Today's content

→ Recursion basics ↷ function calling itself : solving a problem using smaller instance of the same problem. (sub-problem)

→ fact() / power() / fib()

→ Recursive relations

→ Gray code.

Recursion steps

① Assumption → Decide what should our function do, Assume it does.

② Main logic → solving assumption using sub-problems.

③ Base condition → When should the function terminate.

Factorial

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

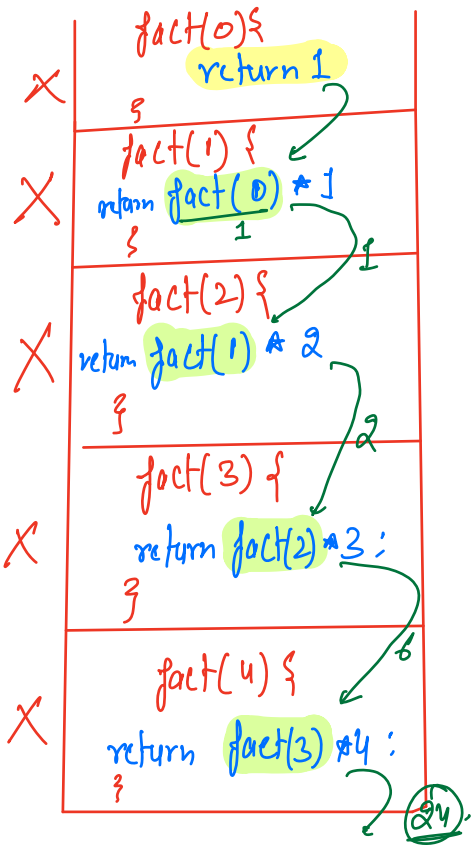
fact(6) : 720.

{ N! = (N-1)! * N. }

```
int fact ( N ) { //Assumption - Given N, calculate & return N!
    if ( N == 0 ) { return 1 }
    return fact(N-1) * N ;
}
```

T(N) = 1 + T(N-1)

{T(0) = 1}



Time Complexity

$T(N) = T(N-1) + 1$

$T(N-1) = T(N-2) + 1$

$T(N) = T(N-2) + 2$

$T(N-2) = T(N-3) + 1$

$T(N) = T(N-3) + 3$

After - K steps

$T(N) = T(N-K) + K$

$N-K = 0$

$\Rightarrow K = N$

// we know that, $T(0) = 1$

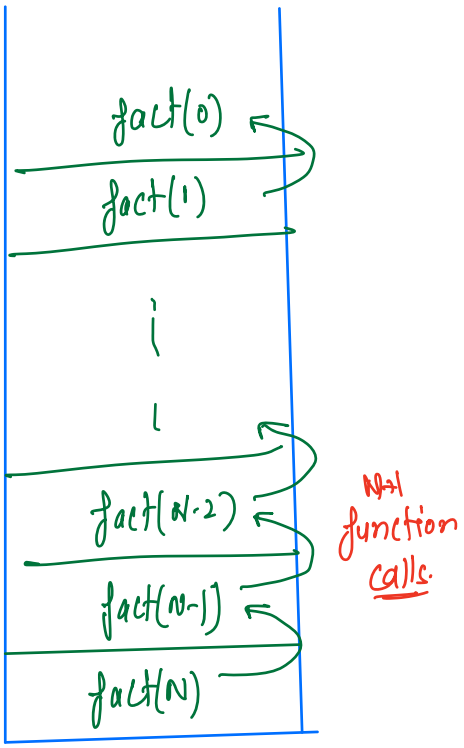
$T(N) = T(0) + N$

$T(N) = 1 + N$

{T.C → O(N)}

Space Complexity

↳ max size of stack.



