Recursion 2

Content

- Quizzes on Recursion
- Tower of Hanoi
- Balanced Parenthesis.

Contest → 9:00 pm { 5 mins early }

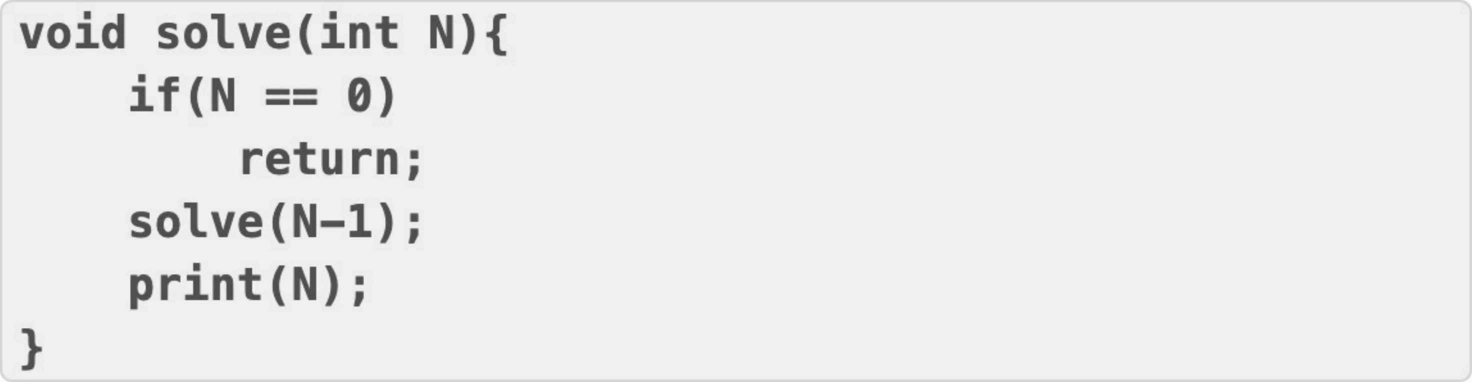
↓

10:30 pm duration of contest

3 DSA problems

Contest Discussion 10:30 pm

What is the output of the following code for N = 3?



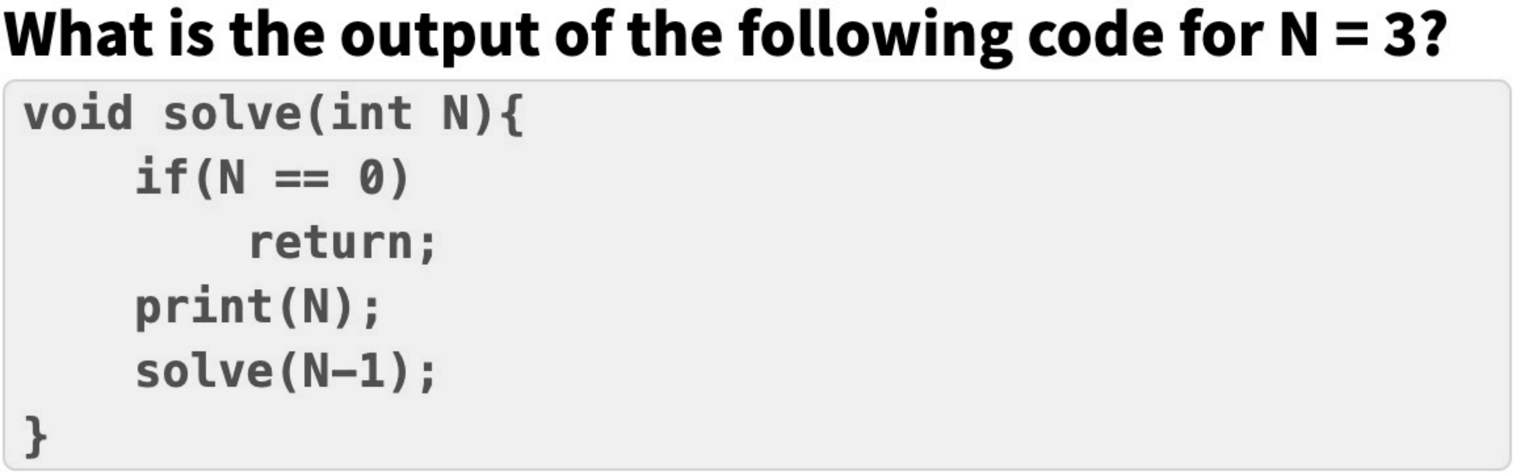
```
void solve(int N){
    if(N == 0)
        return;
    solve(N-1);
    print(N);
}
```

```
solve (3) {                                                   output
|     solve (2) {                                               1
|     |     solve (1) {                                         2
|     |     |     solve (0) { return }                          3
|     |     |     print (1)
|     |     }
|     |     print (2)
|     }
|     print (3)
}
```

