High Achievers

Nov23_PSP_22Jan
VIDYA CHAITANYA
Vijay V A
Vigneshwaran K
Tushar Desarda
Sai Sharath
Suraj Devraye
Kevin Theodore E
Prabhakar
Shaurya Srivastava
Mayur Hadawale
Pranadarth S
Varun
Yash Malviya
Harshil Dabhoya
Panduranga

Nov23_PSP_22Jan
RAMESH R S
Shashi
Pratham Singh
Manjunatha I
Rajeev Singh Khati
Sarat Patel
M.veronica
Rahul Kotha
ALLEN GEOSHAN M
Manikandan M
Mateen
Akanksha Narayan Shinde
Pushkar Deshpande
SIJU SAMSON
Pravin Raj

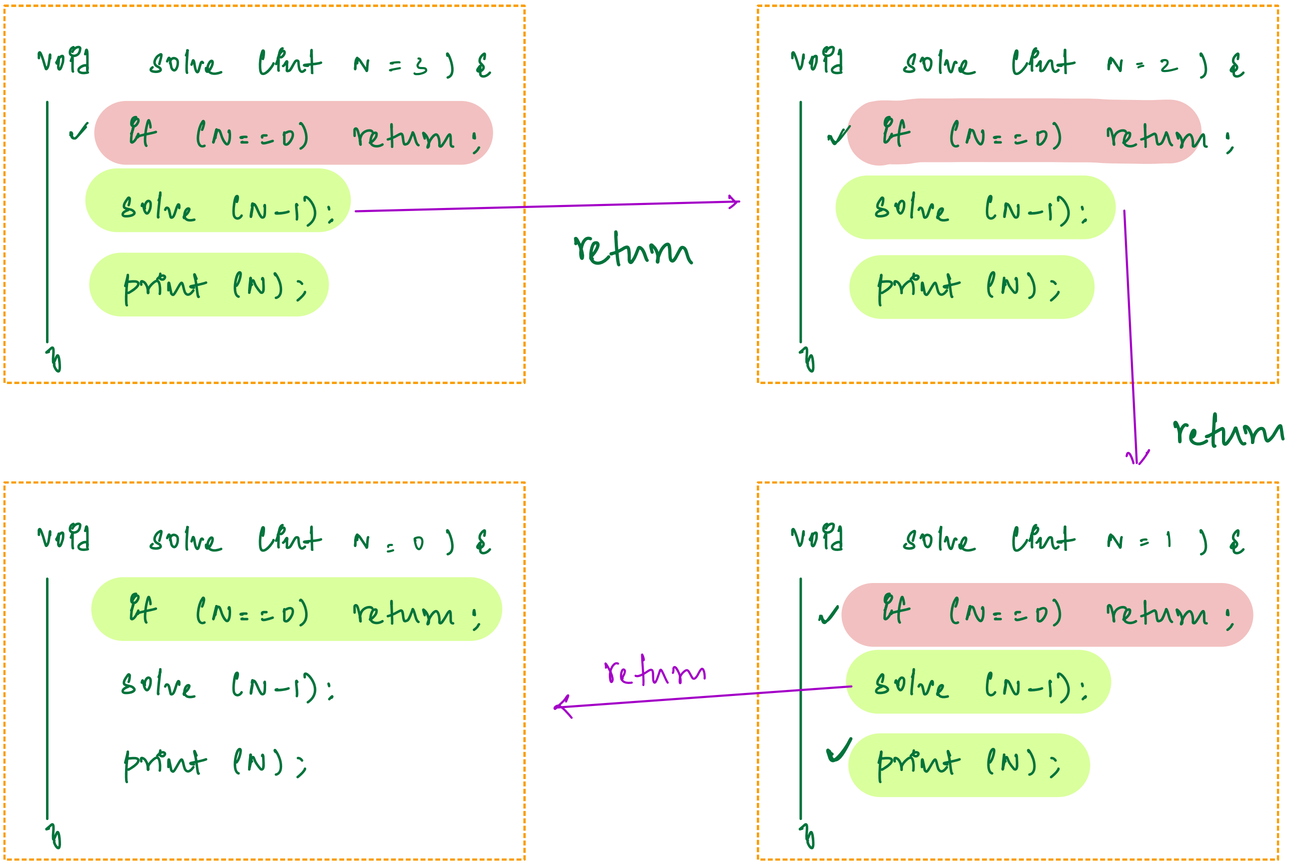
Quiz 1

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

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

Tracing

Output : 1 2 3



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

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

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

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

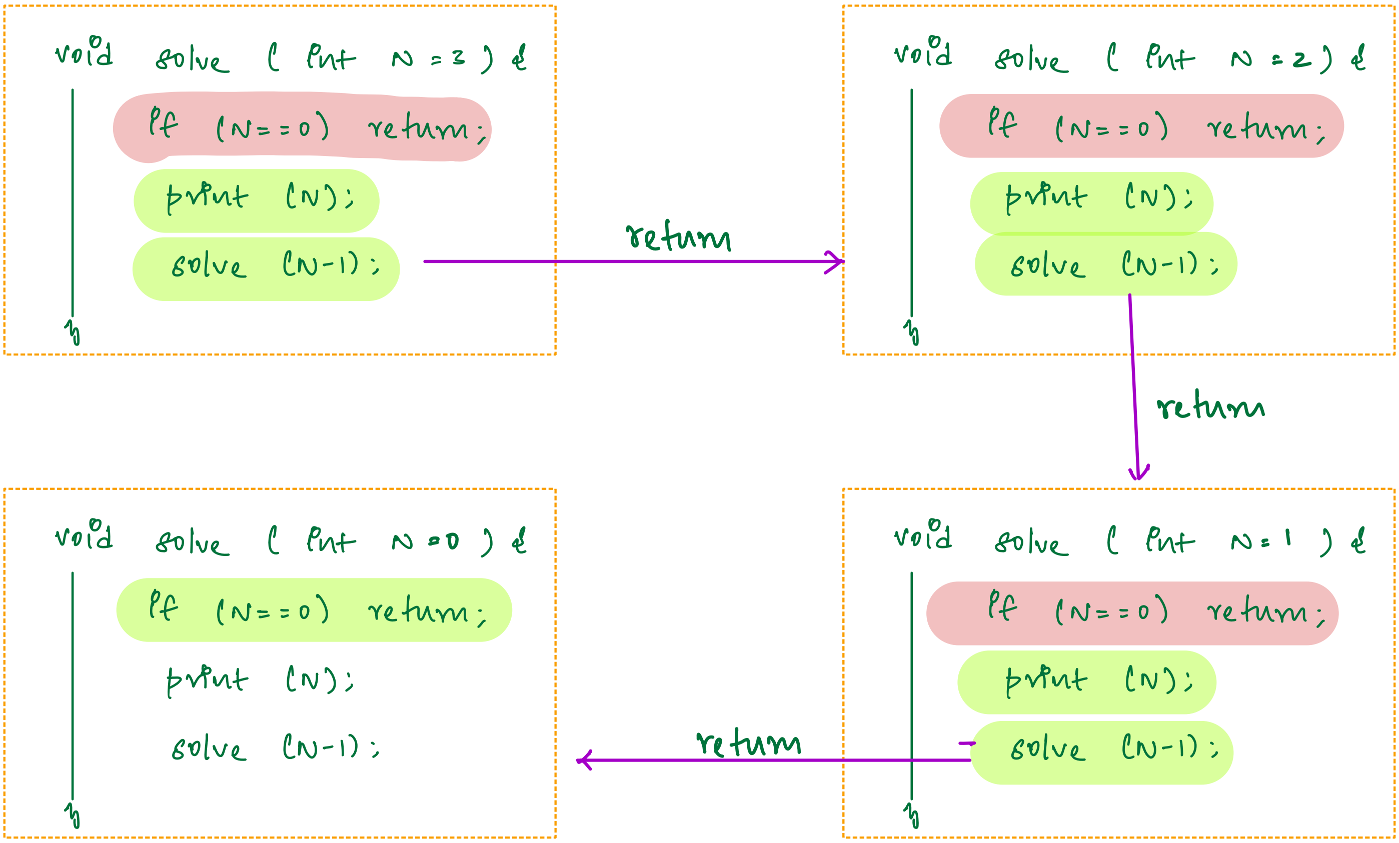
Quiz 2

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

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

Tracing

Output : 3 2 1



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

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

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

