Todays Content:

- Recursion Basics
- Towers of Hanoi
- Gray code
- 7:05 start

Recursion: Solving Problems using subproblems, called recursion

Steps:

Assumption: Decide what your function does

Mainlogic: Solving assumption using subproblems

Base Condition: When to stop

Fact(N): N >= 0

fact(3) = 3 * 2 * 1 = 6

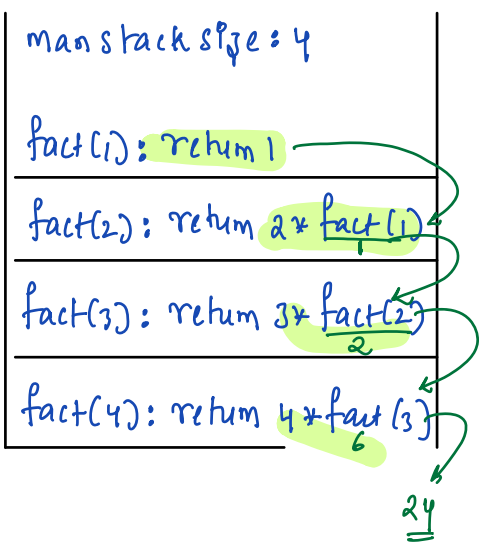
fact(5) = 5 * 4 * 3 * 2 * 1 = 120

Ass: Given N, calulate & return N!

```
___ fact(N){
    if(N<=1){return 1}
    return N * fact(N-1)
}
```

// function calls

SC: max stack size:



Recusive Relation TC:

Assume time fact(N) = f(N)

f(N) = 1 + f(N-1)

f(0) = 1, f(1) = 1

TODO

SC: O(N): We store N function in stack.

Towers of Hanoi:

→ Given 3 Towers A, B, C

→ N discs {inc order of radius} placed on Tower A

→ Move all discs from A → C using B

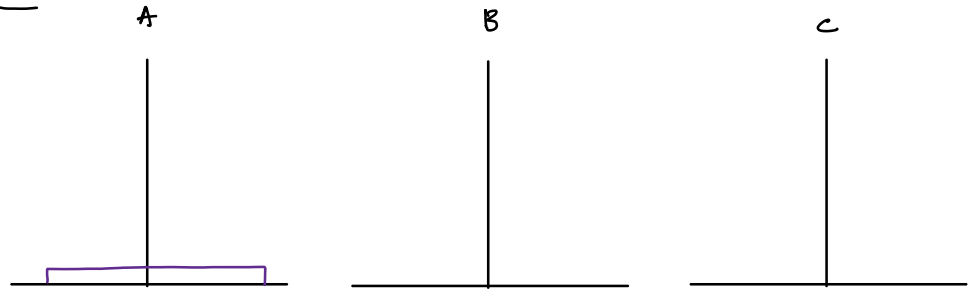
Note:

Only 1 disc can be moved at a time

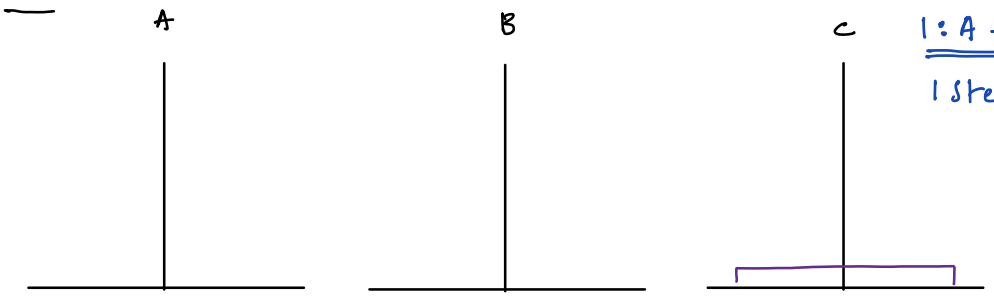
larger disc cannot be placed on a smaller disc

