Todays Content:

- Sum of digits
- pow(a, n) {
  | // way1, // way2, // way3
  }
- pow(a, n, p) { - }
- TC of recursive Codes
- SC of recursive Codes

18) Given a $N \geq 0$ find sum of digits using recursion

Eg: Sum(123) = 6

Sum(257) = 14

```
Sum(N){
    if(N==0){return 0}
    return{sum(N/10)+ N%10}
}
```

Ass: Given N, calculate & return sum of digits

Eg: N = Sod(1245 3)

↳ Sod(1245) + 3

Sod(N) = N%10 + Sod(N/10)

2Q) Given a, n find a^n using recursion

Eg1:

a	n	a^n
2	5	32
3	4	81

```
int pow1(a, n){ Ass: Calulate & return a^n
    if(N==0){return 1}
    return pow1(a, n-1) * a
}
```

$a^n = \underbrace{a * a * .. \ a}_{\#n \text{ times}} * a$

$\#n-1$ times

$a^n = a^{n-1} * a$

$= pow(a, n-1) * a$

Dry Run → TODO

```
int pow2(a, n){ Ass: Calulate & return a^n
    if(n==0){return 1}
    if(n%2==0){
        return [ pow2(a, n/2) * pow2(a, n/2) ]
    }
    else{
        return [ pow2(a, n/2) * pow2(a, n/2) * a ]
    }
}
```