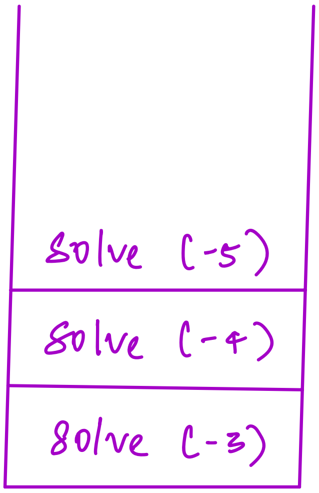
```
void solve ( int N = 0 ) {
    if (N==0) return;
    print (N);
    solve (N-1);
}
```

Quiz 3

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

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

// infinite loop and exits with out of memory



Tracing

-3 -4

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

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

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

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

Time Complexity

T.c = # of recurrence calls * T.c of individual call

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

Time taken for single function call = O(1)

Total calls

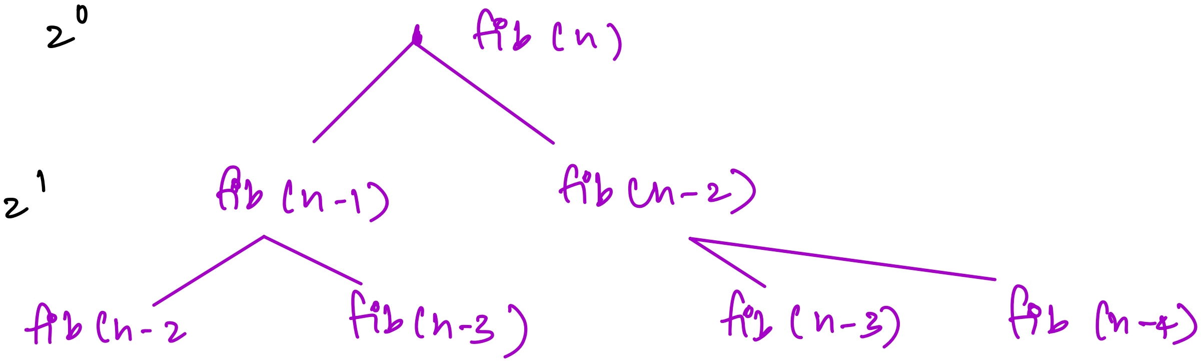
level 0 = 2^0

1= 2^1

2 = 2^2

⋮

n = 2^n



Add : $2^0 + 2^1 + 2^2 + \dots + 2^n$

GP

$a = 1 \qquad r = 2 \qquad n = n \qquad \dfrac{a(r^n - 1)}{(r-1)} = 2^n$

$T.C = 2^n * 1$

$= O(2^n)$

Tower of Hanoi

There are n disks placed on tower A of different sizes

Goal :

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

Constraints

① Only 1 disk can be moved at a time.

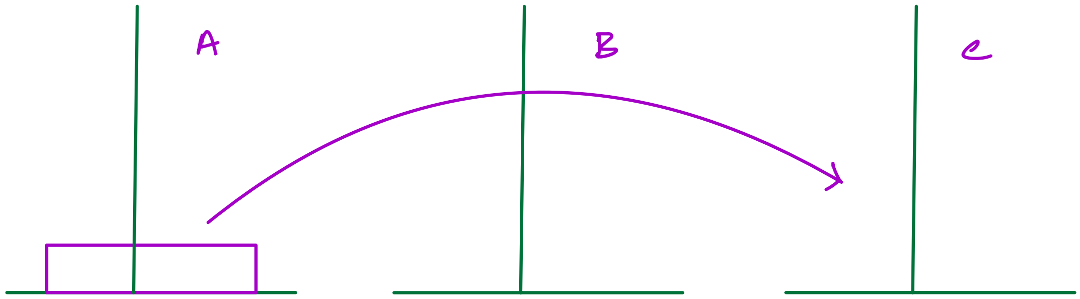
② Larger disk cannot be placed on a small disk at any step.

