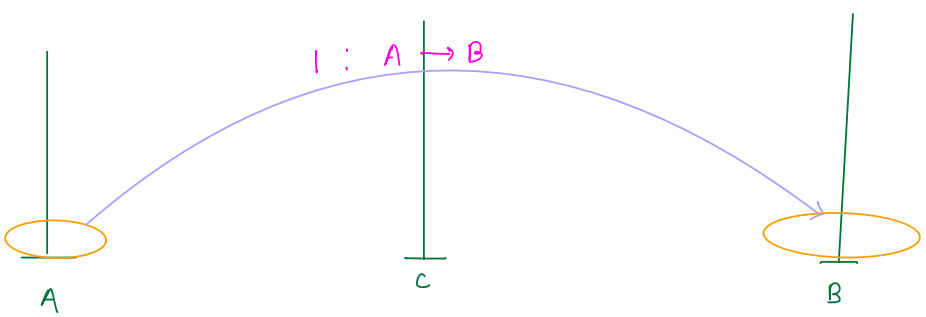
Constraint:

Only 1 disk can be moved at a time.

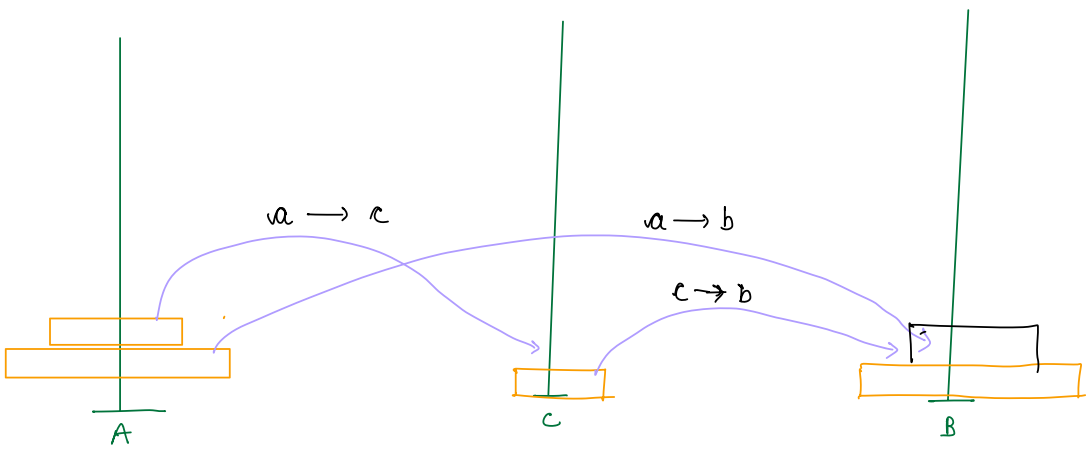
Large disk cannot be placed on smaller disk.

Problem: print the moves such that all disks move from tower A to B.