S.C → O(N)

Fibonacci Series

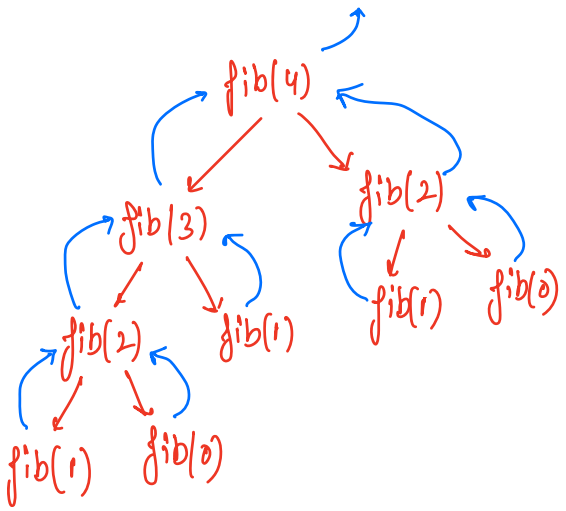
	0	1	2	3	4	5	6	7	8	9	10
fib →	0	1	1	2	3	5	8	13	21	34	55

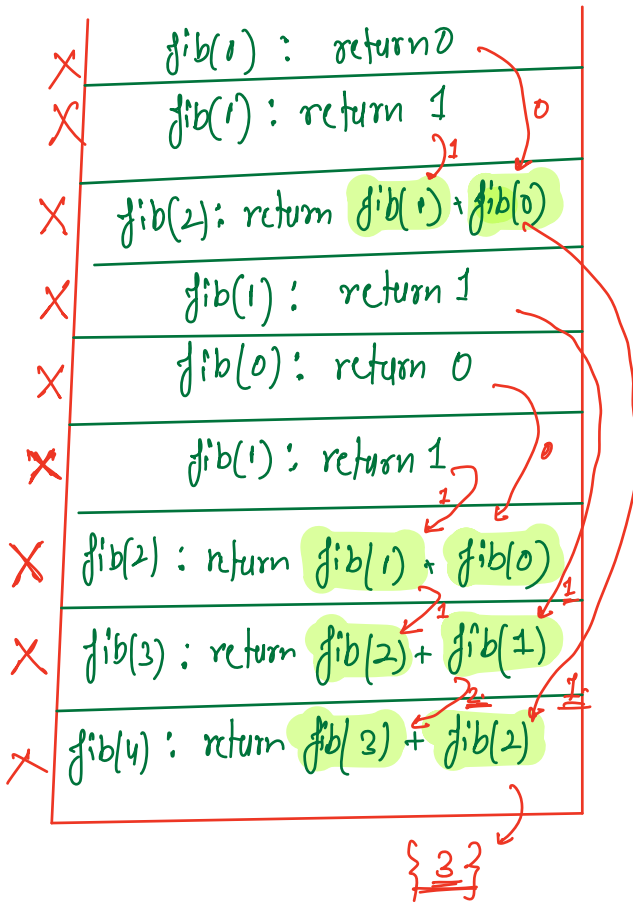
// N^{th} fibonacci = $(N-1)^{th}$ fibonacci + $(N-2)^{th}$ fibonacci

// Assm → Given N, calc. & return N^{th} fibonacci no.

```
int fib (N) {
    if ( N <= 1) { return N }
    return fib(N-1) + fib(N-2)
}
```