Example 1

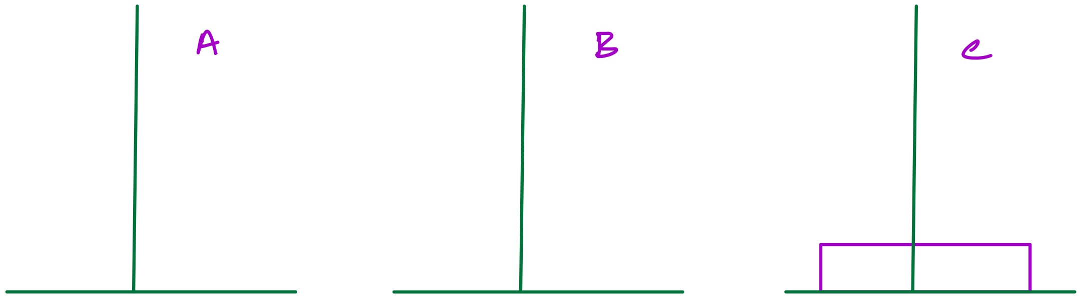
N=1

1step

1: A → C

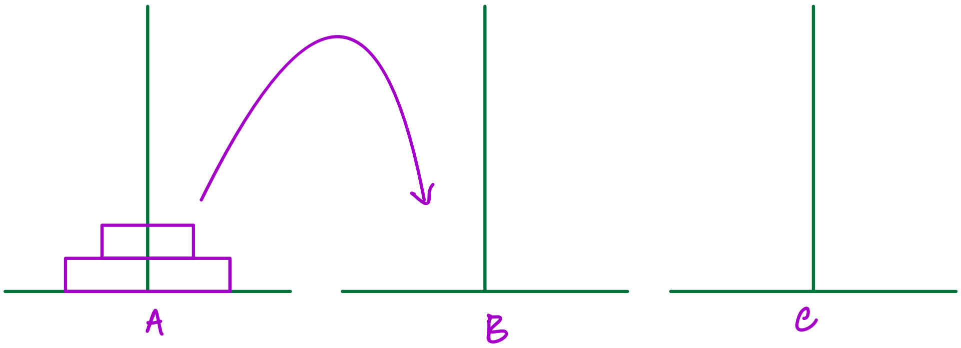


Output :



Example 2

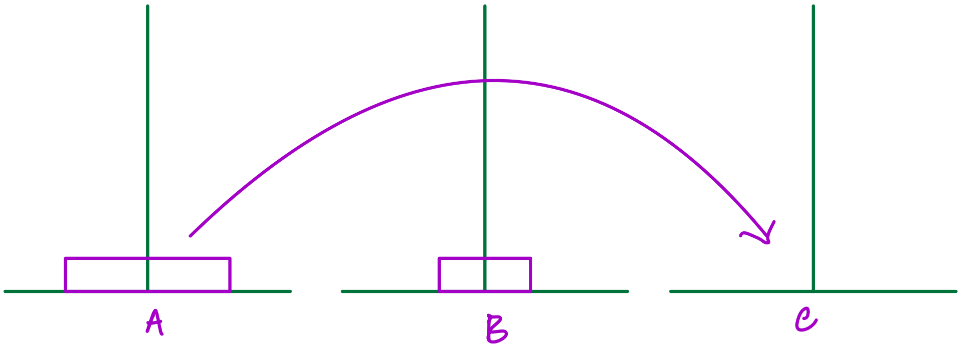
n=2



Step 1:

move disk from

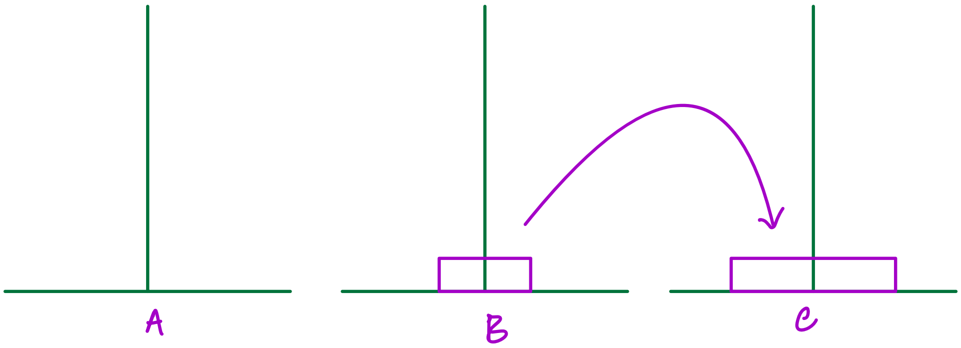
A → B



Step 2:

move disk from

A → C

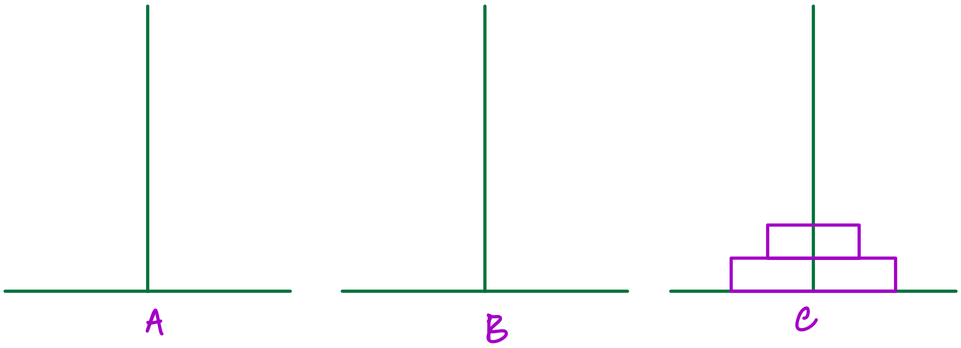


Step 3

Move disk from

B → C

Result



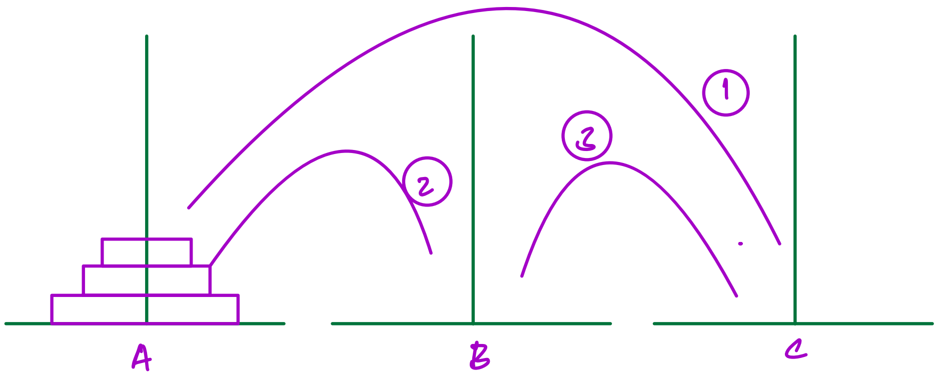
2 Blocks from

A → C using

B

Example 3

N = 3



Subproblem 3

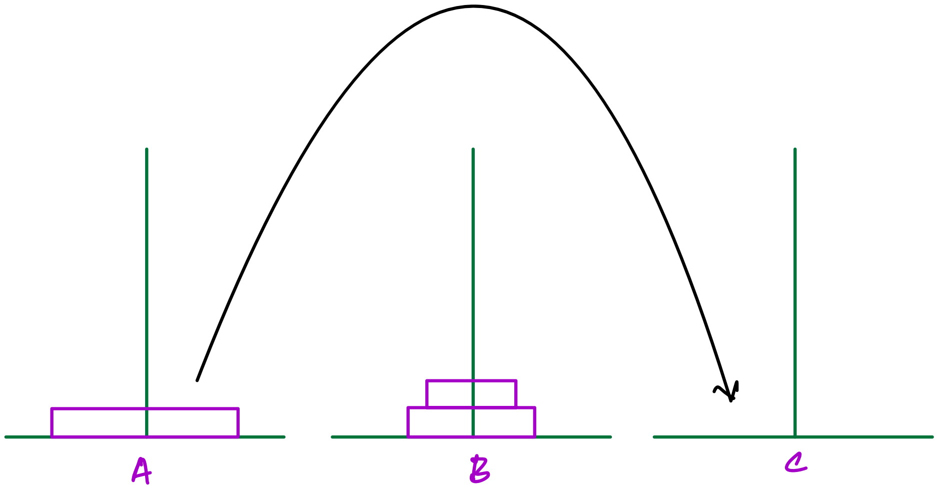
move 2 blocks

from A to B

using C

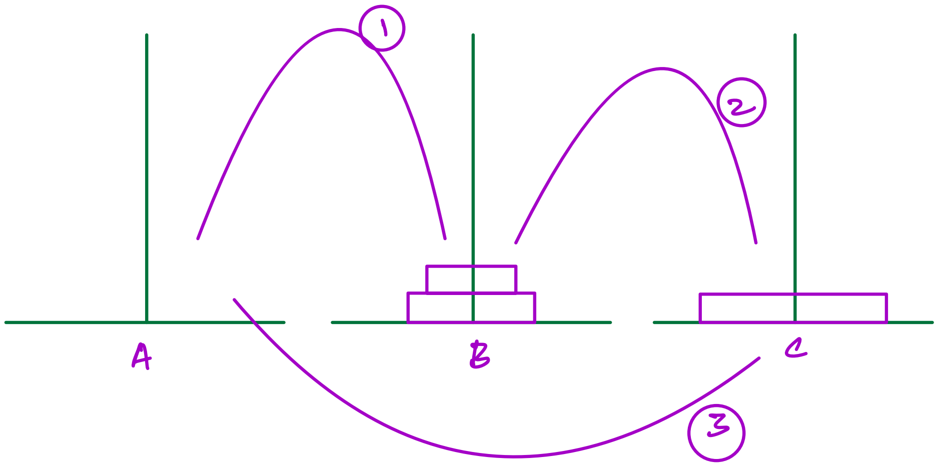
A → C   A → B

C → B



Intermediate step

A → C



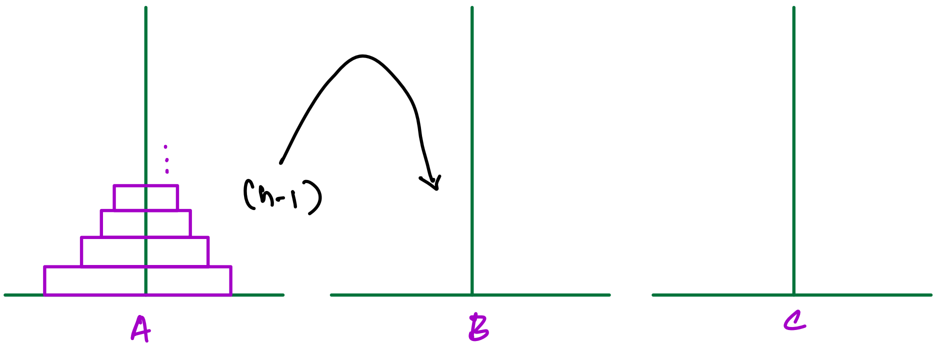
Subproblem 2

move 2 blocks

from B to C

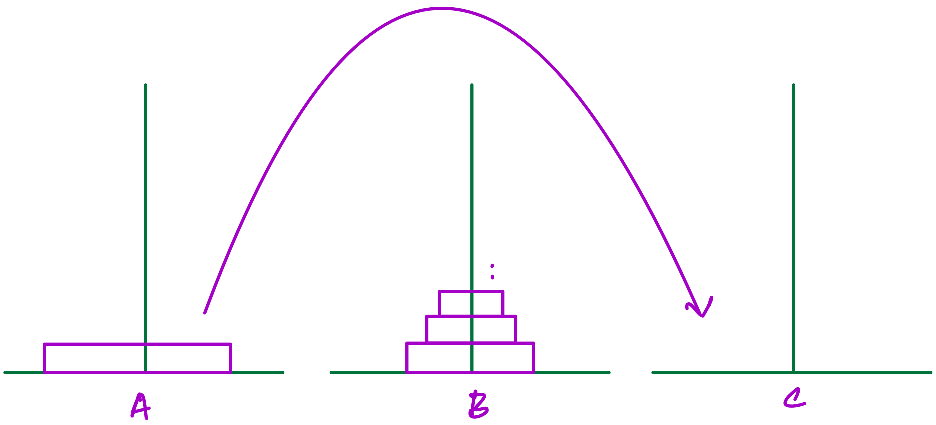
using A

Generalization for N Blocks



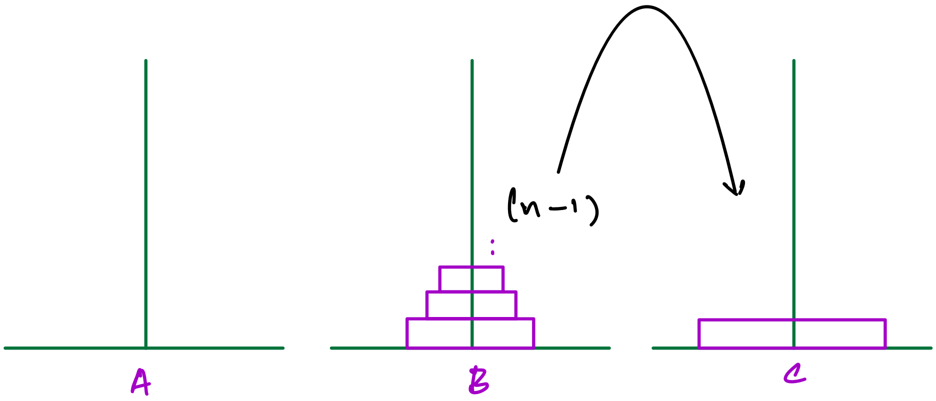
