Todays Content

1. Sum of Digits of N
2. Print 1 2 2 3 3 3 4 4 4 4
3. Fibanacci..
4. Power Function
5. Fast exponentiation with % arithmetic.

Steps for Recursion:

Assumption: Decide what your function does # {input, does, return}

Main logic: Solve problem using Subproblems # Recursive step

Have a believe that subproblem will work as per assumption.

Base Condition: Input for which recursion needs to stop

Q: Given N return sum of digits using recursion.

Note: $N > 0$

Ex: $Sum(239) = 2+3+9 = 14$

$Sum(7864) = 7+8+6+4 = 25$

$Sum(7864) = sum(786) + 4$

Ex: $N = \boxed{d_1 d_2 \ldots d_{y-1}}\ \boxed{d_y}$ $\rightarrow N\%10 = d_y$

$\underbrace{\phantom{d_1 d_2 \ldots d_{y-1}}}_{N/10}\ \underbrace{}_{N\%10}$ $\rightarrow N/10 = d_1 d_2 \ldots d_{y-1}$

$Sum(N) = Sum(N/10) + N\%10$

Ass: Given N calculate & return sum of digits of N.

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

Trace:

```
int Sum(N=365) : a
    if (N==0) {return 0}
    return Sum(N/10) + N%10;
}
```

28: Given N print below pattern 1 2 2 3 3 3 ....

Pat(4) : 1 22 333 4444

Pat(5) : 1 22 333 4444 55555

Pat(4)

Iterate 5 times & Print 5

Pat(N) : 1 22 333 .. N-1 N-1 .. N-1 N N N .. N

Pat(N-1)

Iterate N times & Print N

Ass: Given N print pattern return nothing.

```
void Pat(int N){
    if(N==0){ return; }
    Pat(N-1)
    for(int i=1; i<=N; i++){
        print(N);
    }
    print(" ");
}
```

Fibanacci:

0 1 2 3 4 5 6 7 8 9 10

Series = 0 1 1 2 3 5 8 13 21 34 55

Note: $N >= 0$

$Fib(6) = Fib(5) + Fib(4)$

$Fib(10) = Fib(9) + Fib(8)$

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

Ass: Given N, calculate & return N^{th} fib number.

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

How to get base conditions?

Inputs for which main logic fails.

Ex:

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

✓ $Fib(2) = Fib(1) + Fib(0)$

✗ $Fib(1) = Fib(0) + Fib(-1)$

✗ $Fib(0) = Fib(-1) + Fib(-2)$

Q2 : pow(a,n) : Calculate & return a^n

pow(a,5) = $\underbrace{a * a * a * a}_{} * a = a^5$

pow(a,5) = pow(a,4) * a

$\overbrace{\qquad\qquad}^{N-1}$

pow(a,n) = $a * a \ldots a * a = a^N$

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

Ass: Given N, calculate & return a^N

```
long pow(long a, long N){   TC: O(N)   SC: O(N)
    if(N==0){return 1;}
}   return pow(a,N-1) * a;
```

Other ways to solve a problem with Subproblems:

Note: Can break a problem at corners or at center.

pow(a,8) = $a^7 * a$

pow(a,8) = $a^4 * a^4$ # Break problem at centre.

pow(a,8) = pow(a,4) * pow(a,4)

pow(a,9) = $a^4 * a^4 * a$; # 9/2 = 4. $a^{9/2} * a^{9/2} * a$; # odd power

pow(a,4) * pow(a,4) * a

Ass: Given N calculate a^N & return it;

```
long pow(a, n){  TC: O(N)   SC: O(logN)
    if(N==0){return 1;}
    if(N%2==0){
    }   return pow(a,n/2) * pow(a,n/2)
    else{
}   }   return pow(a,n/2) * pow(a,n/2) * a; # odd power
```

long pow(a, N=4)

return $\underbrace{pow(a,2)}_{a^2}$ * $\underbrace{pow(a,2)}_{a^2}$ = a^4

→ return $\underbrace{pow(a,1)}_{a}$ * $\underbrace{pow(a,1)}_{a}$ # a*a = a^2

→ return pow(a,0) * pow(a,0) * a

→ return pow(a,0) * pow(a,0) * a # 1*1*a

#obs: In above code, same subproblem is called twice, so call it once store it & re-use it.