1

2

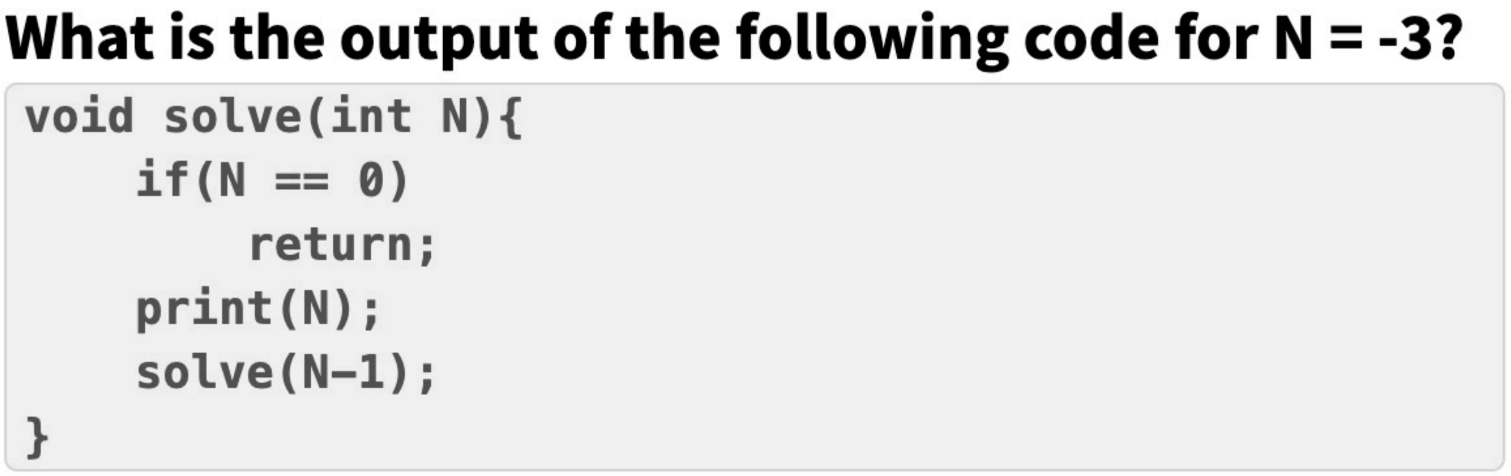
3



What is the output of the following code for N = 3?

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

```
solve (3) {                                              Output
|       print (3)                                          3
|       solve (2) {                                        2
|       |       print (2)                                  1
|       |       solve (1) {
|       |       |       print (1)
|       |       |       solve (0)  { return }
|       |       }
|       }
}
```



What is the output of the following code for N = -3?

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

```
solve (-3) {
    print (-3)
    solve (-4) {
        print (-4)
        solve (-5) {
            print (-5)
```