Subproblem 1

move (n-1) blocks from A to B using C



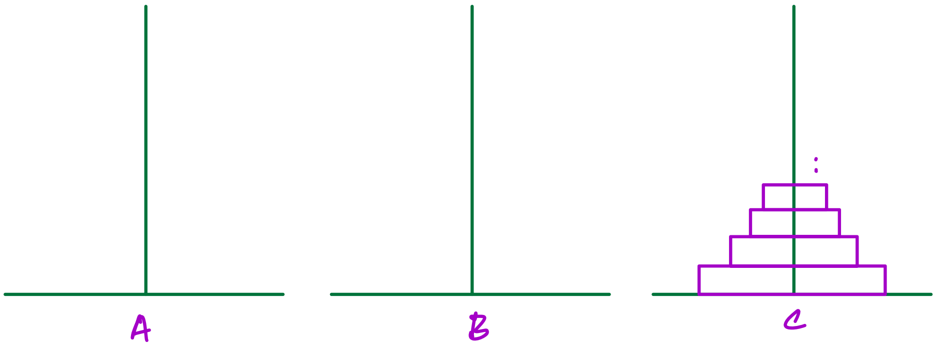
Intermediate Step

move largest block from A to C



Subproblem 2

move (n-1) blocks from B to C using a



Result

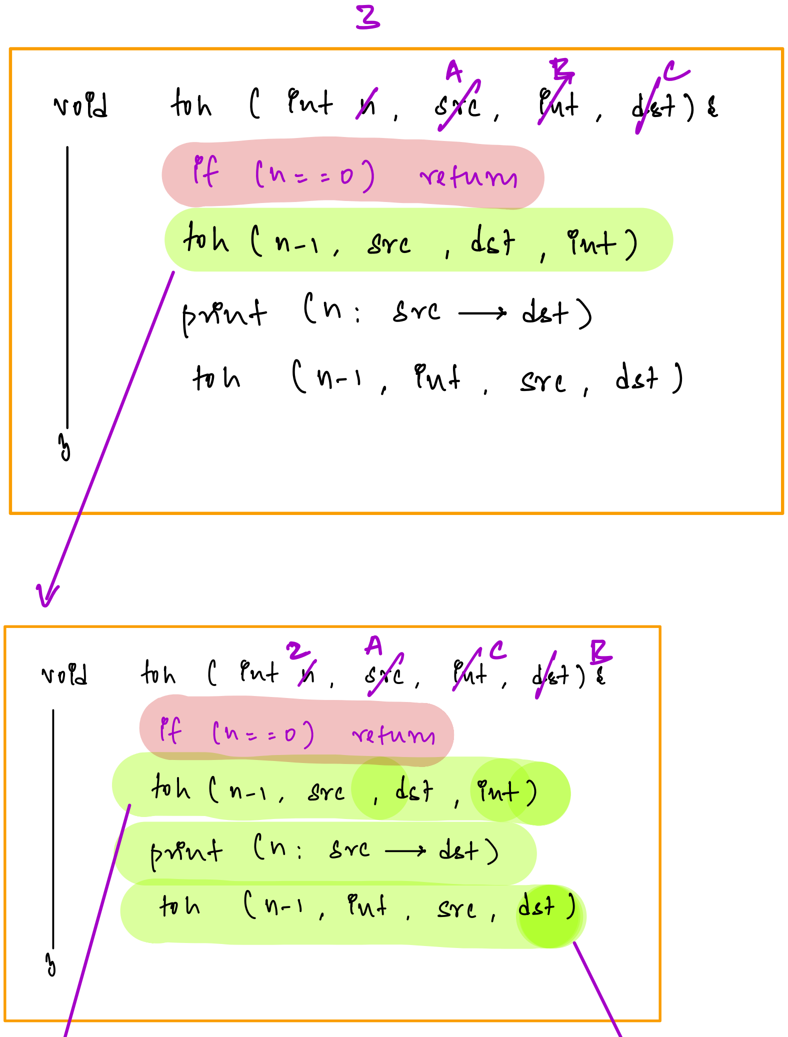
Tada !!

pseudo code

```
                                   char
void   toh ( int n , src , int , dst ) {
    if (n==0) return    // Base condition
    toh (n-1, src , dst , int)   //subproblem 1
    print (n: src → dst)   // intermediat step
    toh (n-1, int, src, dst)   // subproblem 2
}
```

