Todays Content:

- Recursion Basics
- Towers of Hanoi
- Gray code
- 9:05 PM

Recursion: Solving a Problems using SubProblems

Steps:

Assumption: Decide what your function does

Mainlogic: Solving assumption using subproblems

Base Condition: When should recursion end

Fact(N): $N >= 0$

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

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

Ass: Given N, calculate & return N!

```
fact(N){        TC: O(N)
                SC: O(N)
    if( N<=1) {return 1}

    return N * fact(N-1)
                 f(n-1)
}
```

SC: max stack size

fact(1) = return 1
fact(2) = 2 * fact(1)
fact(3) = 3 * fact(2)
fact(4) = 4 * fact(3)

fact(4) 24

Recursive Relation TC:

Assume time taken to calulate fact(N) = f(n)

$f(n) = f(n-1) + 1 \quad T(1) = 1$

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

$= f(n-2) + 2$

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

$= f(n-3) + 3$

//generalized $f(n-k) + k \quad f(1) = 1 \} \; n-k = 1, \; k = n-1$

$f(1) + n - 1 \Rightarrow \boxed{O(N) = TC}$

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

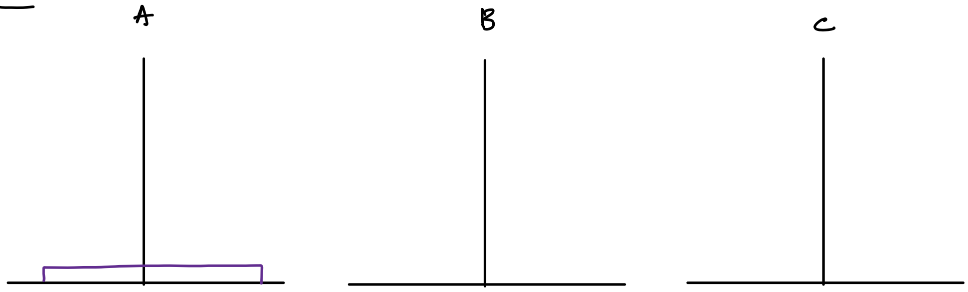
Note:

Only 1 disc can be moved at a time

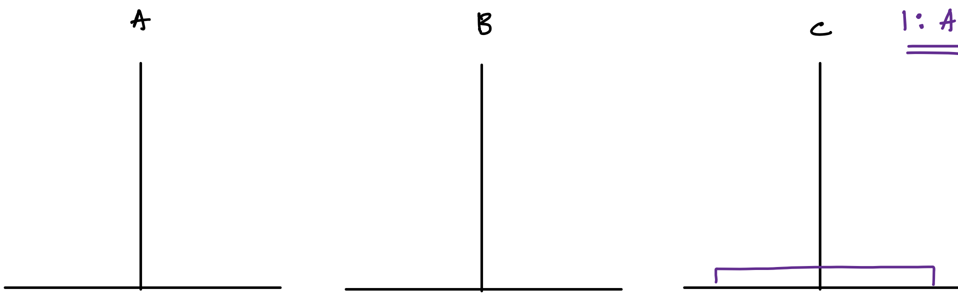
larger disc cannot be placed on a smaller disc