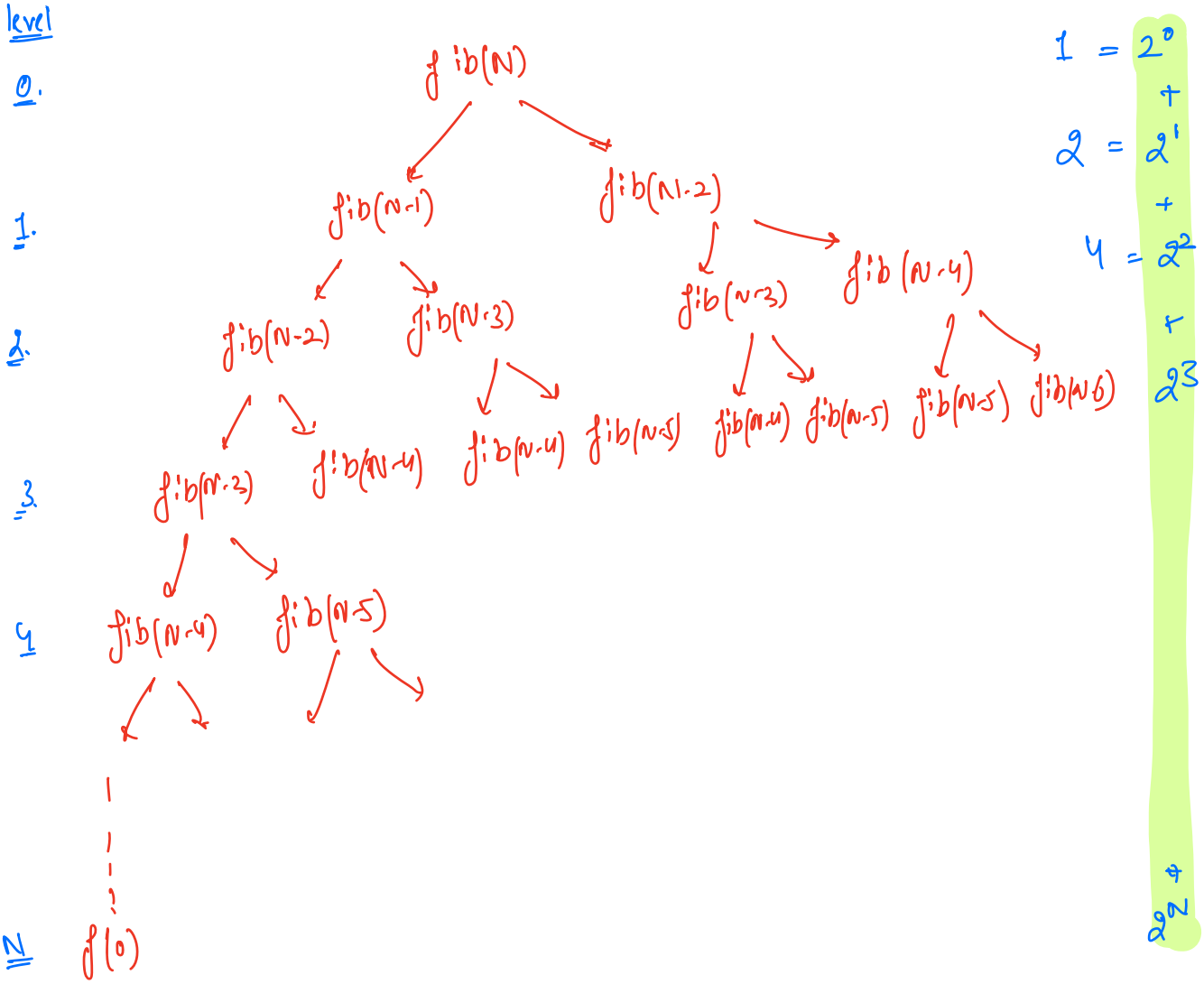
$$T(N) = 1 + T(N-1) + T(N-2)$$





Time Complexity

$T(N) = T(N-1) + T(N-2) + 1$



T.C → $1 + 2^1 + 2^2 + 2^3 + \cdots + 2^N$

[$a = 1$, $r = 2$, no. of terms → $n+1$]

$$\text{Sum} = \frac{a(r^{\text{no. of terms}} - 1)}{(r-1)} = \frac{1[2^{n+1} - 1]}{(2-1)} = 2^{n+1} - 1$$

$$= 2 \cdot 2^n - 1$$

T.C → $O(2^N)$

S.C → $O(N)$

Power function

Q) Given a, N. Calculate a^N.

$a = 2, \; N = 5 \longrightarrow 2^5 \rightarrow 32$

$a = 3, \; N = 4 \longrightarrow 3^4 \rightarrow 81$

Assm: Given a, N. Calculate & return a^N.