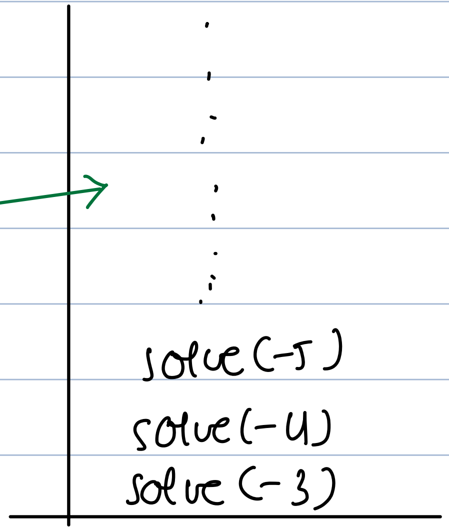
Output

-3

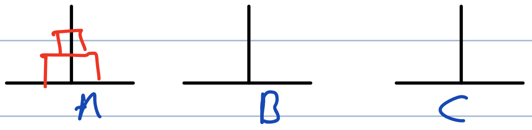
-4

-5

Stackoverflow



Tower of Hanoi ***



There are n disks placed on a tower A
{different sizes}

Move all disks from tower A to C using B

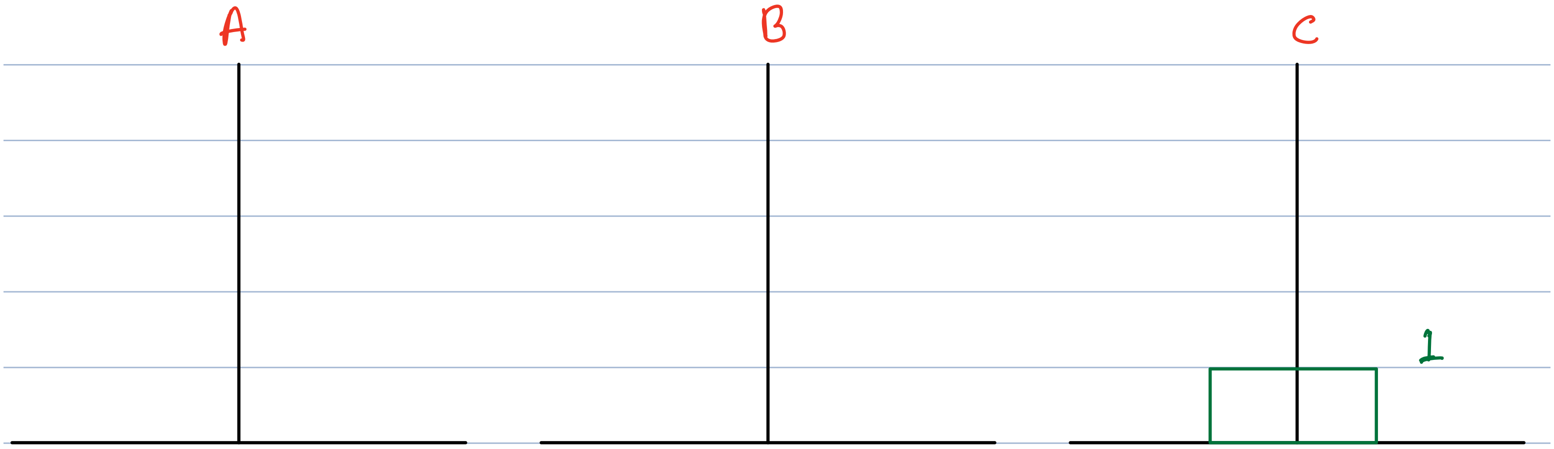
Constraints

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

Print the movement of disks from A to C in minimum steps.

Example

N = 1

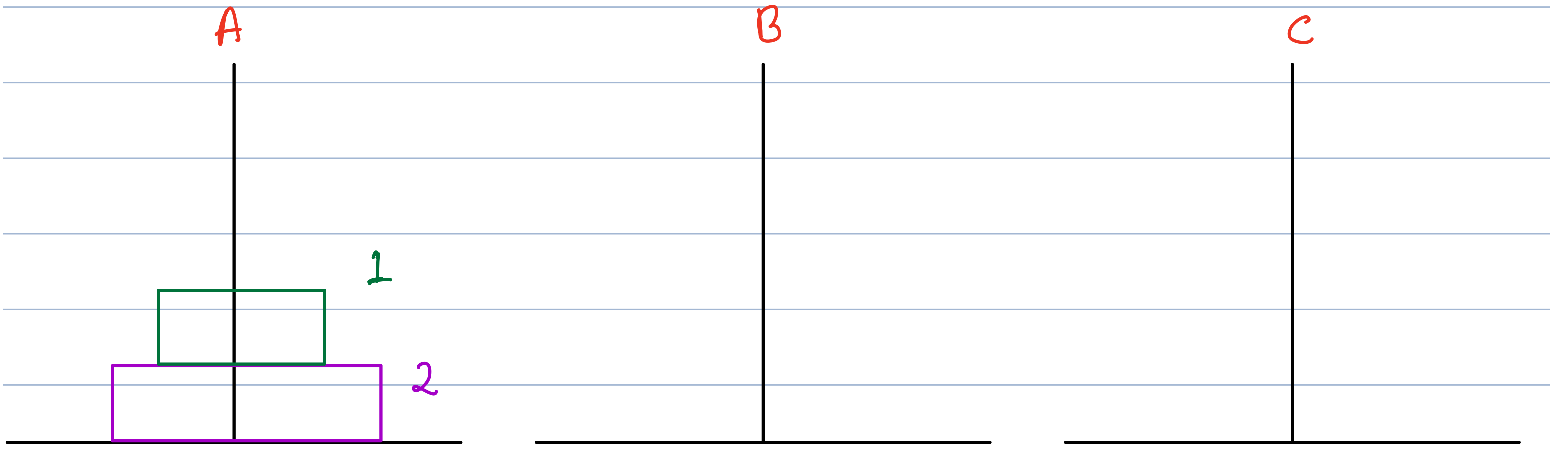


Disk 1 A ⟶ C

N = 2

TOH (2, A, B, C)