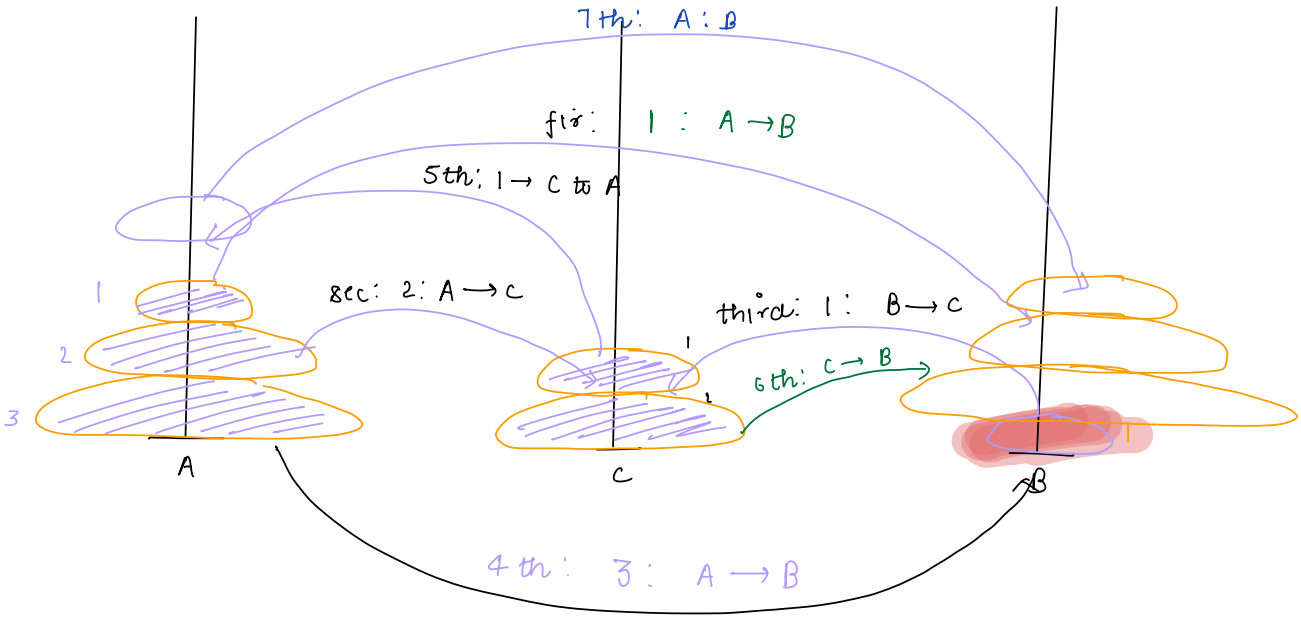
Eg: n = 1



n = 2



n = 3 [n = 2 : source: A dest: C aux: B ; n = 2 : source: C dest: B aux: A]

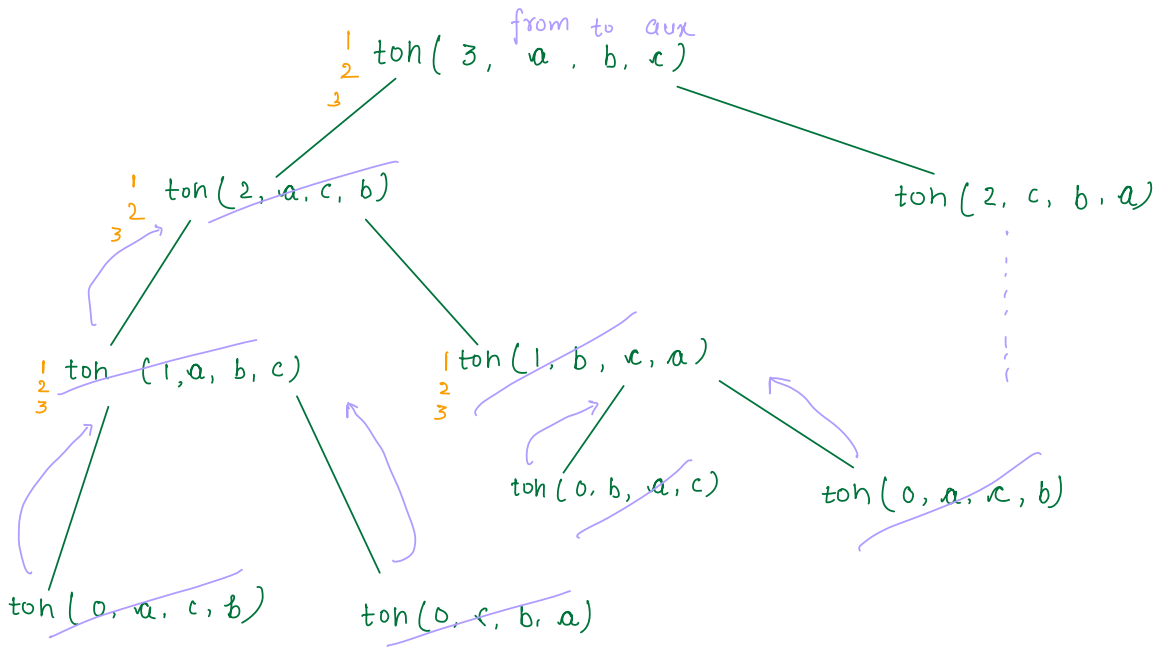


Pseudo code:

```
void towerofHanoi(int n, char from, char to, char aux) {
    if (n == 0) {
        return;
    }
1   towerofHanoi(n-1, from, aux, to);
2   print(n + " : " + from + " → " + to);
3   towerofHanoi(n-1, aux, to, from);
}
```

Dry run:

```
void towerofHanoi(int n, char from, char to, char aux) {
    if (n == 0) {
        return;
    }
1   towerofHanoi(n-1, from, aux, to);
2   print(n + ":" + from + "⟶" + to);
3   towerofHanoi(n-1, aux, to, from);
}
```



prints: 1 : a ⟶ b

2 : a ⟶ c

1 : b ⟶ c

3 : a ⟶ b

⋮

TC: n = 1 [1] ⟶ $2^1 - 1$: $O(2^n)$

n = 2 [3] ⟶ $2^2 - 1$

n = 3 [7] ⟶ $2^3 - 1$

⋮

n. [$2^n - 1$]

SC: h/w

Break: 8 : 33 AM

Qu: **Generate parenthesis**

given no. n, Generate all valid parenthesis of sequence len: 2*n

Eg: n=1 ()

(( )))(— Invalid

n=2 (())

()()

n=3 ((()))

(())()

()()()

()(())

Brute force: Generate all parenthesis.

n=2. [(( ))]

