Steps → 1) Decide what the function do.
2) Build logic on how to use subproblems to solve the current problem.
3) Define base case.

Q → Given an integer N, find the sum of digits of N.

N = 2386   Ans = 2 + 3 + 8 + 6 = 19

(N >= 0)

1) int sod (N) { . . . }

2)

N = 18636 ⟶ 1 + 8 + 6 + 3 + 6

N/10   N%10

$\boxed{sod(N) = sod(N/10) + (N\%10)}$

3) if (N <= 9) ⟶ Ans = N ✓ (Any)
if (N == 0) ⟶ Ans = 0 ✓

```
int sod (N) {
    if (N == 0)
        return 0;
    return sod(N/10) + N % 10;
}
```

```
int sod (N) {
    if (N <= 9)
        return N;
    return sod(N/10) + N % 10;
}
```

Q → Given integers a, b. Find a^b using recursion. $\boxed{b >= 0}$ $\boxed{a > 0}$

Eg → a = 2   b = 3   Ans = 2^3 = 8
a = 3   b = 3   Ans = 3^3 = 27

$\boxed{x * y = y * x}$

1) long pow(a, b) { . . . }

2)

$2^5 = \underbrace{2 * 2 * 2 * 2}_{2^4} * 2$ $2^4 = \underbrace{2 * 2 * 2}_{2^3} * 2$

$$\boxed{a^b = a^{b-1} * a}$$

3) if (b == 0) ⟶ Ans = 1

```
long pow(a, b) {   // Power
    if (b == 0)
        return 1;
    return pow(a, b-1) * a;
}
```

pow(2, 9) $\xrightarrow{1}$ pow(2, 8) $\xrightarrow{2}$ pow(2, 7) $\xrightarrow{3}$ pow(2, 6) $\xrightarrow{4}$ pow(2, 5) $\xrightarrow{5}$ pow(2, 4) $\xrightarrow{6}$ pow(2, 3) $\xrightarrow{7}$ pow(2, 2) $\xrightarrow{8}$ pow(2, 1) $\xrightarrow{9}$ pow(2, 0)

$\boxed{\#\text{function calls} = b}$

---

1) long pow(a, b) { . . . }

3) if (b == 0) ⟶ Ans = 1

2)

$$2^7 = \underbrace{2 * 2 * 2}_{2^3} * \underbrace{2 * 2 * 2}_{2^3} * \underline{2}$$

$$2^7 = 2^3 * 2^3 * 2$$

$$2^6 = \underbrace{2 * 2 * 2}_{2^3} * \underbrace{2 * 2 * 2}_{2^3}$$

$$2^6 = 2^3 * 2^3$$

$$\boxed{a^b \rightarrow \begin{cases} a^{b/2} * a^{b/2} * a, & b \text{ is odd} \\ a^{b/2} * a^{b/2}, & b \text{ is even} \end{cases}}$$

```
long pow(a, b) {
    if (b == 0)
        return 1;
    if (b % 2 == 1) {
        return pow(a, b/2) * pow(a, b/2) * a;
    else
```

```
    return pow(a, b/2) * pow(a, b/2);
}
```

```
long pow(a, b) {  // Fast Power
    if (b == 0)
        return 1;
    he = pow(a, b/2);
    if (b % 2 == 1)
        return he * he * a;
    else
        return he * he;
}
```

$\boxed{\#\ \text{function calls} = \log(b)}$

```
512
pow(2, 9) {
    he = pow(2, 4) {              16
        he = pow(2, 2) {          4
            he = pow(2, 1) {      2
                he = pow(2, 0) {  1
                    return = 1;
                }
                return 1*1*2;
            }
            return 2*2;
        }
        return 4*4;
    }
    return 16*16*2;
}
```

Time complexity of Basic Recursion

TC → function of input to define time.

$f(x) = x^2 + 3x + \dots$

$T(N) \rightarrow$ function of N

1) // sum of natural numbers

```
int sum(N) {
    if (N == 1)
        return 1;
    return sum(N-1) + N;
}
```

$\boxed{T(N) = T(N-1) + 1}$

```
int sum(N) {
    if (N == 1)
        return 1;      } TC = O(1)
    return N;
}
```

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

$T(N-2) + 1$

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

$T(N-3) + 1$

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

$T(N-4) + 1$

$\boxed{T(1) = 1} \rightarrow$ Base case

$T(N) = T(N-4) + 4$

$\vdots$

$\boxed{T(N) = T(N-K) + K}$

$N - K = 1 \Rightarrow K = (N-1)$

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

$= 1 + N - 1 = N$

$TC = O(N)$

---

2)

```
long pow(a, b) {   //Power
    if (b == 0)
        return 1;
    return pow(a, b-1) * a;
}
```

$T(N) = \boxed{T(b) = T(b-1) + 1}$

$TC = O(b)$ # function calls = b

3)

```
long pow(a, b) {   //Fast Power
    if (b == 0)
        return 1;
    he = pow(a, b/2);
    if (b % 2 == 1)
        return he * he * a;
    else
        return he * he;
}
```