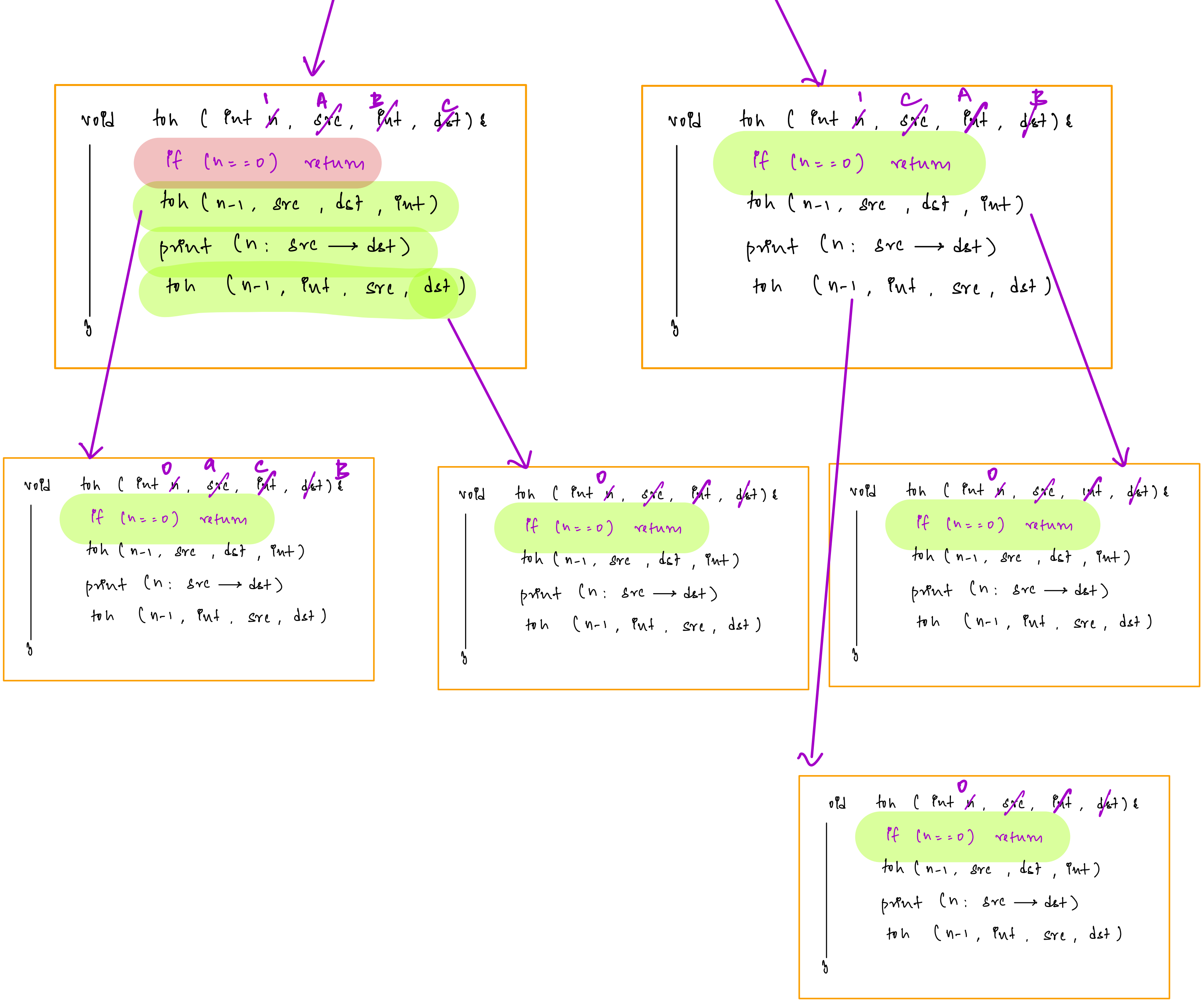
Tracing

Output : 1: A → c   2: A → B   1: c → B
3: A → c



```
void   toh ( int n , src , int , dst ) {
    if (n==0) return
    toh (n-1, src , dst , int)
    print (n: src → dst)
    toh (n-1, int, src, dst)
}
```

```
void   toh ( int n , src , int , dst ) {
    if (n==0) return
    toh (n-1, src , dst , int)
    print (n: src → dst)
    toh (n-1, int, src, dst)
}
```



```
void  toh ( int ~~n~~ 1, ~~src~~ A, ~~int~~ B, ~~dst~~ C) {
        if (n==0) return
        toh (n-1, src, dst, int)
        print (n: src → dst)
        toh (n-1, int, src, dst)
}
```

```
void  toh ( int ~~n~~ 1, ~~src~~ C, ~~int~~ A, ~~dst~~ B) {
        if (n==0) return
        toh (n-1, src, dst, int)
        print (n: src → dst)
        toh (n-1, int, src, dst)
}
```

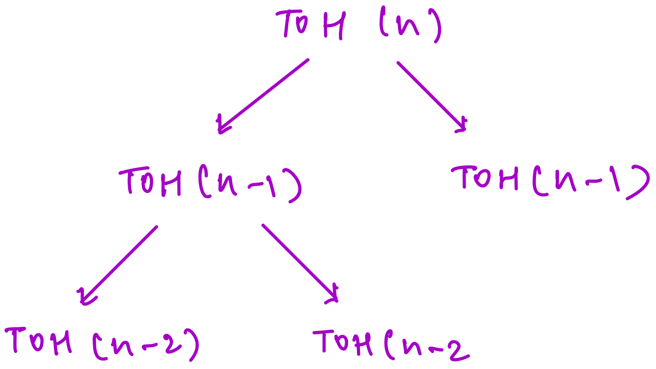
```
void  toh ( int ~~n~~ 0, ~~src~~ a, ~~int~~ C, ~~dst~~ B) {
        if (n==0) return
        toh (n-1, src, dst, int)
        print (n: src → dst)
        toh (n-1, int, src, dst)
}
```

```
void  toh ( int ~~n~~ 0, ~~src~~, ~~int~~, ~~dst~~) {
        if (n==0) return
        toh (n-1, src, dst, int)
        print (n: src → dst)
        toh (n-1, int, src, dst)
}
```

```
void  toh ( int ~~n~~ 0, ~~src~~, ~~int~~, ~~dst~~) {
        if (n==0) return
        toh (n-1, src, dst, int)
        print (n: src → dst)
        toh (n-1, int, src, dst)
}
```

```
oid  toh ( int ~~n~~ 0, ~~src~~, ~~int~~, ~~dst~~) {
        if (n==0) return
        toh (n-1, src, dst, int)
        print (n: src → dst)
        toh (n-1, int, src, dst)
}
```

Time and Space Complexity



function calls

2^n

T.C individual fun

$O(1)$

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

Print valid paranthesis

Print all valid paranthesis of length 2N given value of N.