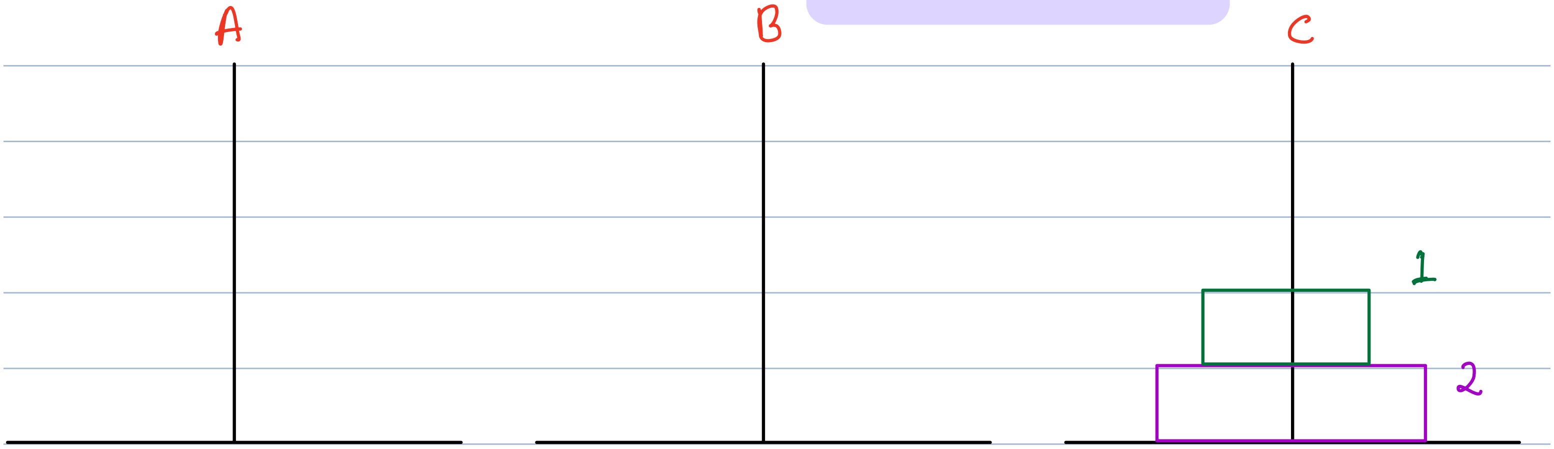
Initial State



steps →

D 1 A → B
D 2 A → C
D 1 B → C

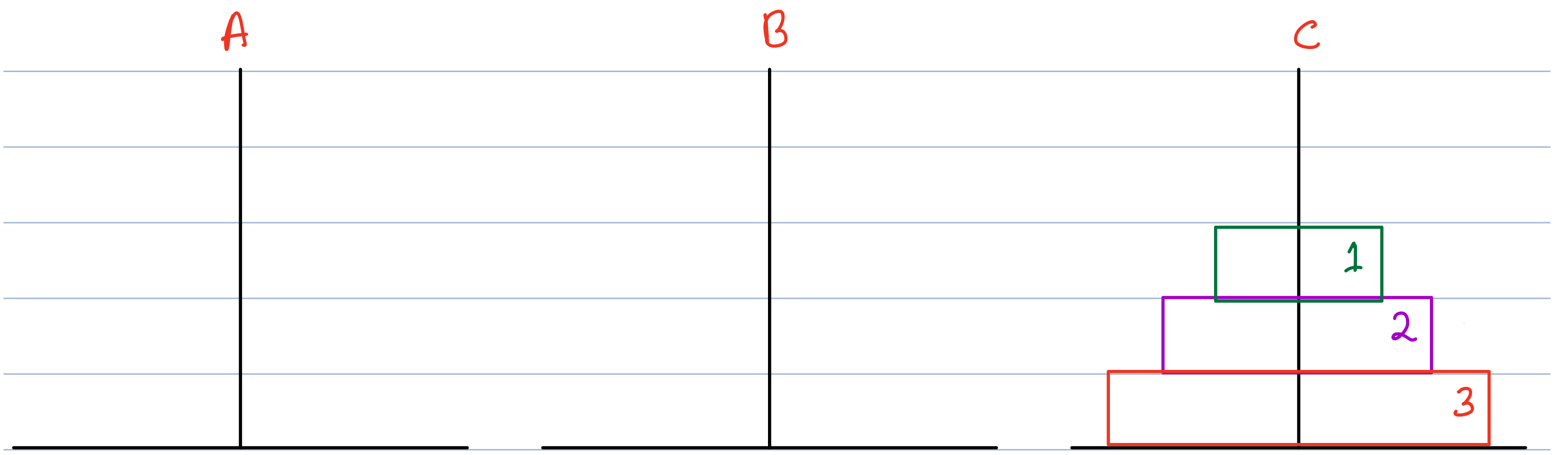
Final state



Assumption → TOH (N, A, B, C)

→ Move N disks from tower A to tower C using a spare tower B

N = 3



We moved 2 disks from tower A to tower B using C as spare tower.

spare

from to

TOH (2, A, C, B)

D3 A → G

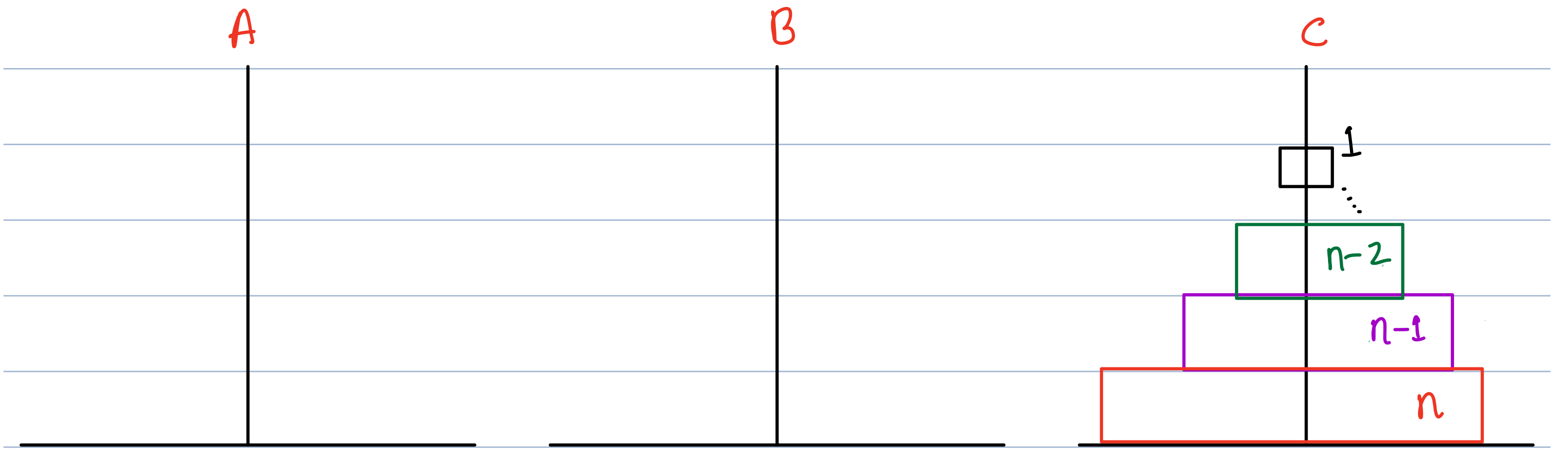
TOH (2, B, A, C)