Obs: Calulate pow2(a, n/2) once & Use it

$a^{10} = a^5 * a^5$

$a^{14} = a^7 * a^7$

$a^{11} = a^5 * a^5 * a^1$

$a^{13} = a^6 * a^6 * a$

$a^n =$ if n is even

$\underbrace{a^{n/2} * a^{n/2}}$

$pow(a, n/2) * pow(a, n/2)$

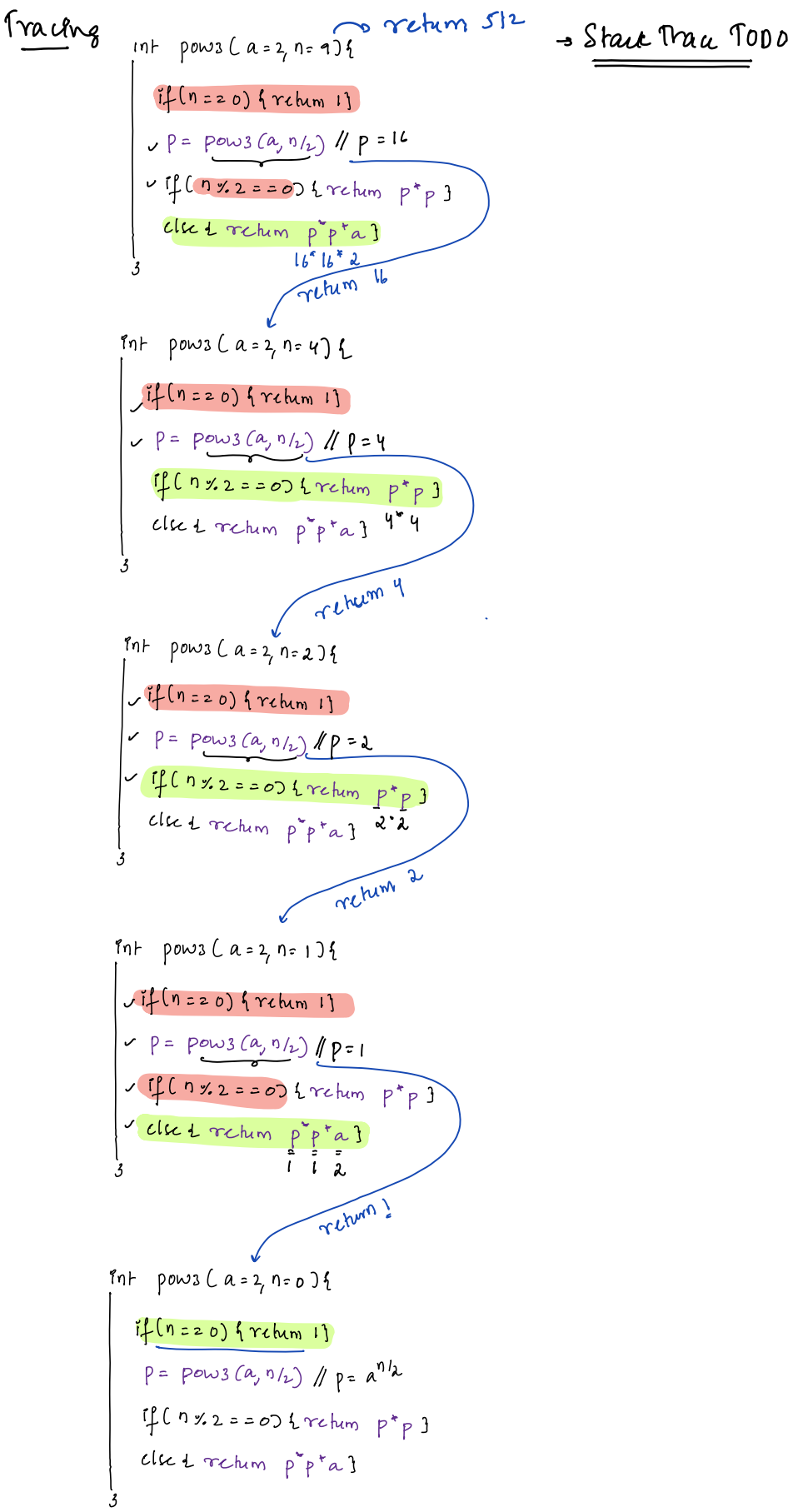
if n is odd

$\underbrace{a^{n/2} * a^{n/2} * a}$

$pow(a, n/2) * pow(a, n/2) * a$

```
int pow3(a, n){ Ass: Calculate & return a^n
    if(n==0){return 1}
    p = pow3(a, n/2) // p = a^(n/2)
    if(n%2==0){return p*p}
    else{return p*p*a}
}
```

Tracing



→ Stack Trace TODO

Q) Given a, n, m calculate $a^n \% m$

Constraints: Note: Take care of overflows

$1 \le a \le 10^9$

$1 \le n \le 10^9$

$2 \le m \le 10^9$

Ass: Calculate & return $a^n \% m$

```
powmod(int a, int n, int m){
    if(n==0) return 1;
    long p = powmod(a, n/2, m); // a^(n/2) % m
                 └ max = [m-1] ≈ 10^9
    if(n%2==0){
        return (p*p)%m;   → can exceed integer range
    }                       10^9 10^9 → (10^18)%m → ≈10^9
    else{
        return (p*p*a)%m =
                 ↓
        ≈10^9 * ≈10^9 * ≈10^9 = (10^27)%m   → exceeding long range
        return [(p*p)%m * a]%m
                 ↓ ↓
        ≈10^9 * 10^9
          [10^18 % m]
                          ≈ 10^9 * 10^9 = 10^18 % m ≈ 10^9
    }
}
```

Fast Exponentation TC: $O(\log_2 N)$ 8:25 → 8:36

```
int powmod(int a, int n, int m){
    if(n==0){return 1;}
    long p = powmod(a, n/2, m); // p = a^(n/2)%m
    if(n%2==0){return (p*p)%m;}
    else{return [(p*p)%m * a]%m;}
}
```

TC for Recursive Code using Recusive Relation

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

Time take to calculate Sum(N) = f(N) = O(N)

sum(N-1) = f(N-1)

$f(N) = f(N-1) + 1, \quad f(1) = 1$

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

$= f(N-2) + 2$

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

$= f(N-3) + 3$

$f(N-3) = f(N-4) + 1$

$= f(N-4) + 4$

// Generalization

(unknown term, known term)

$f(N) = f(N-k) + k, \quad f(1) = 1$

$N - k = 1 \Rightarrow N = 1 + k \Rightarrow k = N-1$

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

$f(N) = 1 + N - 1 \Rightarrow O(N)$

```
int fact(N){ // Time taken to calculate fact(N) = f(N)
    if(N==1){return 1}
    return [fact(N-1) * N]
}
```

$f(N) = f(N-1) + O(1) \quad f(1) = 1$

fact(N-1) = f(N-1)

```
int pow1(a, n){ // Time taken to calculate pow(a,n) = f(n)
    if(N==0) {return 1;}
    return pow1(a, n-1) * a
}                 f(n-1)
```

$f(N) = f(N-1) + 1, \quad f(0) = 1$

TODO

```
int pow2(a, n){ // Time taken to calculate pow2(a,n) = f(N)
    if(n==0) {return 1;}
    if(n%2==0){
        return [ pow2(a, n/2) * pow2(a, n/2) ]
                 f(N/2)          f(N/2)
    }
    else{
        return [ pow2(a, n/2) * pow2(a, n/2) * a ]
                 f(N/2)          f(N/2)
    }
}
```