Main logic:

$$a^{10} = a^9 * a$$
$$a^{14} = a^{13} * a$$
$$\Big] \; a^N = a^{N-1} * a$$

$$a^{10} = a^5 * a^5$$
$$a^{14} = a^7 * a^7$$
$$a^{15} = a^7 * a^7 * a$$
$$a^{21} = a^{10} * a^{10} * a$$

$$\Big] \; \text{if } (N\%2 == 0) \; a^N = a^{N/2} * a^{N/2}$$
$$\text{else } a^N = a^{N/2} * a^{N/2} * a$$

```
int pow ( a, N ) {
    if (a == 0) { return 0 }
    if (N == 0) { return 1 }
    p = pow (a, N/2);
    if ( N % 2 == 0) {
        return p * p
    }
    else {
        return p * p * a
    }
}
```

$$\Rightarrow \{ T(0) = 1 \}$$
$$\Rightarrow \{ T(1) = 1 \}$$

$$T(N) = 1 + T(N/2)$$

pow(a, N)
↓
pow(a, N/2)
↓
pow(a, N/4)
↓
pow(a, N/8)
↓
.
.
.
pow(a, 1)

[S.C → O(log₂N)]

Time Complexity.

$T(N) = T(N/2) + 1$

$T(N/2) = T(N/4) + 1$

$T(N) = T(N/4) + 2$

$T(N/4) = T(N/8) + 1$

$T(N) = T(N/8) + 3$

// generalisation

$T(N) = T(N/2^K) + K$

$\frac{N}{2^K} = 0$ ✗

$\frac{N}{2^K} = 1$

$\Rightarrow N = 2^K$

$\Rightarrow \log_2 N = K$

$T(N) = T(1) + \log_2 N$

[T.C → O(log₂N)]

Gray Code.

Given N, generate all N-bit numbers.

Note. → Numbers in the sequence should differ by exactly 1-bit.

N=2.

0 0] 1bit
0 1] 2 bit
1 0] 1-bit
1 1

// Not a valid sequence

N=2.

0 0] 1
0 1] 1
1 1] 1
1 0

ans → [0 1 3 2]

N=2

0 1] 1
0 0] 1
1 0] 1
1 1

[1 0 2 3]

N=2.

0 0] 1
1 0] 1
1 1] 1
0 1

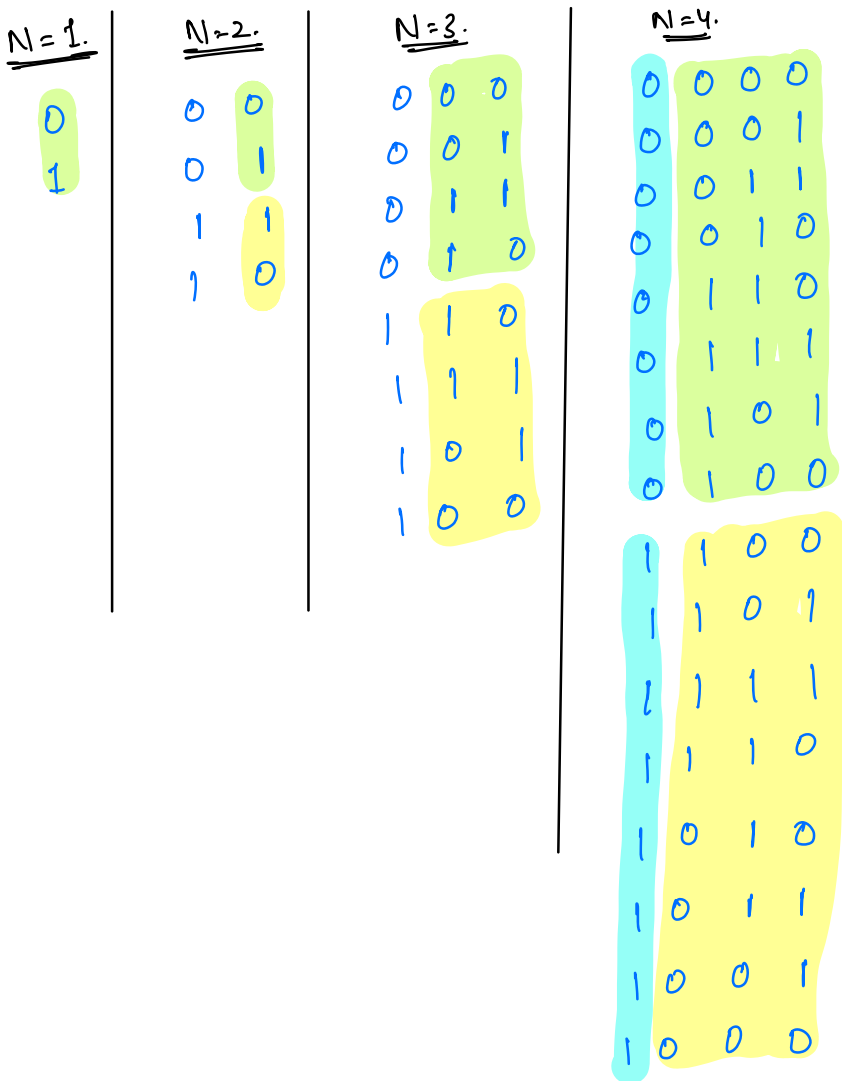
[0 2 3 1]