Valid paranthesie means which has an equal no. of opening and closing paranthesis in correct order

examples

N= 1

((()   ))   )(

N= 2

(())   ()()   ((((((()   (()(   ))))   ))((

)))(

N= 3

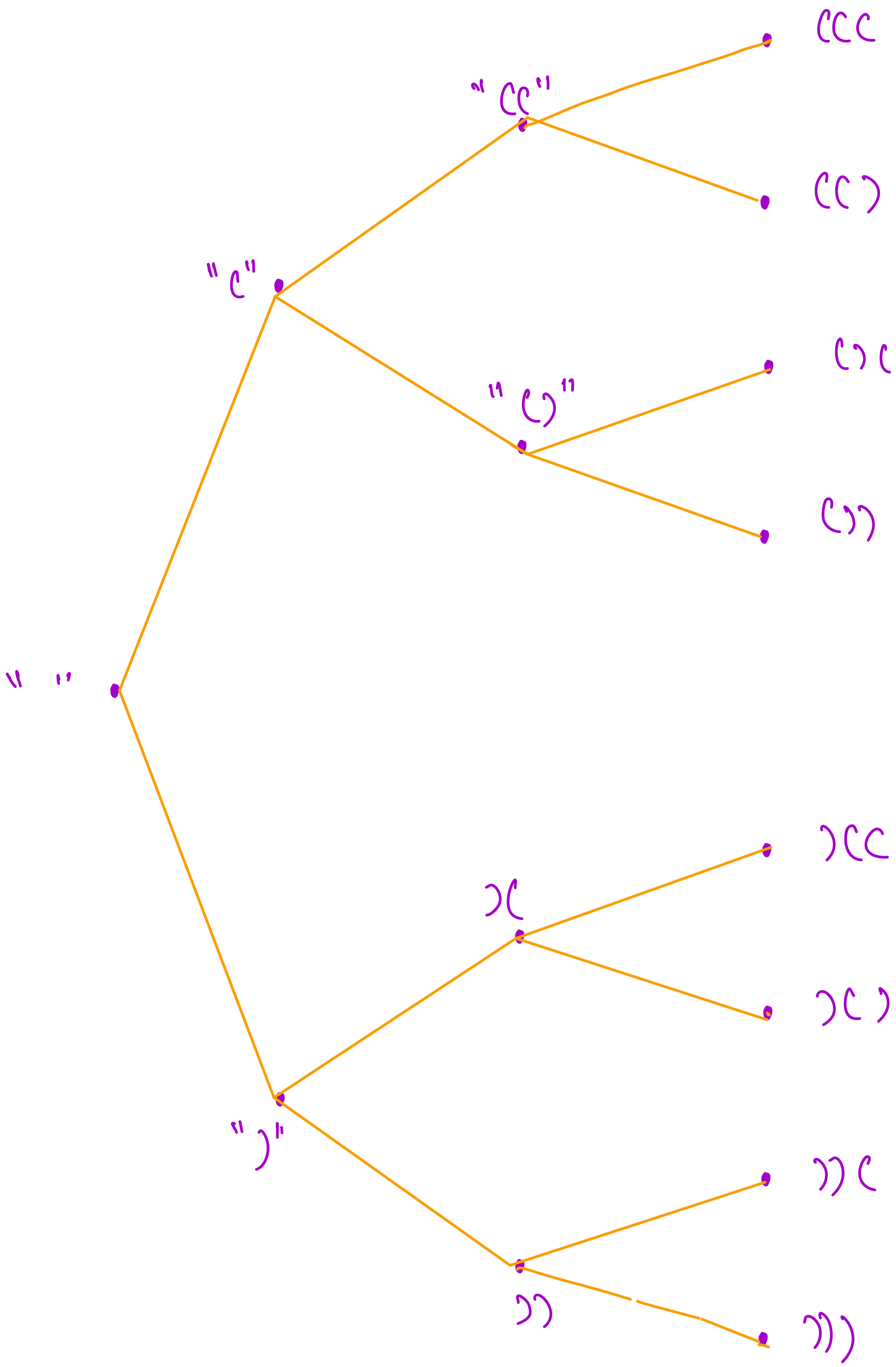
((()))   ()()()   (())()   ()(())   (()())

((((((((((()   (((())   (((()(   ))))))

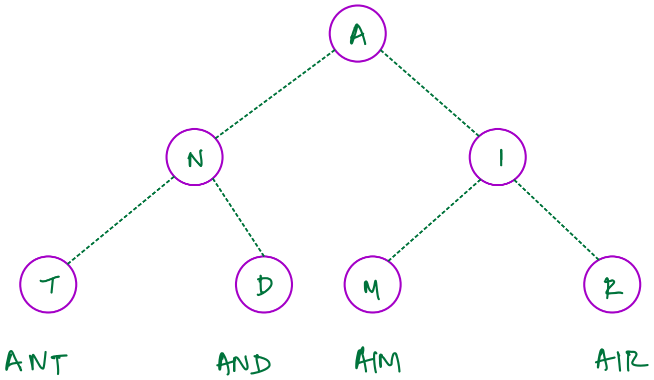
))))()