Steps

D1 A → C
D2 A → B
D1 C → B

D3 A → C

D1 B → A
D2 B → C
D1 A → C



```
          fr   sp   to
TOH ( N , A , B , C )                    Step 1

// Move N-1 disks from A to B            Step 2
         using spare C

      TOH ( N-1 , A , C , B )
      print ( D-N , A → C )
// Move N-1 disks from B to C
        using spare A
      TOH ( N-1 , B , A , C )
```

Base case

```
    if N == 0 return.
```

Pseudocode

```
                        spare
                 fr       ↓       to
void   TOH ( N,  A  ,    B  ,   C ) {
        if (N==0)  return
        TOH (N-1 , A , C , B )
        print ( D-N , A → C )
        TOH (N-1 , B , A , C )
}
```

N	steps	
1	1	2-1
2	3	4-1
3	7	8-1
4	7 1 7 = 15	16-1
5		32-1
⋮		
n	2^n-1	

```
N= 2
                 spare
          fr       ↓       to
          ↓                ↓
TOH (2,  A ,      B,       C ) {
      TOH ( 1 , A , C , B ) {
            TOH ( 0 , A , B , C ) { return }
            print (1 , A → B )
            TOH ( 0 , C , A , B ) { return }
      }

      print ( 2, A → C )
```

```
                        spare
                 fr       ↓       to
void   TOH ( N,  A  ,    B  ,   C ) {
        if (N==0)  return
        TOH (N-1 , A , C , B )
        print ( D-N , A → C )
        TOH (N-1 , B , A , C )
}
```

```
        TOH ( 1 , B , A , C ) {
            TOH ( O , B , C , A ) { return }
            print ( 1   B ⟶ C )
            TOH ( O , A , B , C ) { return )
        }
}
```

Output

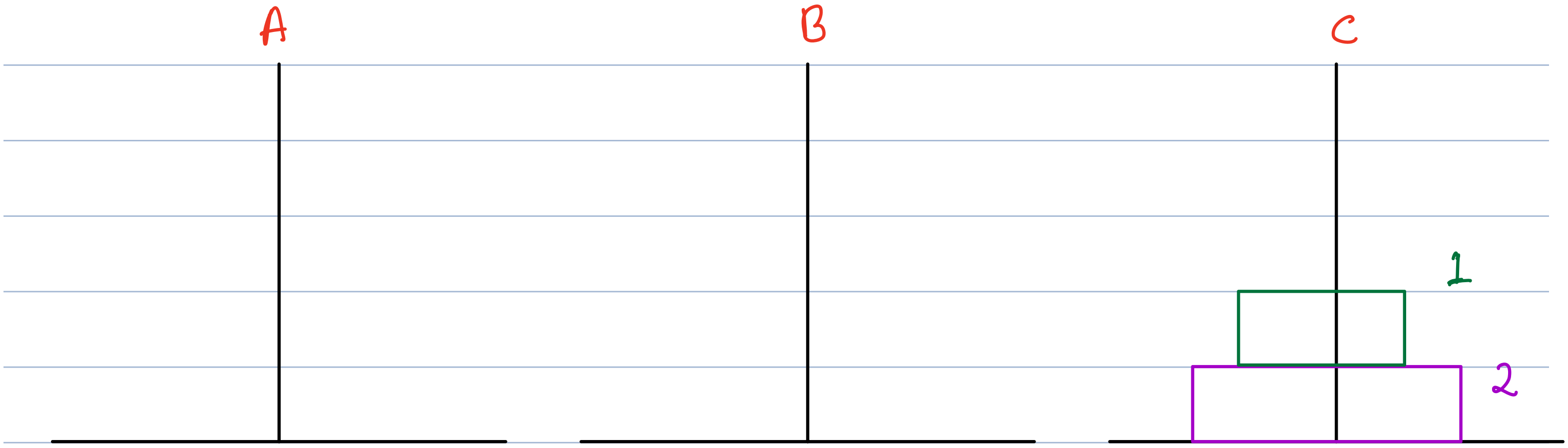
1 A ⟶ B

2, A ⟶ C

1 B ⟶ C

TOH (2, A, B, C)

Initial State



Break : 22 : 34

TC : $O(2^N)$

SC : O(N) ⟶ recursive stack space.

TOH (N .....)
TO (N -1 .....)
TOH (N-2 ....)
TOH (O .....)

N+1 function calls at max.