Q) Print movement of discs

N = 1



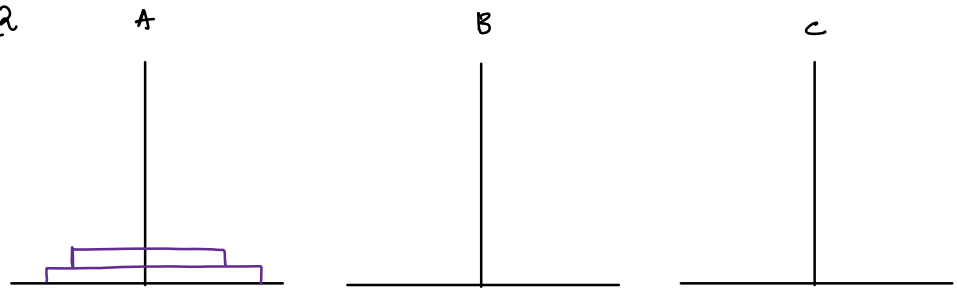
N = 1

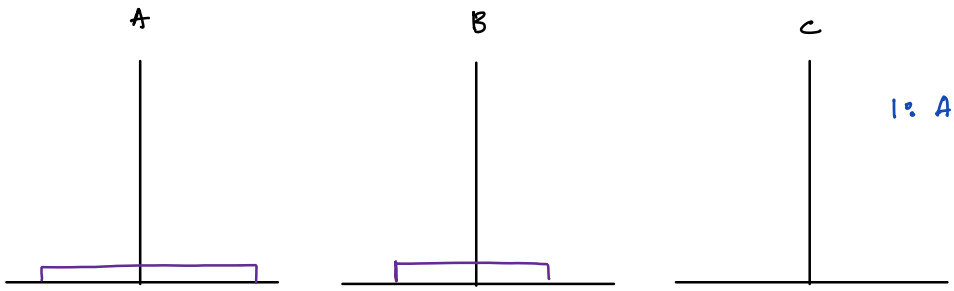


1 : A → C

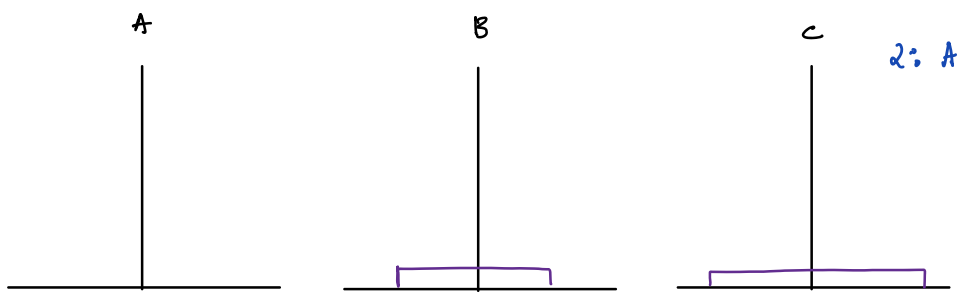
1 Step

N=2