Brute force

Generate all possible paranthesis and verify if it is valid or not

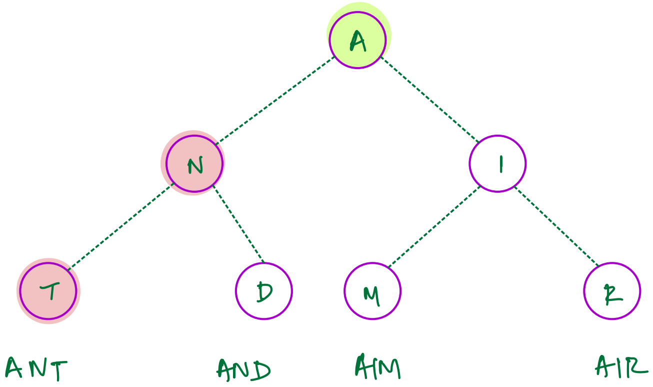


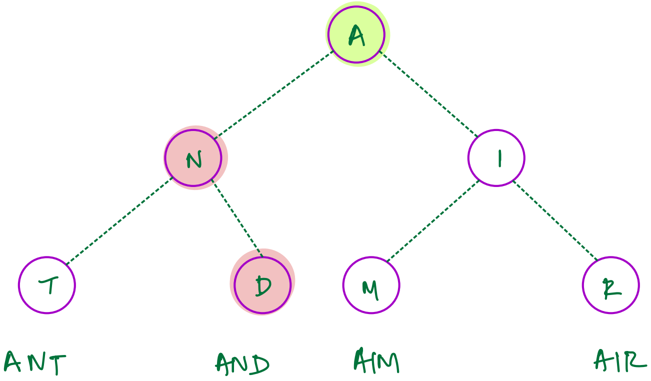
Optimization using Backtracking



I search for AIM

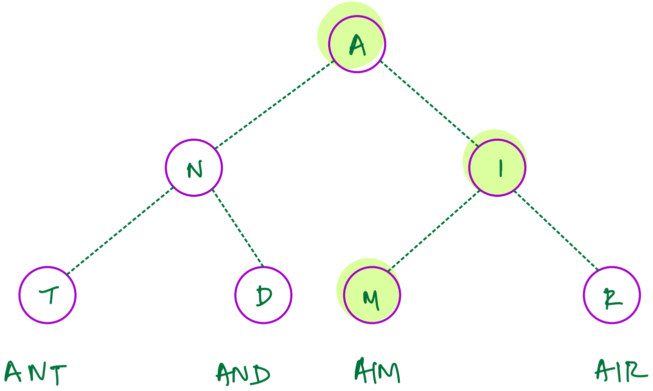
Approach 1





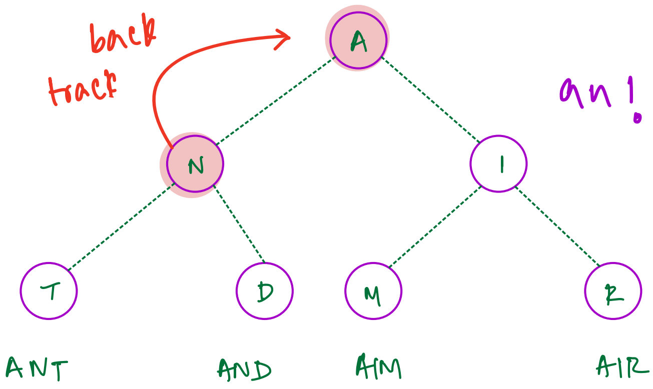
ANT == AIM

AND == AIM

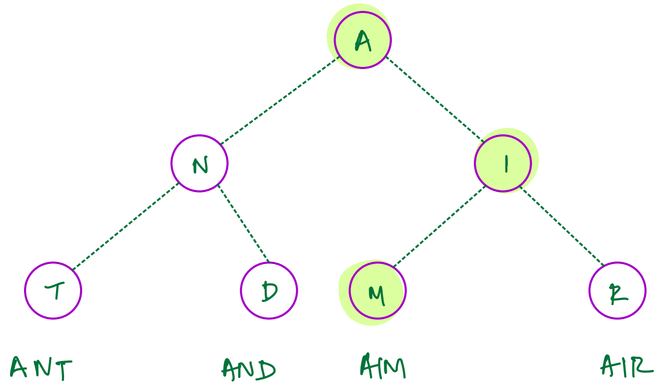


AIM == AIM

Approach 2



an != ai



pruning

Back to valid paranthesis

What is the condition for pruning ?

((()))   ()()()   (())()   ()(())   (()())

using count of open and close paranthesis at any given point

At any given point in time string is valid only when

open <= n

Closin <= open

pseudo code

```
void   solve (str, n, open, close) {

        if (len(str) == 2n)  print(str)  return;
        if (open < n)
            solve (str + "(", n, open+1, close)
        if (close < open)
            solve (str + ")", n, open, close+1)
}
```

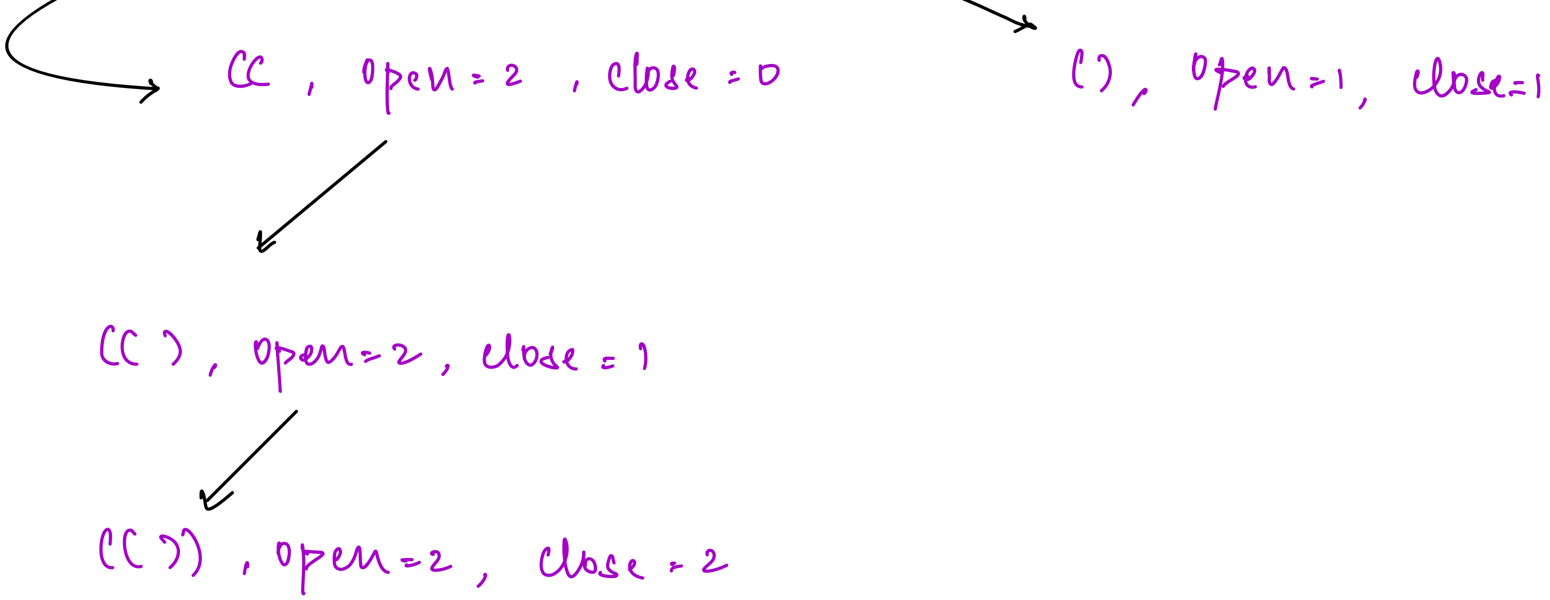
Tracing

$n = 2$

void solve (~~str~~ "", ~~n~~ 2, ~~open~~ 0, ~~close~~ 0) {

```
    if (len(str) == 2n)  print(str)  return;
    if (open < n)
        solve (str + "(", n, open+1, close)
    if (close < open)
        solve (str + ")", n, open, close+1)
}
```

(, open = 1 , close = 0



((, open = 2, close = 0

(), open = 1, close=1

(( ), open=2, close = 1

(( )), open=2, close = 2

Time Complexity

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

S.C = $O(n)$