Valid Parenthesis

Print all valid parenthesis of length 2*N for a given N

at any instant open >= close } cond for valid parenthesis
open == close
== N

Example

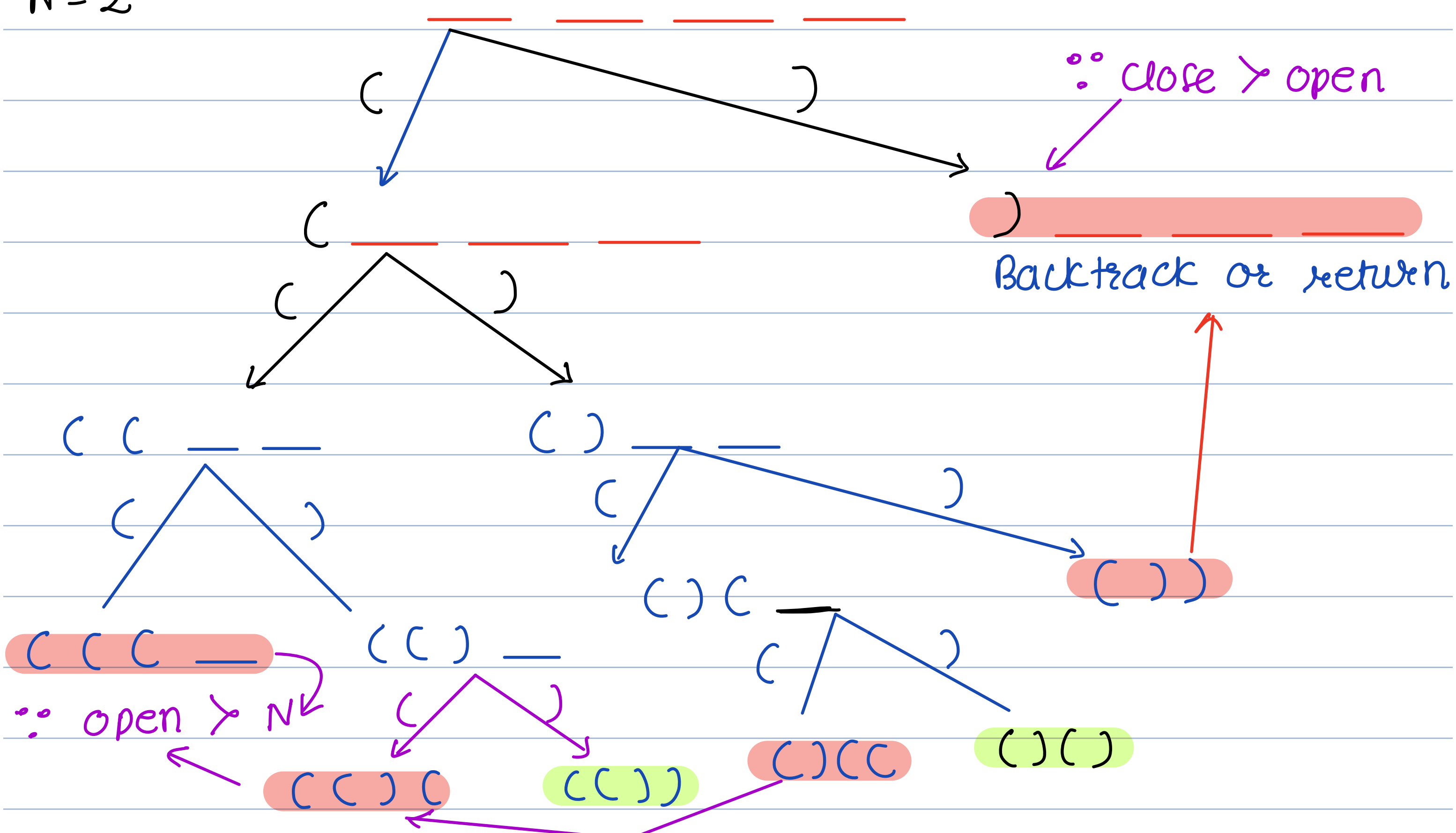
Output

N = 1 ()

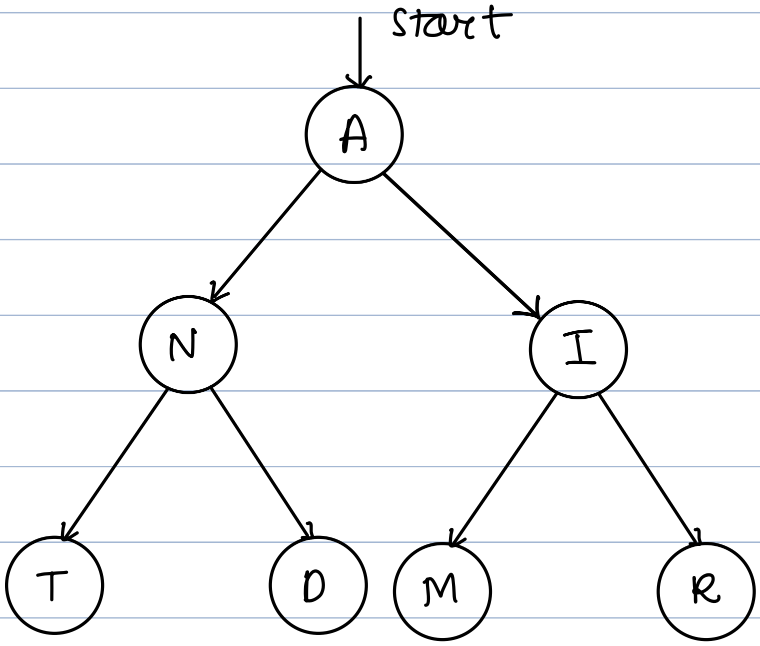
N = 2 (()) ()()