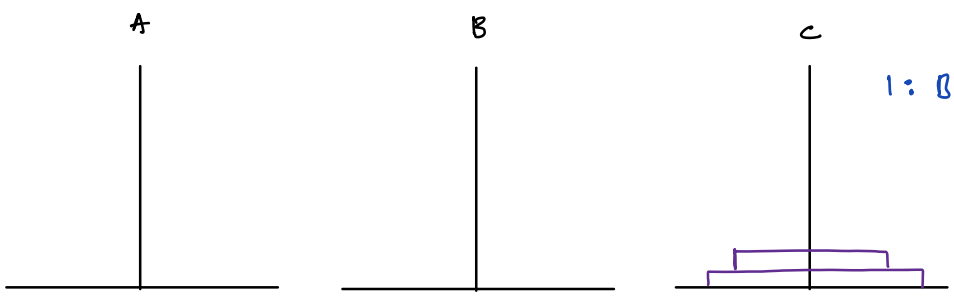




1: A → B



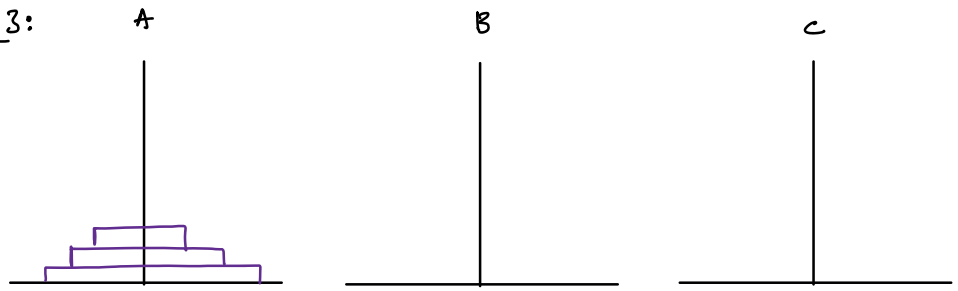
2: A → C



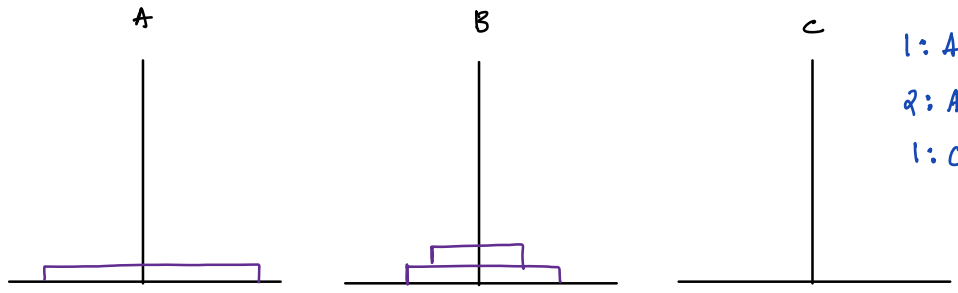
1: B → C

3 Steps

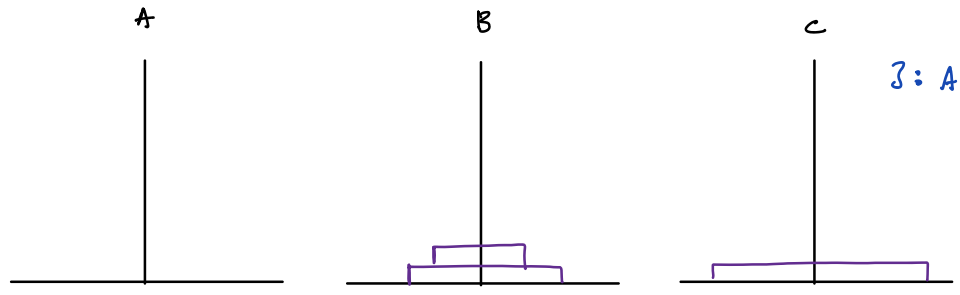
N=3:



// 1st 2 discs from A → B : {Subproblem}

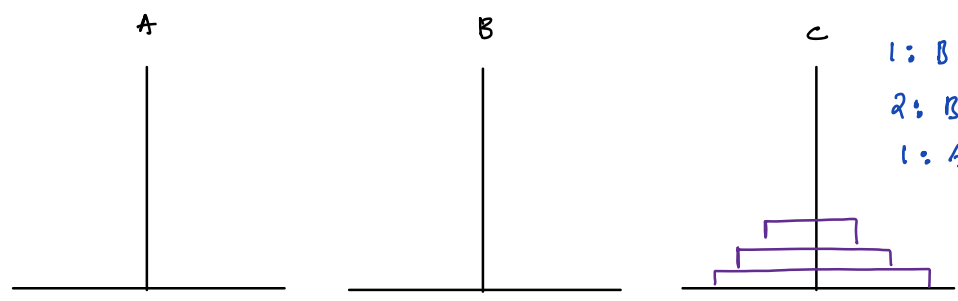


1: A → C
2: A → B
1: C → B



3: A → C

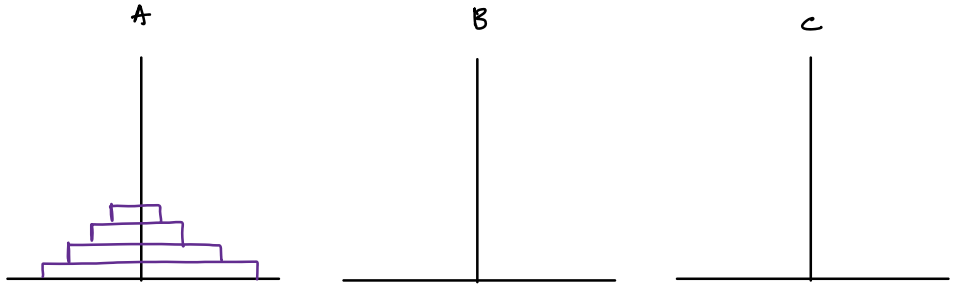
// 2 discs from B → C



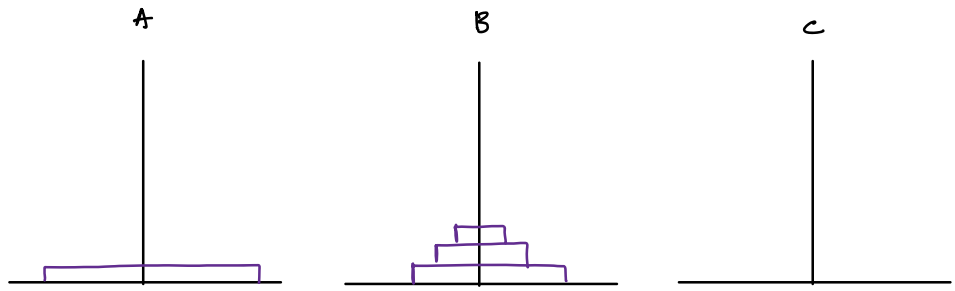
1: B → A
2: B → C
1: A → C

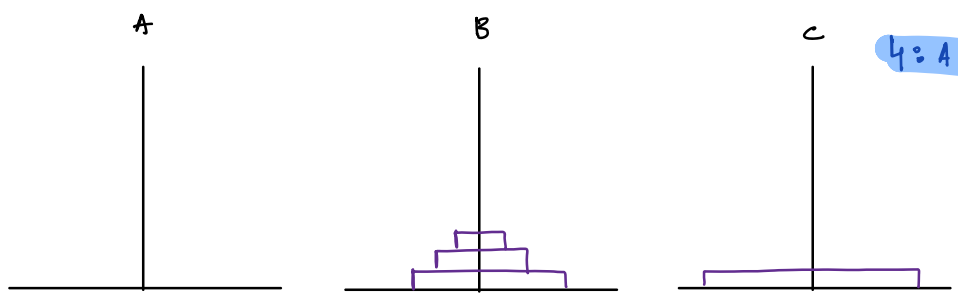
7 Steps

N=4:



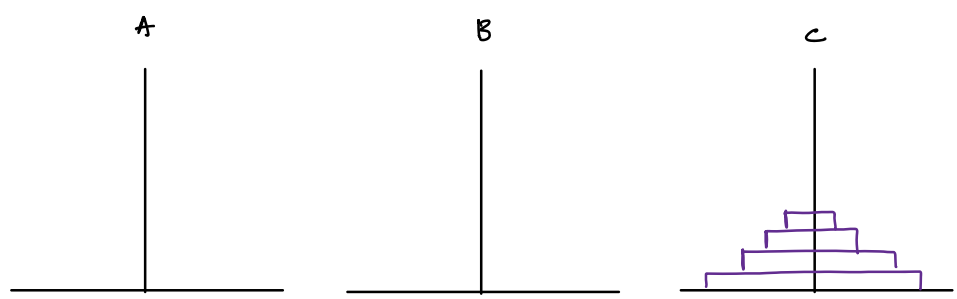
move 3 discs from A → B : 7 steps





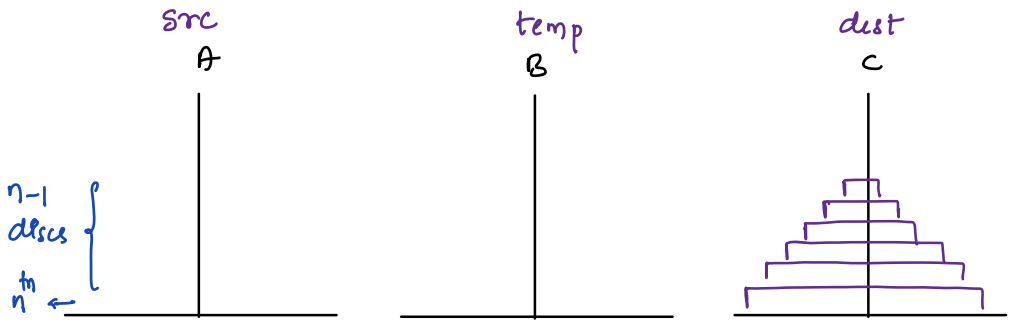
4 : A → C

move 3 discs from B → C : 7 steps



Total = 15 steps

Idea: N discs



Pseudo Code: → min no: of steps TODO / Cross Check

Ass: Print step by step movement of all N discs from S → D, using T

```
                                src     temp    dest
void TOH(int n, char S, char T, char D){
    if(n==0){return;} // nothing to print
    TOH(n-1, S, D, T) // Step by by n-1 disc from S → T using D
    print(N^th: S → D) // Move N^th disc from S → D
    TOH(n-1, T, S, D) // Step by by n-1 disc from T → D using S
}
```

Rough:

```
void TOH(n  S  T  D){
1  if(n==0){return;}
2  TOH(n-1, S, D, T)
3  print(N^th: S → D)
4  TOH(n-1, T, S, D)
}
```

line: 2 ⇒ T & D will interchange

line: 4 ⇒ S & T will interchange

Tracing:

src tmp dst

TOH(3, A B C)

n src t dest

2: TOH(2, A, C, B) ✓

n src t dest

2: TOH(1 A B C) ✓

n src t dest

2: TOH(0, A, C, B): return

3: print(1: A → C)

n src t dest

4: TOH(0, B, A, C): return

}

3: print(2: A → B)

n src t dest

4: TOH(1, C A B) ✓

2: TOH(0, C, B, A): return

3: print(1: C → B)

4: TOH(0, A, C, B): return

}

}

3: print(3: A → C)

n src t dest

4: TOH(2, B, A, C): function call trace TODO

}

Output:

1 : A → C

2 : A → B

1 : C → B

3 : A → C

N	→	Steps
1		1, 2^1-1
2		3, 2^2-1
3		7, 2^3-1
4		15, 2^4-1
N	→	2^n-1

If we print 2^n-1 steps

TC : $O(2^N)$

Recursive Relation:

Assume time taken to move N disc = $f(n)$

$f(n) = 2*f(n-1)+1$, $f(0)=1$

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

$= 2*[2f(n-2)+1]+1 \Rightarrow 4f(n-2)+3$

$= 4f(n-2)+3 \longrightarrow 2^2 f(n-2)+2^2-1$

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

$= 4[2f(n-3)+1]+3 \Rightarrow 8f(n-3)+7$

$= 8f(n-3)+7 \longrightarrow 2^3 f(n-3)+2^3-1$

$f(n-3) = 2f(n-4)+1$

$= 8[2f(n-4)+1]+7 \Rightarrow 16f(n-4)+15$

$= 16f(n-4)+15 \longrightarrow 2^4 f(n-4)+2^4-1$

After k steps:

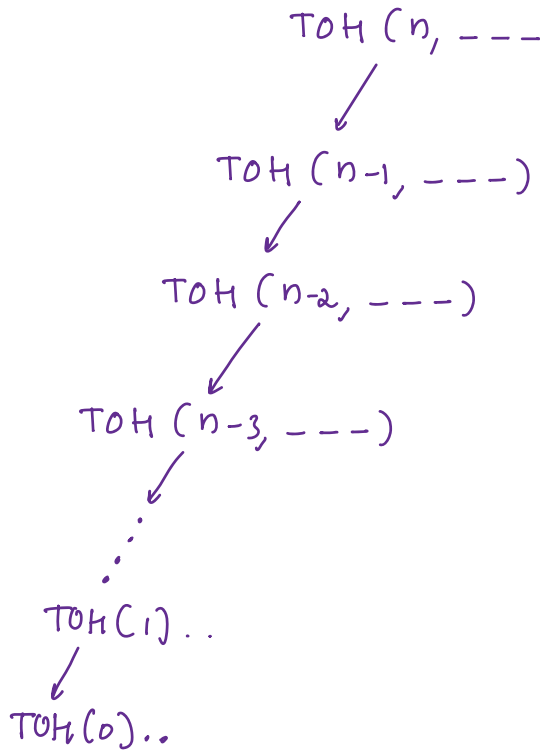
$= 2^k f(n-k) + 2^k - 1$, $f(0)=1$, # $n-k=0$, $k=n$