$f(N) = 2f(N/2) + 1 \quad f(0) = 1, \; f(1) = O(1)$

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

$= 2[2f(N/4) + 1] + 1$

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

$f(N/4) = 2f(N/8) + 1$

$= 4[2f(N/8) + 1] + 3$

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

$f(N/8) = 2f(N/16) + 1$

$= 8[2 * f(N/16) + 1] + 7$

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

// After k Substitution :

$f(N) = 2^k f(N/2^k) + 2^k - 1, \quad f(0) = 1, \; f(1) = 1$

$N/2^k = 1 \Rightarrow N = 2^k \Rightarrow$ Apply $\log_2$ both sides $\boxed{k = \log_2 N}$

$N f(N/N) + N - 1 \Rightarrow N f(1) + N - 1 \Rightarrow O(N) = f(N)$

```
int pows(a, n){ // Time taken n to calculate pows(a,n) = f(N)
    if(n == 0){return 1}
    p = pows(a, n/2)  f(N/2)
    if(n%2 == 0){return p*p}
    else{return p*p*a}
}
```

$f(N) = f(N/2) + 1$, $f(0) = 1$, $f(1) = 1$

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

$= f(N/4) + 2 = f(N/2^2) + 2$

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

$= f(N/8) + 3 = f(N/2^3) + 3$

$f(N/8) = f(N/16) + 1$

$= f(N/16) + 4 = f(N/2^4) + 4$

// After k substitutions:

$f(N) = f(N/2^k) + k \quad T(0) = 1, T(1) = 1$

$N/2^k = 1, \; 2^k = N, \; k = \log_2 N$

$= f(N/N) + \log_2 N$

$= O(\log_2 N)$

$f(N) = O(\log_2 N)$

```
int powmod(int a, int n, int m){
    if(n==0){return 1}
    long p = powmod(a, n/2, m) // p = a^(n/2) % m
    if(n%2==0){ return (p*p)%m}
    else{ return [(p*p)%m * a]%m }
}
```

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

$T(N) = O(\log_2 N)$

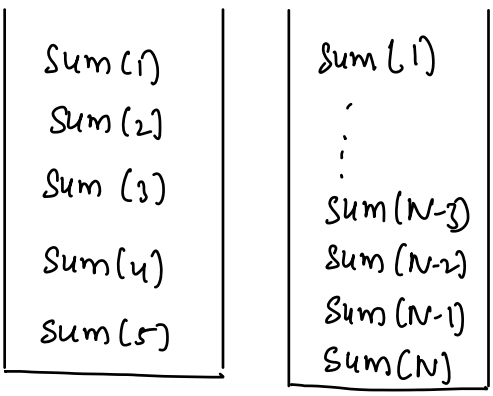
Space Complexity for Recursive Code ?

↳ obs1: We have Space, because we store function calls

↳ SC: Our Stack Size

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

f(N-1)



↳ SC: O(N)

```
int fact(N){ ⟶ SC: O(N)
    if(N==1){return 1}
    return [fact(N-1) * N]
}
```

f(N-1)

```
int pow1(a, n){ ⟶ SC: O(N)
    if(N==0){return 1}
    return pow1(a, n-1) * a
}
```

f(n-1)

```
int pow3(a, n){   SC: O(log N)
    if(n == 0){ return 1 }
    p = pow3(a, n/2)   f(N/2)
    if(n % 2 == 0){ return p*p }
    else{ return p*p*a }
}
```

```
pow(a, 0)
pow(a, 1)
.
pow(a, n/8)
pow(a, n/4)
pow(a, n/2)
pow(a, n)
```

→ Stack Size → O(log N)

// fib(): → TC : $O(2^N)$

```
int fib(N){   // Time taken to calculate fib(N) = f(N)
    if(N <= 1) return N
    return [fib(N-1) + fib(N-2)]
           f(N-1)    f(N-2)
}
```