Q) Print movement of discs

N = 1



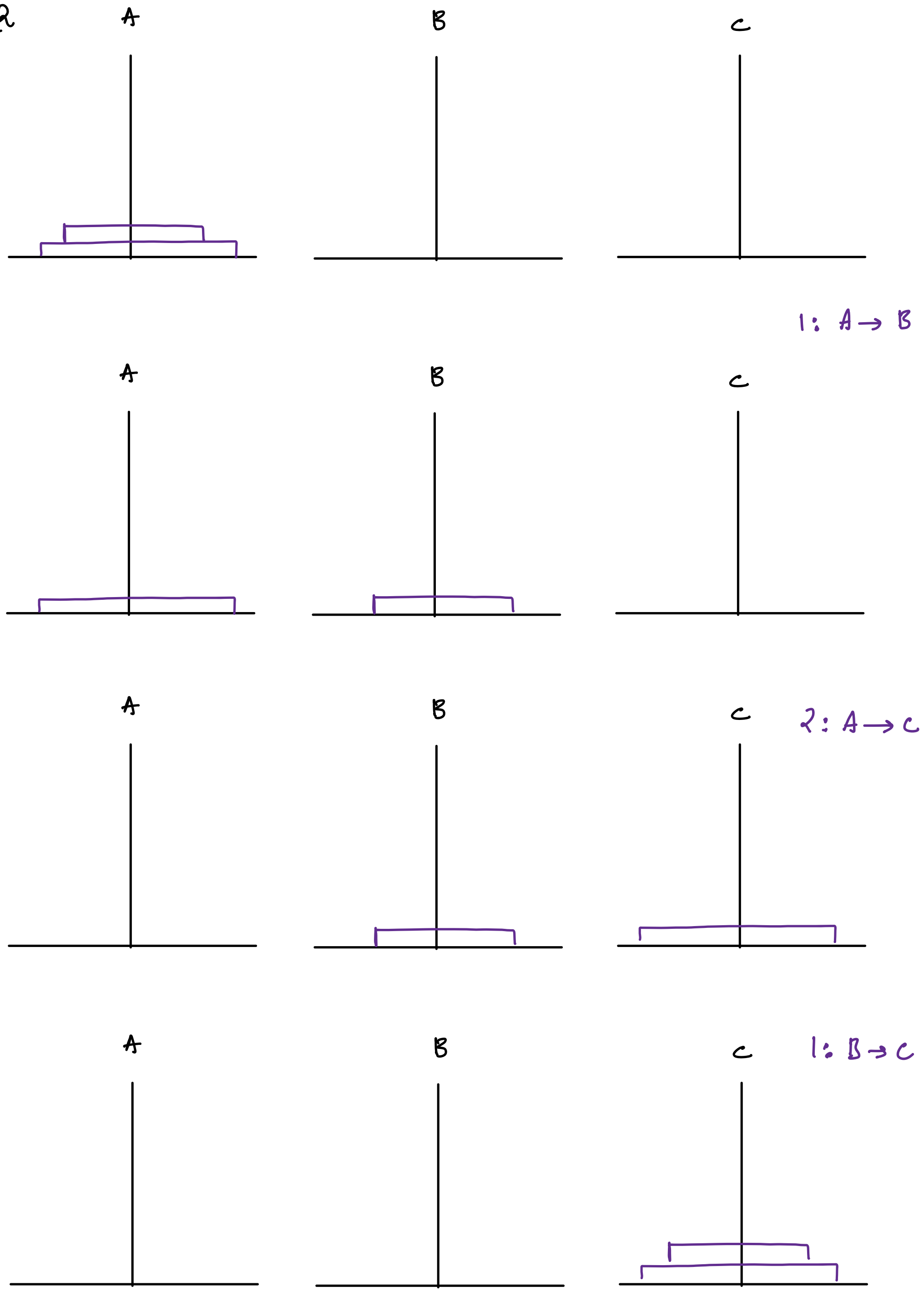
N = 1



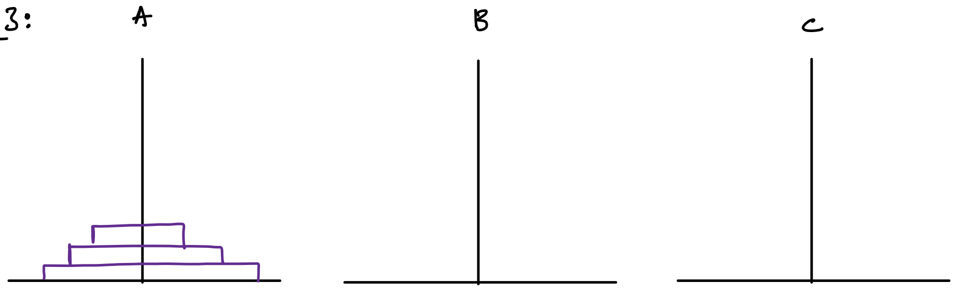
Output:

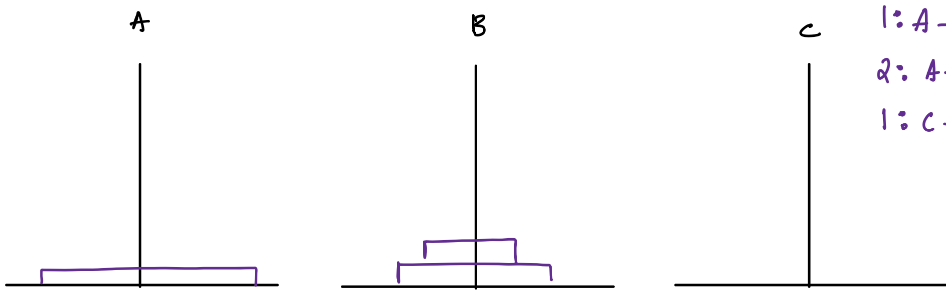
1: A → C

N=2

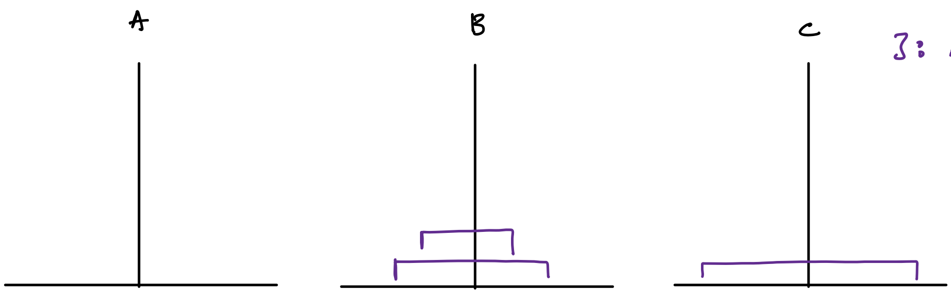


N=3:



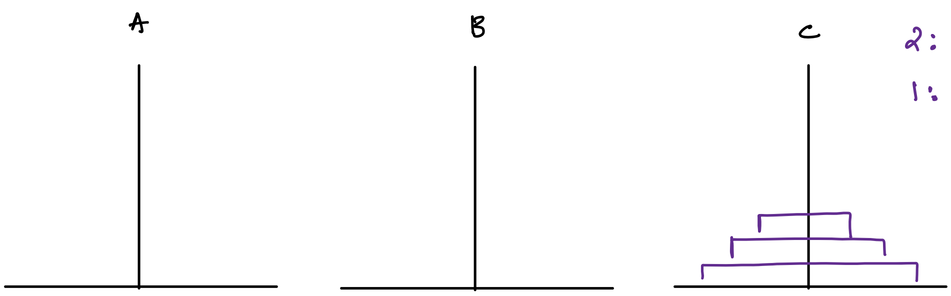


1: A → C
2: A → B
1: C → B

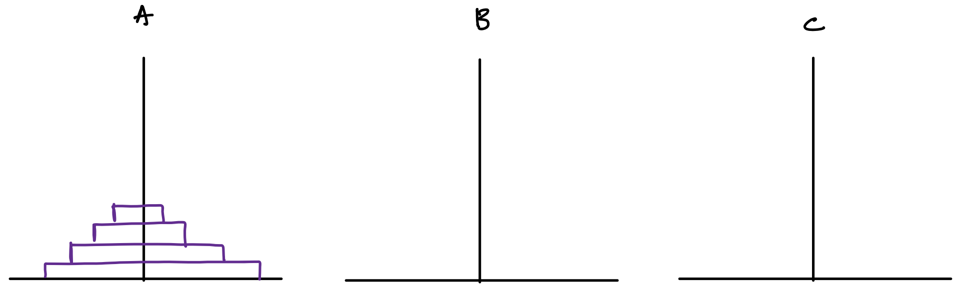


3: A → C

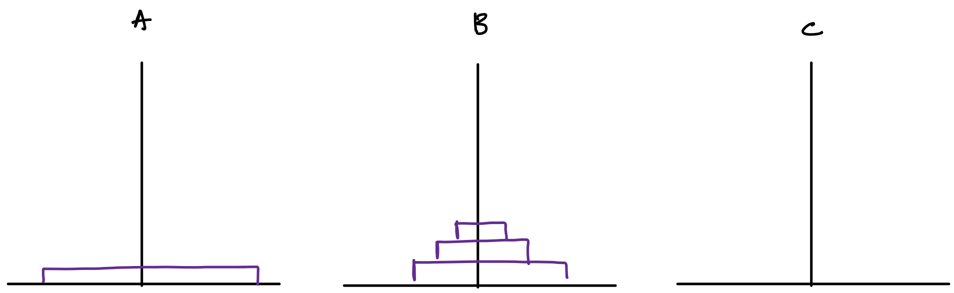
1: B → A
2: B → C
1: A → C



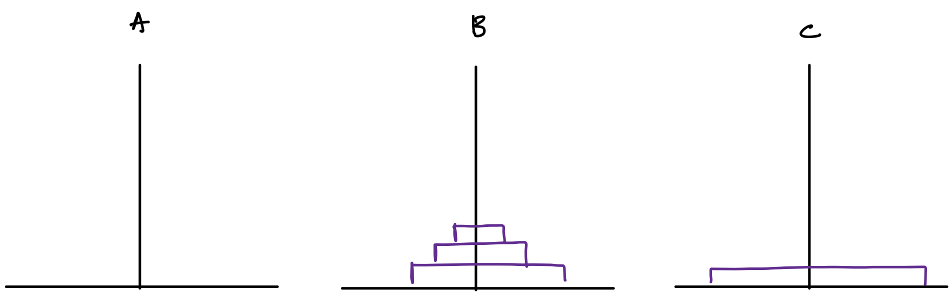
N=4:



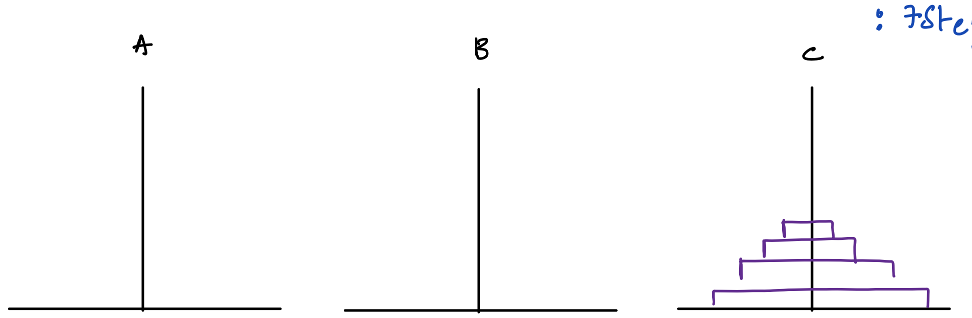
// move 3 discs from A → B
: 7 steps



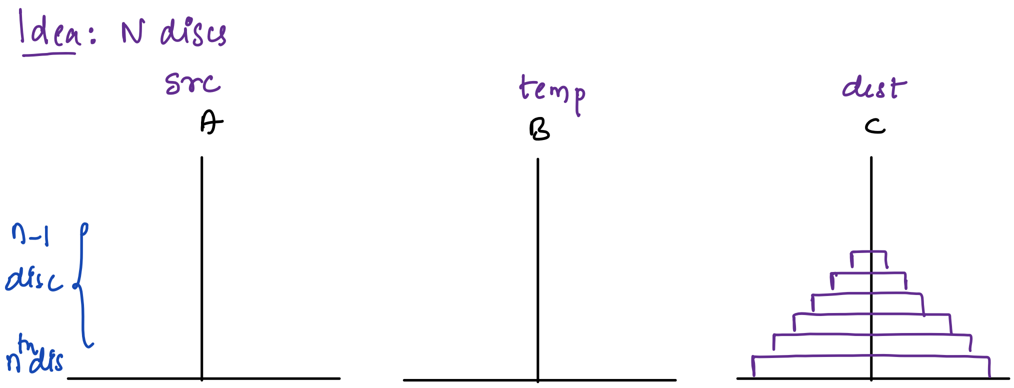
4th : A → C



// move 3 disc from B → C
: 7steps



Idea: N discs



Pseudo Code:

Ass: Given N, move all N discs from A → C step by step

```
                        src       temp      dest
void TOH(int n, char S, char T, char D){
                        A         B         C
    if(n==0){return}
               src temp dest
    TOH(n-1, S,   D,   T) // move n-1 discs from A → B Step by Step
    print(nth: S → D) // move nth disc from A → C
              src temp dest
    TOH(n-1, T,  S,   D) // move n-1 discs from B → C Step by Step
}
```

Rough:

```
void TOH(n S T D){
1 if(n==0){return}
2 TOH(n-1, S, D, T) // line 2 T & D are changing
3 print(Nth: S → D)
4 TOH(n-1, T, S, D) // line 4 S & T are changing
}
```

Tracing:

n src tmp dst

TOH(3 A B C) ~~1~~ ~~2~~ ~~3~~ ~~4~~

    n src tmp dst

    2: TOH(2 a c b) : ~~1~~ ~~2~~ ~~3~~ ~~4~~

        n src T D

        2: TOH(1, a, b, c) : ~~1~~ ~~2~~ ~~3~~ ~~4~~

            2: TOH(0, a, c b) : return

            3: print(1: a → c)

            4: TOH(0, b, a, c) : return

        }

        print(2 : a → b)

        n S T D

        4: TOH(1, c, a, b) : ~~1~~ ~~2~~ ~~3~~ ~~4~~

            2: TOH(0, c, b, a) return

            print(1: c → b)

            4: TOH (0, a, c, b) · return

        }

    }

    3 : print(3 : A → C)

    n S T D

    4 : TOH(2, B, A, C) ~~1~~ 2 3 4

}

N	→ Steps
1	$1 \Rightarrow 2^1 - 1$
2	$3 \Rightarrow 2^2 - 1$
3	$7 \Rightarrow 2^3 - 1$
4	$15 \Rightarrow 2^4 - 1$

// N disc : Steps $2^N - 1$ Steps

Recursive Relation:

Assume time taken to move N disc

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

SC: max stack size = N+1

↳ SC: O(N)