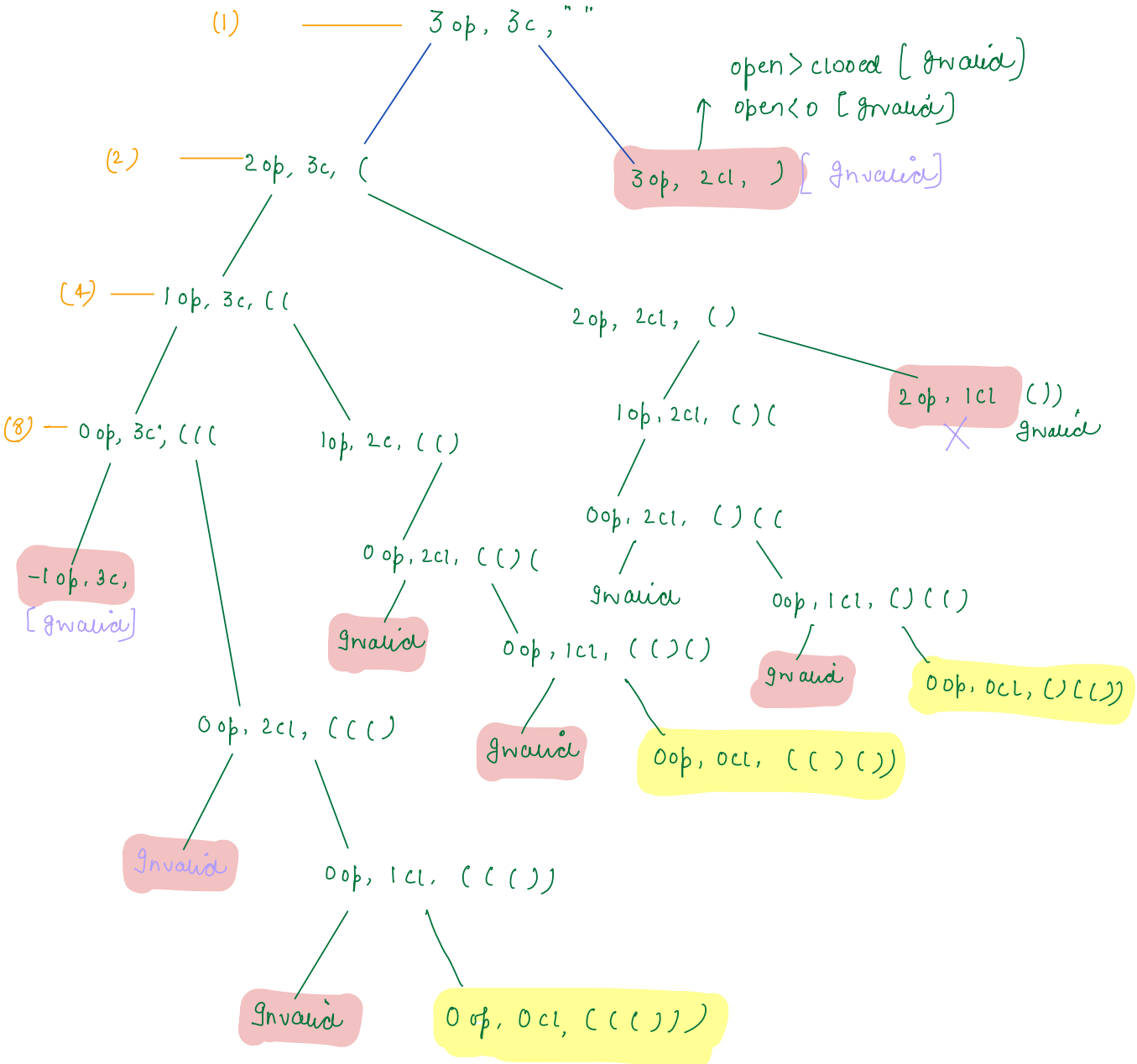
↑

Generate all subsets [Backtracking]

↓

Check for valid ones [stack].

n = 3 [3 open, 3 closed]



```
void generate (int n) {
        generate (n, n, "");
}
```

```
void generate( int open, int closed, string seq) {
        if( open==0 && closed==0)
                print (seq);
                return;
        }
        if ( open>0) {   // left part : deal with (
                generate( open-1, close, seq + '(');
        }
        if ( open < closed) {   // right part : )
                generate ( open, clooed-1 , seq + ')');
        }
}
```

TC : $O(2^n)$

SC: h|w

Thankyou ☺

Doubts:

A = [2 , 3 , 4]

B[] = [2 6 24]

↑ factorial

↑ total no of non-empty subsequences such that each el in subsequence has same prime factors.

B[] = [2 6 24]

2 → , 6 → , 24 → } can

2 6 → 2: 2 ✗
6: 2, 3

6 24

6: 2 3

24: 2 3

2 24

2 6 24

a b c

c
b
a
a b
a c
b c
a b c