$f(N) = f(N-1) + f(N-2) + O(1)$

$f(1) = 1,\ f(0) = 0$

$f(N) = f(N-1) + f(N-2) + O(1)$, Note: Solving in Substitution is tricky

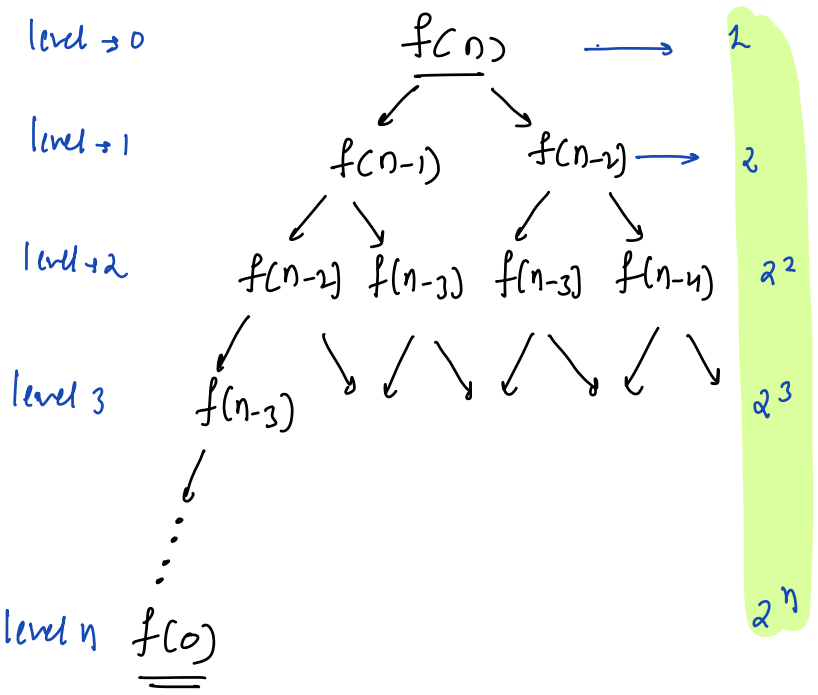
$f(N-1) = f(N-2) + f(N-3) + 1$

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

$= f(N-2) + f(N-3) + f(N-3) + f(N-4) + 3$

$= f(N-2) + 2f(N-3) + f(N-4) + 3$

// TC → level → 0



// Total function Calls = $2^0 + 2^1 + 2^2 + \cdots 2^n$

G.P, $a = 1$, $r = 2$, $t = n+1$

$$\frac{a * [r^{n-1}]}{r-1} = \frac{1 * [2^{n+1} - 1]}{2-1} = 2^{n+1} - 1$$

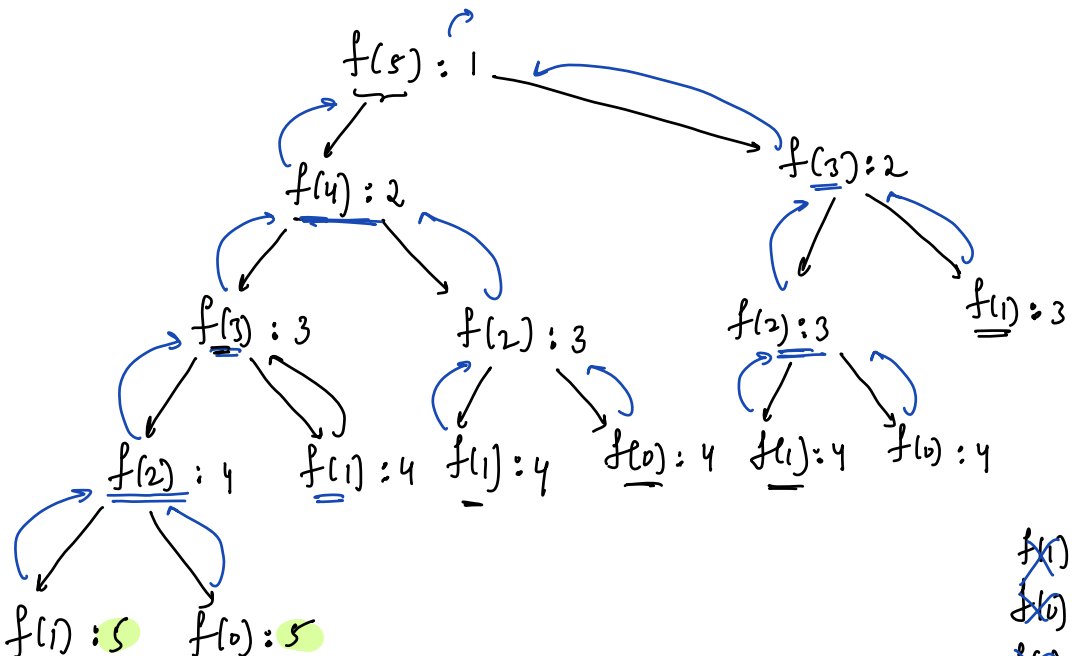
$$= 2 * 2^n - 1$$

$$= O(2^n)$$

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

SC: O(N) ?

↳ max stack size



f(1)
f(0)
f(1)
f(2)
f(3)
f(0)
f(1)
f(2)
f(4)
f(0)
f(1)
f(2)
f(3)
f(4)
f(5)

obs1 : Max stack size = 5