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

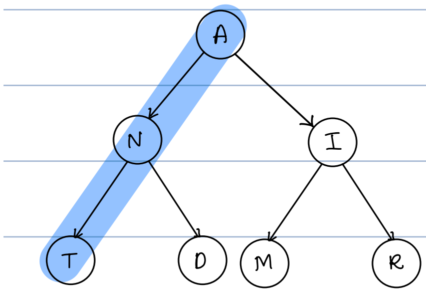
N=2



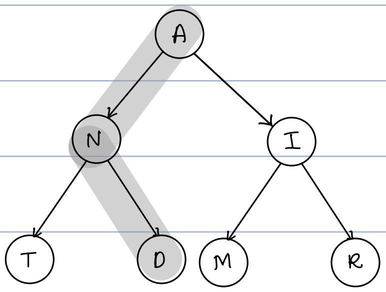
what is Backtracking ?



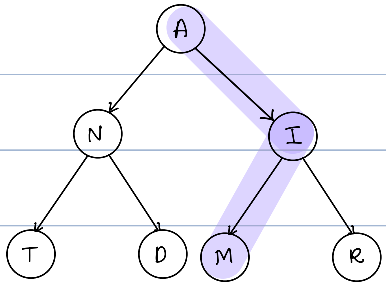
search for AIM



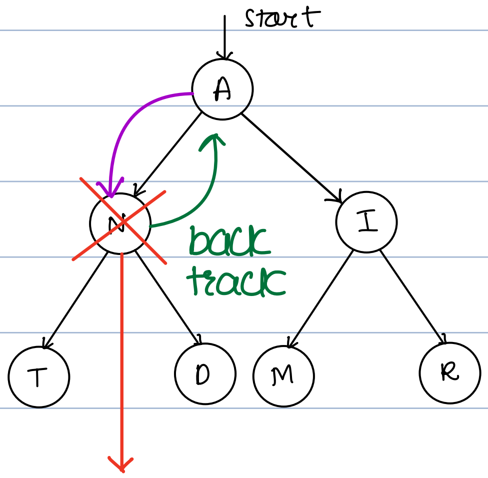
ANT != AIM



AND != AIM



AIM == AIM



Try out all possibilities to find a solution.

If sol" is not possible from a path we exit early { backtrack}

Backtrack is a technique

I can never find AIM, if I doesn't match with N

Pseudocode

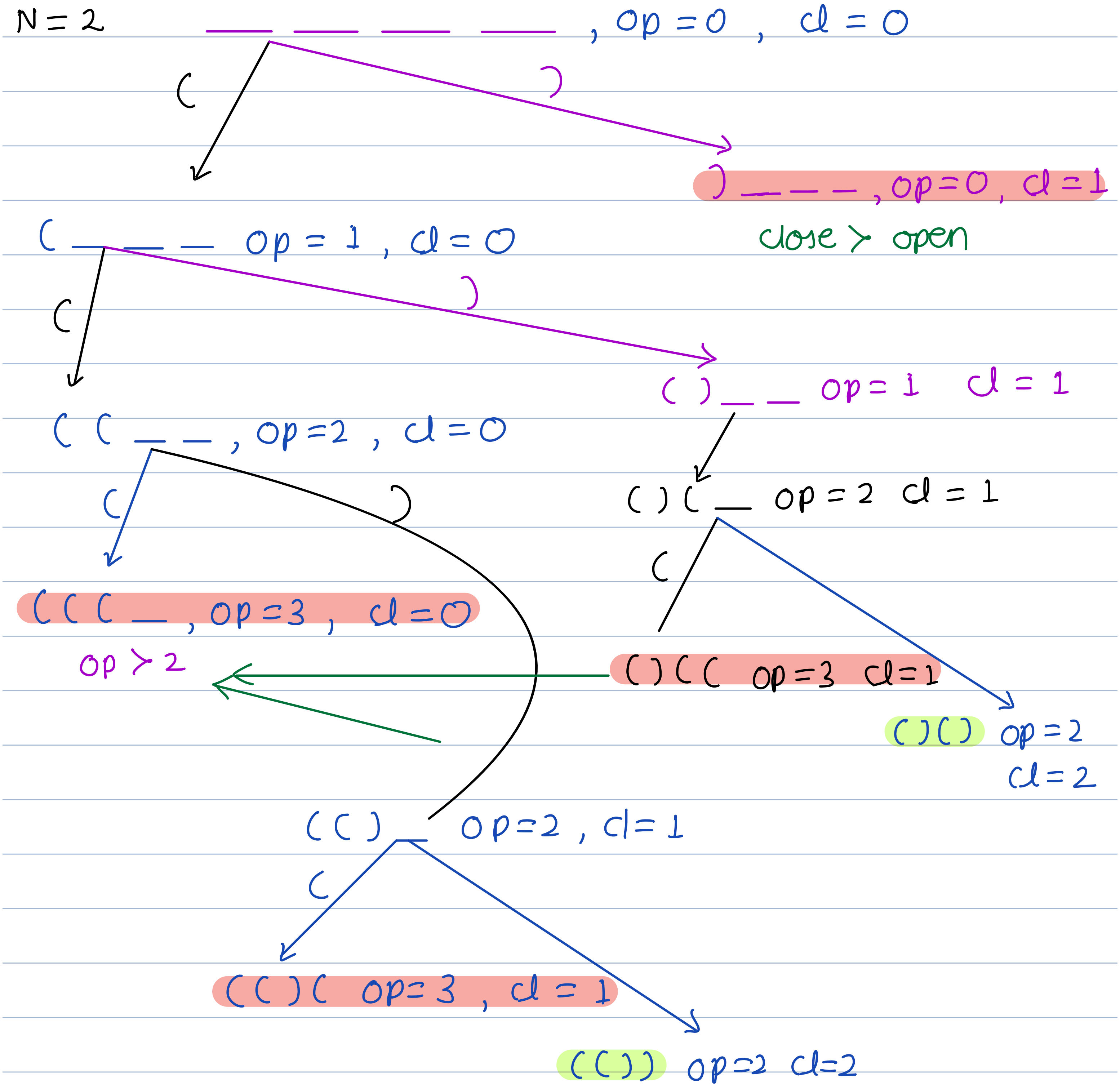
```
                                     ''            0             0
void   valid ( int N , String s , int open , int close ) {
        // Base condition
        // opening brackets cannot exceed N
         if ( open > N )  return
        // closing bracket > opening
         if ( close > open )  return

         // cond" for valid ans                              OR
         if (open == N && close == N) { // s.len = 2N
               print (s)
               return
         }

         // Main logic
         // add opening bracket
         valid ( N , s + '(' , open + 1 , close )
         // add closing bracket
         valid ( N , s + ')' , open , close + 1 )
}
```