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

$= 2^n + 2^n - 1 = 2*2^n - 1 \Rightarrow$ TC : $O(2^n)$

SC: max stack size: N+1 ⇒ O(N)

Doubts: refer fibnacci Space complexity



Gray Code: { → }

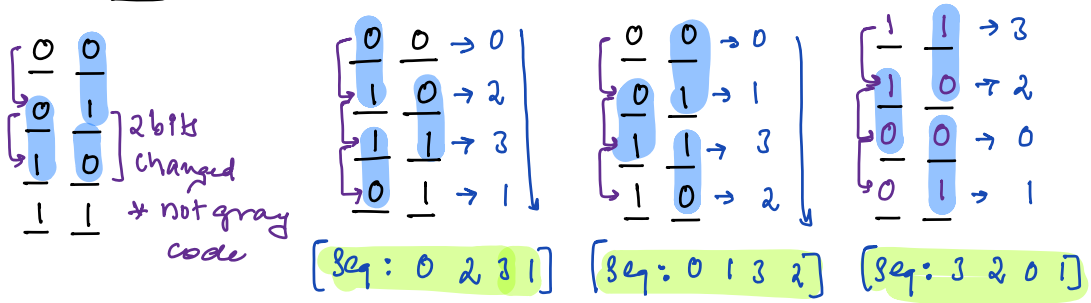
#count : 2^n elements

Given N, generate all N bits numbers

Note: Numbers in sequence should differ by Exactly 1 bit

We can return any valid sequence that works

Ex: N = 2



N = 1:

0
1

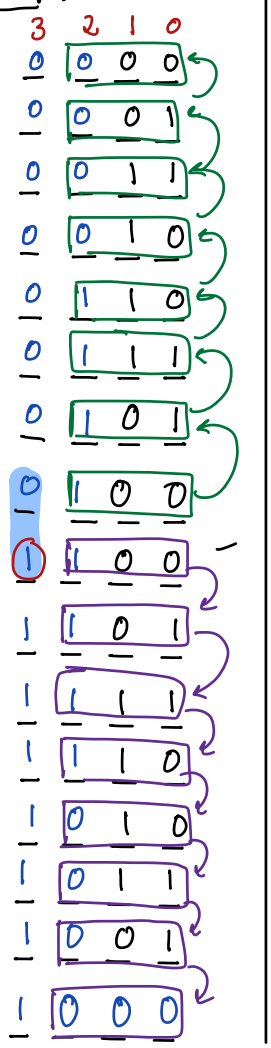
N = 2:

0 0
0 1
1 1
1 0

N = 3:

0 0 0 → 0
0 0 1 → 1
0 1 1 → 3
0 1 0 → 2
1 1 0 → 6
1 1 1 → 7
1 0 1 → 5
1 0 0 → 4

N = 4:



Ass: Given N, return N bit gray code seq

```
                                          n>=0
list<int> grayCode(n){
    if(n==1){ list b; b.insert(0), b.insert(1) return b}
    list sb = grayCode(n-1); // gray code of n-1 bits
    list ans; // gray code of n bits
    i=0; i< sb.size(); i++){ // add 0 at n-1th bit,
        ans.insert(sb[i])        decimal value won't change
    }
    i = sb.size()-1; i>=0; i--){
        // set n-1th bit pos for an number
        ans.insert(sb[i] + 1<<(n-1))
                           2^(n-1)
    }
    return ans;
}
```