$f(n) = 2f(n-1) + 1 \quad f(0) = 1$

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

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

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

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

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

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

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

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

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

After k Substitutions

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

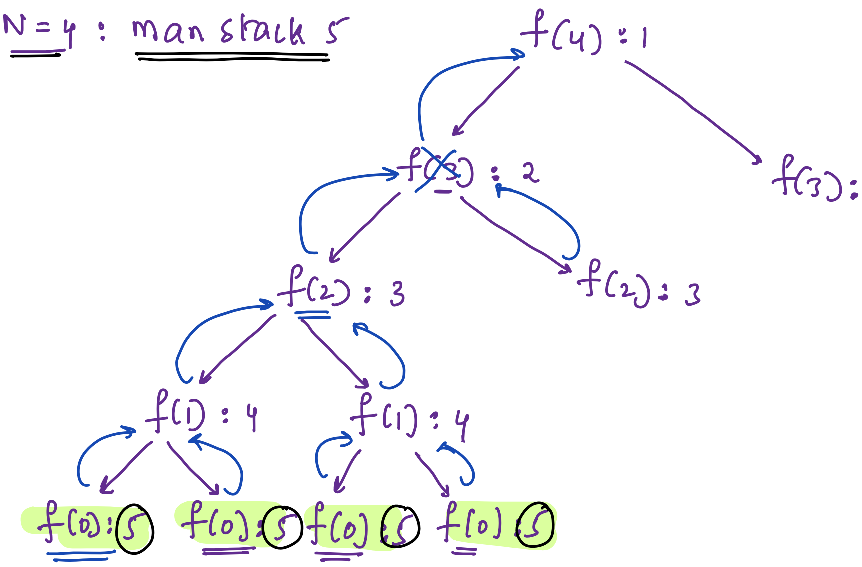
// $n - k = 0 \Rightarrow k = n$

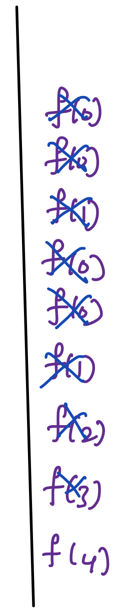
$= 2^n f(0) + 2^n - 1$

$= 2^n + 2^n - 1 \Rightarrow 2 * 2^n - 1 \Rightarrow$

TC: $O(2^n)$

N = 4 : max stack 5





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

0 0 → 0
0 1 → 1
1 1 → 3
1 0 → 2

Seq : 0 1 3 2

N = 2

1 0 → 2
1 1 → 3
0 1 → 1
0 0 → 0

Seq : 2 3 1 0

N = 2

0 0
0 1
1 0 *
1 1

N = 2

0 0 → 0
1 0 → 2
1 1 → 3
0 1 → 1

Seq : 0 2 3 1

Input :

N = 1

2^0	
0	→ 0
1	→ 1

N = 2

2^1	2^0	
0	0	→ 0
0	1	→ 1
1	1	→ 3
1	0	→ 2

N = 3

2^2	2^1	2^0	
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

2^3	2^2	2^1	2^0	
0	0	0	0	→ 0
0	0	0	1	→ 1
0	0	1	1	→ 3
0	0	1	0	→ 2
0	1	1	0	→ 6
0	1	1	1	→ 7
0	1	0	1	→ 5
0	1	0	0	→ 4
1	1	0	0	→ 8 + 4
1	1	0	1	→ 8 + 5
1	1	1	1	→ 8 + 7
1	1	1	0	→ 8 + 6

Ass: Given N, return all N bit numbers in gray code seq

TC: O(2^N)

1	0	1	0	→ 8 + 2
1	0	1	1	→ 8 + 3
1	0	0	1	→ 8 + 1
1	0	0	0	→ 8 + 0

```
list<int> graycode(N){        SC: O(2^N + N)
                                         Recursive stacksize
    if(n==1) { list<int> b;
               b.insert(0) b.insert(1)
               return b }

    // list of n-1 bits gray code seq
    list<int> sq = graycode(n-1)      : f(n-1)
    list<int> ans;
    int x = sq.size() // x = 2^(N-1) //
    i=0; i<x; i++){
        ans.insert(sq[i])
    }                                  } x
    i = x-1; i>=0; i--){
        ans.insert( sq[i] + 1<<(n-1))      1<<(n-1) = 2^(n-1)
    }                                  } x
    return ans;
}
```

$2x$ iterations

$2 * 2^{N-1} \Rightarrow 2^N$

// Assume time taken to calculate N bit gray code = f(n)

$f(n) = f(n-1) + 2^n \qquad f(1) = 2$

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

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

$f(n-2) = f(n-3) + 2^{n-2}$

$= f(n-3) + 2^{n-2} + 2^{n-1} + 2^n$

$f(n-3) = f(n-4) + 2^{n-3}$

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

$\downarrow$

$\downarrow$

$\downarrow$

$= f(1) + 2^2 + 2^3 + 2^4 + 2^5 \ldots\ldots 2^n$

$= 2^1 + 2^2 + 2^3 + 2^4 + 2^5 \ldots\ldots 2^n$

G.P:

$a = 2,\ r = 2,\ t = n$

$\Rightarrow \frac{a * \{r^t - 1\}}{r-1} \Rightarrow \frac{2 * \{2^n - 1\}}{2-1} \Rightarrow \frac{2 * [2^n - 1]}{1}$

$\approx O(2^n)$

Stack Size:

gracode(n)

$\searrow$ graycode(n-1)

$\searrow$ graycode(n-2)

$\searrow$ graycode(n-3) ..... graycode(1) 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

}