Assumption: Given N calculate a^N & return it;

```
long pow(long a, long n){ TC: O(logN) SC: O(logN)
    if(N==0){return 1;}
    long t= pow(a, n/2);
    if(N%2==0){
        return t*t;
    }
    else{
        return t*t*a;
    }
}
```

5Q Given a, N, M calculate and return $a^N \% M$

pow(a, n, m) : $a^n \% m$

Constraints

$1 <= a <= 10^9$ Ex: pow(3, 4, 4) = $(3^4) \% 4 = 81 \% 4 = 1$ (a n m)

$1 <= N <= 10^{18}$ pow(a = 2, n = 1000, m = 10^9+7) : $(2^{1000}) \% (10^9+7)$

$m = 10^9+7$ pow(a = 10, n = 1000, m = 10^9+7) = $(10^{1000}) \% (10^9+7)$

Issue: We cannot calculate a^n & take $\% m$ later, because a^n will overflow

Hint: We will use modular arithmetics

Ex: pow(a, n, m) = $(a^n) \% m$

if(n%2 == 0){

$(a^{n/2} * a^{n/2}) \% m$ # $(a*b) \% m = (a \% m * b \% m) \% m$

$(a^{n/2} \% m * a^{n/2} \% m) \% m$

(pow(a, n/2, m) * pow(a, n/2, m)) %m

}

else{ $(\underbrace{a^{n/2} * a^{n/2}}_{c} * \underbrace{a}_{d}) \% m$ # $(a*b) \% m = (a \% m * b \% m) \% m$

$((a^{n/2} * a^{n/2}) \% m * a \% m) \% m$

t = pow(a, n/2, m);

$((a^{n/2} \% m * a^{n/2} \% m) \% m * a \% m) \% m$

((pow(a, n/2, m) * pow(a, n/2, m)) %m * a%m) %m

$((m-1 * m-1) \% m = m-1 * m-1) \% m = m-1$

}

$(m-1)^2 = 10^{18} \% m = (m-1 * m-1) = 10^{18} \% m = m-1$

Given a, n, m calculate & return $a^n \% m$

```
long pow(int a, long n, long m){        TC: O(logN)  SC: O(logN)
    if(n==0){return 1;}

    long t = pow(a, n/2, m);
    if(n%2==0){
        return (t*t)%m
    }
    else{
        return ((t*t)%m * a%m)%m
    }
}
```

What if?

$pow(a,n,m) = (a^n)\%m$

if(n%2==0){

$(a^{n/2} * a^{n/2})\%m$

$(a^{n/2}\%m * a^{n/2}\%m)\%m$

} $(pow(a, n/2, m) * pow(a, n/2, m))\%m$

else{

$(a^{n/2} * a^{n/2} * \underline{a})\%m$ # $(a*b*c)\%m = (a\%m * b\%m * c\%m)\%m$

$(\underline{a^{n/2}\%m} * a^{n/2}\%m * a\%m)\%m$

$1 <= a <= 10^9$

$1 <= N <= 10^{18}$

$m = 10^9+7$

$(\underbrace{pow(a,n/2,m)}_{[0..m-1]} * \underbrace{pow(a,n/2,m)}_{[0..m-1]} * \underbrace{a\%m}_{[0..m-1]})\%m$

$= [0..m-1] * [0..m-1] * [0..m-1] \% m$

product of above 3 can go upto 10^{27} cannot store in long * overflow

} at max $\approx (m-1)^3 \% m$

$(m-1) \rightarrow (10^9)^3 \approx 10^{27}$