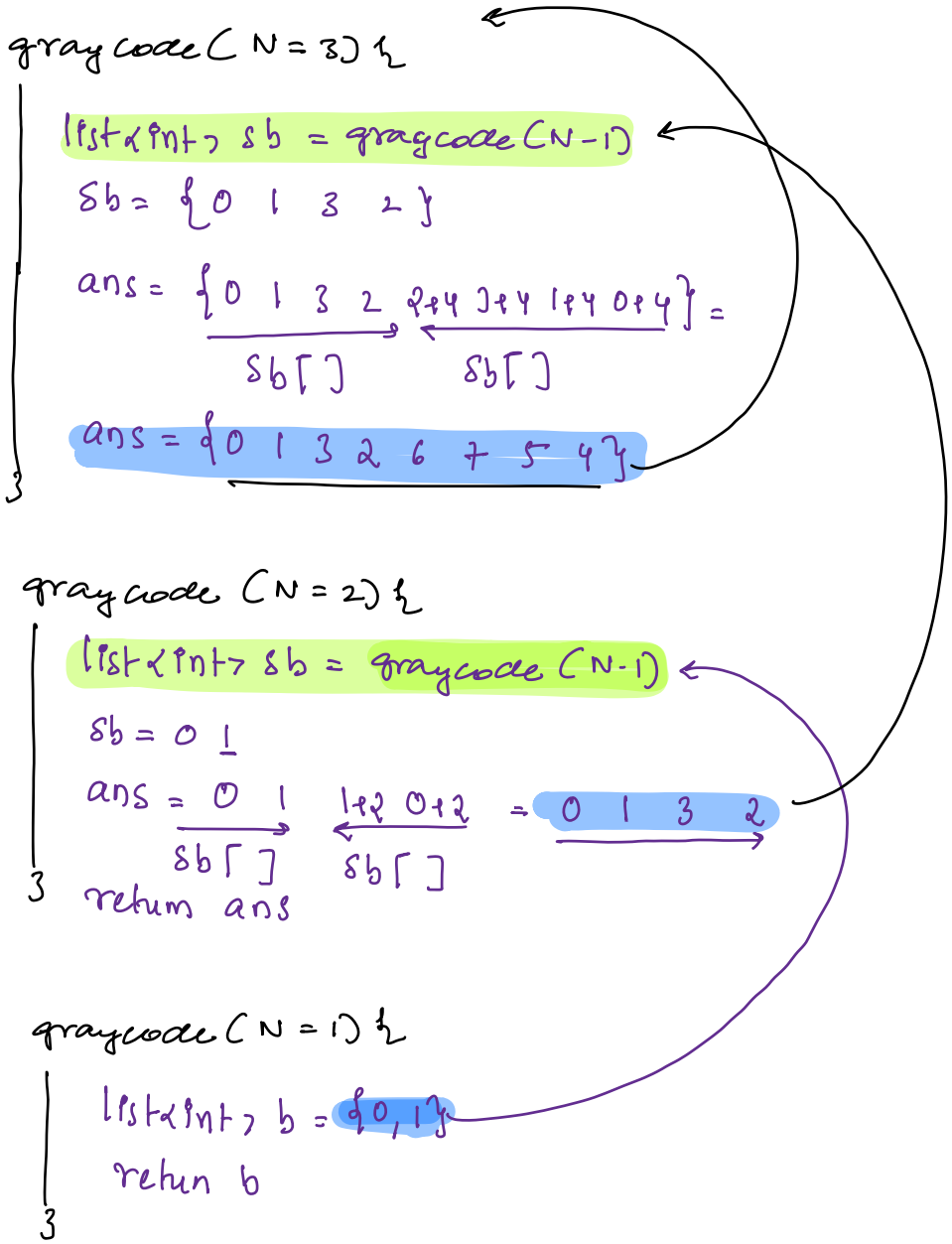
// Gray code : n bits → n-1 bits → n-2 bits → ---- 3 → 2 → 1

↳ TC : $2^n + 2^{n-1} + 2^{n-2} + \dots 2^3 + 2^2 + 2^1 \approx 2^{n+1} - 2 \approx O(2^N)$

SC : $O(2^n)$ + recursive stack size)

$O(2^n)$ ↳ for gray code seq

↳ $O(N)$


grayCode(N=3){
List<Int> sb = grayCode(N-1)
sb = {0 1 3 2}
ans = {0 1 3 2 2+4 3+4 1+4 0+4} =
sb[] sb[]
ans = {0 1 3 2 6 7 5 4}
}
grayCode(N=2){
List<Int> sb = grayCode(N-1)
sb = 0 1
ans = 0 1 1+2 0+2 = 0 1 3 2
sb[] sb[]
return ans
}
grayCode(N=1){
List<Int> b = {0,1}
return b
}