N=3. [0-7]

0 0 0] 1 bit
0 0 1] 1 bit
0 1 1] 1 bit
0 1 0] 1 bit
1 1 0] 1 bit
1 0 0] 1 bit
1 0 1] 1 bit
1 1 1] 1 bit

ans → [0, 1, 3, 2, 6, 4, 5, 7]

N=4. [0-15]



To get grey code for N-bit.

↳ gray code for $(N-1)^{th}$ bit

[finally, we have to return all these no's in decimal format.]

pseudo-code-

//Assm → Given N, return gray code sequnce of N bit numbers.

```
list<int> grayCode ( N ) {
  if (N == 1) { list<int> b ; b.insert(0), b.insert(1) }
              { return b }

  list<int> p = grayCode (N-1) ;
     int  x = p.size() ;        { x = 2^(N-1) elements }
  list < int > ans ;

  for ( i = 0 ; i < x ; i++) {
          ans.insert ( p[i] )
  }                                  ] x.

  for ( i = x-1 ; i >= 0 ; i--) {
          ans.insert ( p[i] + 2^(N-1) )
                 = p[i] | (1 << N-1)
  }                                  ] x

  return ans
}
```

2x iterations

$2 * 2^{N-1}$

$\Rightarrow 2^N$ iterations.

$$T(N) = T(N-1) + 2^N .$$

$$T(N) = T(N-1) + 2^N$$