Output

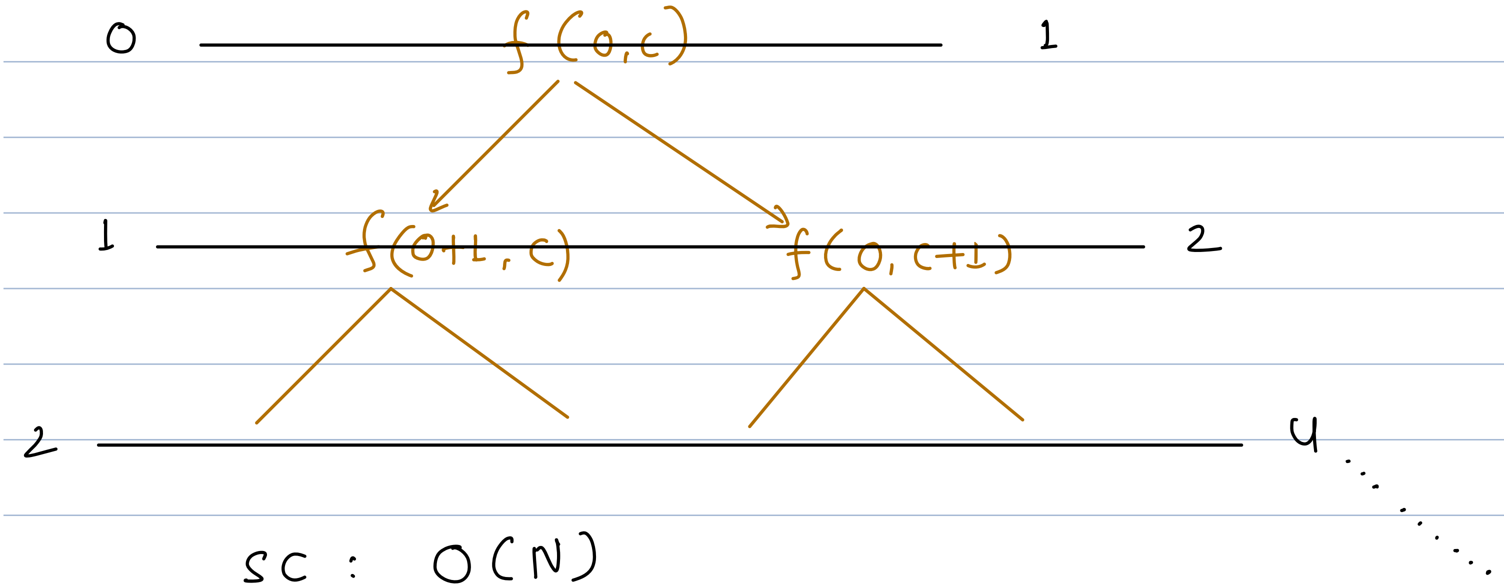
(())

()()

```
void valid (int N, String s, int open, int close) {
    if (open > N) return
    if (close > open) return
    if (open == N && close == N) {
        print (s)
        return
    }
    valid (N, s + '(', open + 1, close)
    valid (N, s + ')', open, close + 1)
}
```



TC for valid Parenthesis. TC : O(2^N)



SC : O(N)