$T(N) = \boxed{T(b) = T(b/2) + 1}$

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

$T(b/4) + 1$

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

$T(b/8) + 1$

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

$T(b/16) + 1$

$T(b) = T(b/16) + 4$

$\vdots$

$\boxed{T(b) = T(b/2^K) + K}$

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

$\boxed{T(1) = 1}$

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

$\Rightarrow b = 2^K \Rightarrow K = \log_2(b)$

$\frac{b}{2^K} = 0 \Rightarrow b = 0$ ✗

$T(b) = T(1) + \log(b) = 1 + \log(b)$

$TC = O(\log(b))$

function calls $= \log(b)$

If all other steps are O(1) in TC except for function calls

$\Rightarrow TC = O(\#\text{ function calls})$

Practice T(N)

10:52 PM

1) $T(N) = 2\,T(N/2) + 1$, $\boxed{T(1) = 1}$

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

$2\,T(N/4) + 1$

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

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

$2\,T(N/8) + 1$

$T(N) = 4(2\,T(N/8) + 1) + (2+1)$

$T(N) = 8\,T(N/8) + (4+2+1)$

$2\,T(N/16) + 1$

$T(N) = 8(2\,T(N/16) + 1) + (4+2+1)$

$= 16\,T(N/16) + (8+4+2+1)$

$\vdots$

$T(N) = 2^K\,T(N/2^K) + (2^{K-1} + 2^{K-2} + 2^{K-3} + \dots + 2 + 1) \rightarrow (2^K - 1)$

$T(N) = 2^K\,T(N/2^K) + 2^K - 1$ $\quad \boxed{T(1) = 1}$

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

$T(N) = N\,T(1) + N - 1 = N + N - 1 = \underline{2N-1}$ ✓

---

2) $T(N) = 2[T(N-1)] + 1$, $\boxed{T(0) = 1}$ $\quad \boxed{f(x) = 2\,f(x/2) + 1}$

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

$2\,T(N-2) + 1$

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

$= 4\,T(N-2) + (2+1)$

$2\,T(N-3) + 1$

$T(N) = 4(2\,T(N-3) + 1) + (2+1)$

$T(N) = 8\,T(N-3) + (4+2+1)$

$\vdots$

$T(N) = 2^K\,T(N-K) + (2^{K-1} + 2^{K-2} + \dots + 2 + 1) \rightarrow 2^K - 1$

$T(N) = 2^K\,T(N-K) + 2^K - 1$ $\quad \boxed{T(0) = 1}$

$N - K = 0 \Rightarrow \boxed{K = N}$

$T(N) = 2^N\,T(0) + 2^N - 1 = 2^N + 2^N - 1 = \underline{2.2^N - 1}$ ✓ $\quad (2.2^N \rightarrow 2^{N+1})$

Fibonacci Numbers

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

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

$$\boxed{T(N-2) < T(N-1)}$$

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

$$= 2T(N-1) + 1 \qquad TC < O(2^N)$$

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

$T(N-1)$ → $T(N-2) + T(N-3) + 1$

$T(N-2)$ → $T(N-3) + T(N-4) + 1$

$$T(N) = T(N-2) + T(N-3) + 1$$
$$+ T(N-3) + T(N-4) + 1 \quad + 1$$
$$T(N) = T(N-2) + 2T(N-3) + T(N-4) + 3$$

☹ complex

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

$$\boxed{T(N-2) < T(N-1)}$$

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

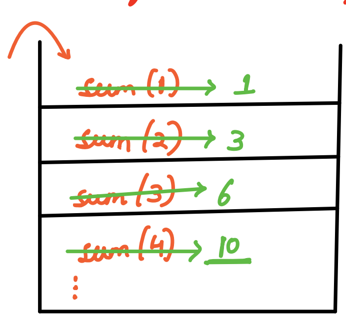
$$\boxed{T(N) < 2T(N-1) + 1} \qquad TC = O(2^N)$$

H.W → Find # function calls & check TC = O(# function calls).

Space complexity → Max size of stack memory at any point.

O(size of stack memory)

1)
```
int sum(N){
    if(N==1)
        return 1;
    return sum(N-1) + N;
}
```



Stack Memory

$SC = O(N)$

2)
```
int fib(N) {
    if (N==0 || N==1)
        return N;
    return fib(N-1) + fib(N-2);
}
```
① ② 3

N=3 →

~~fib(1)~~ → 1 ~~fib(0)~~ → 0
~~fib(2)~~ → 1 ~~fib(1)~~ → 1
~~fib(3)~~ → 2

Stack Memory

SC = O(N)

```
fib(3) { ✓
    return fib(2) + fib(1);

    fib(2) { ✓
    ①   return fib(1) + fib(0);

        fib(1) { ✓
        ①   return 1;
        } ←
        fib(0) { ✓
        ②   return 0;
        } ←
    3   return 1+0;
    } ←
    fib(1) { ✓
    ②   return 1;
    } ←

    return 1+1;
}
```