$$T(N-1) = T(N-2) + 2^{N-1}$$

$$T(N) = T(N-2) + 2^{N-1} + 2^N$$

$$T(N-2) = T(N-3) + 2^{N-2}$$

$$T(N) = T(N-3) + 2^{N-2} + 2^{N-1} + 2^N$$

$$T(N-3) = T(N-4) + 2^{N-3}$$

$$T(N) = T(N-4) + 2^{N-3} + 2^{N-2} + 2^{N-1} + 2^N$$

$$\vdots$$

$$T(N) = 2^0 + 2^1 + 2^2 + 2^3 + \cdots + 2^{N-2} + 2^{N-1} + 2^N$$

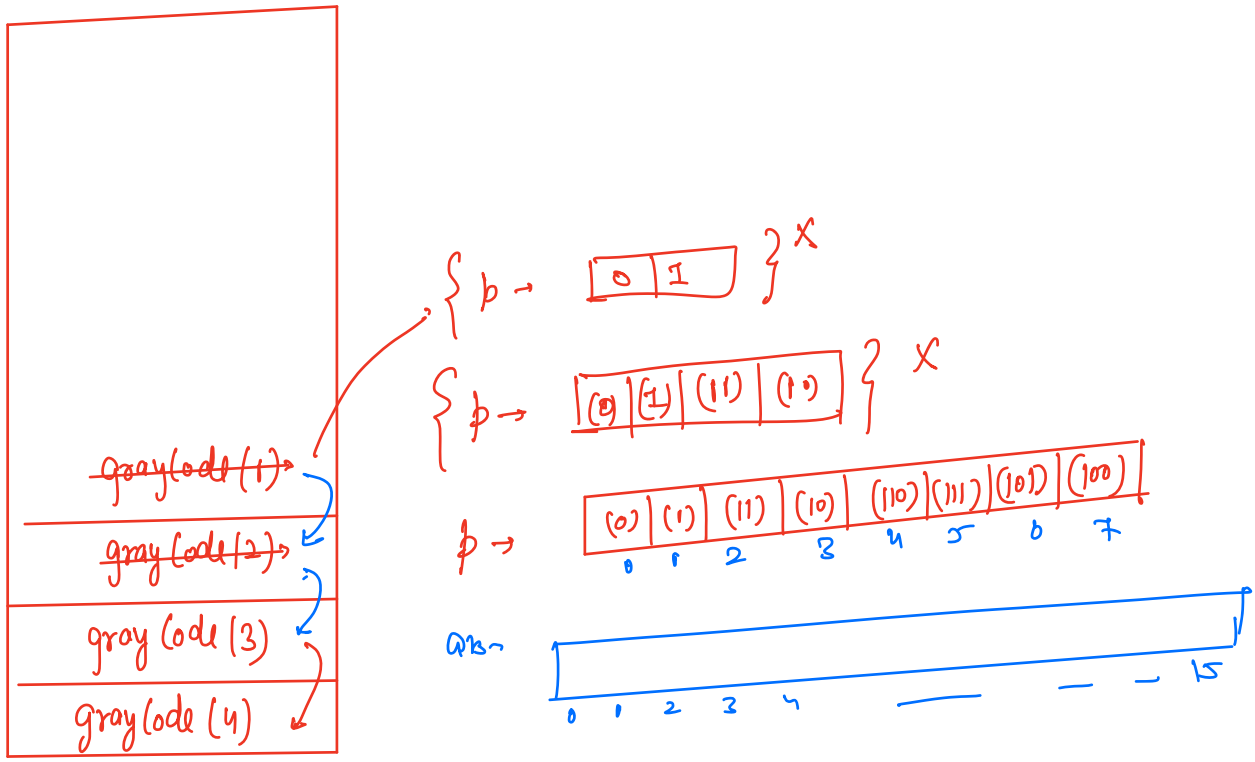
[$a = 1$, $r = 2$, no. of terms $= n+1$]

$$= \frac{1\left[2^{N+1} - 1\right]}{[2-1]} = 2^{N+1} - 1 = 2 \cdot 2^N - 1$$

T.C $\rightarrow O(2^N)$

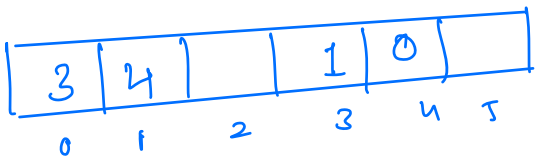
S.C $\rightarrow O(2^N)$

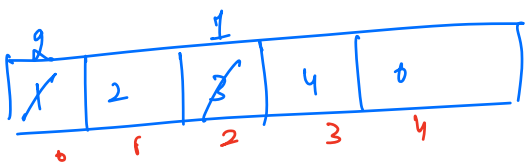
Ans.


gray(code(1))
gray(code(2))
gray(code(3))
gray(code(4))
p →
0 1
X
p →
(0) (1) (11) (10)
X
p →
(0) (1) (11) (10) (110) (111) (101) (100)
0 1 2 3 4 5 6 7
Ans
0 1 2 3 4 15

arr[i] → arr[arr[i]]. {0, N-1}

A[i] with A[A[i]]





i = 0.

```
temp = arr[i]
arr[i] = arr[arr[i]]
arr[arr[i]] = temp.
```

temp → 1

arr[0] = 2.

→ {and at least 1 element at its correct position}