Agenda

i) pow(a,n)

ii) TC of recursive code

Q.1 Given N (N > 0), find sum of digits.

N = 124 ans = 1+2+4 = 7

N = 5921 ans = 5+9+2+1 = 17

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

Assumption: given N, returning sum of digits in N.

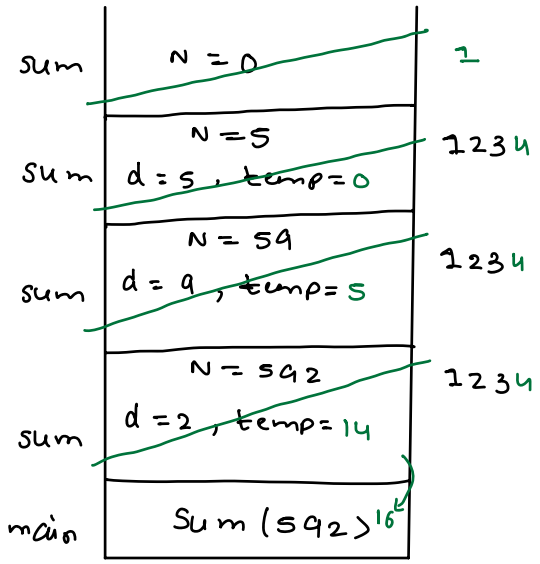
Main logic:

```
int d = N%10;
int temp = sum(N/10);
return d+temp.
```

Base condition:

```
if (N == 0) {
    return 0;
}
```

```
int sum (int N) {
    if (N == 0) {
1       return 0;
3   }
2   int d = N%10;
3   int temp = sum(N/10);
4   return d+temp;
}
```



Q.2 Given a and n, calculate a^n.

a = 2, n = 5 ans = $2^5 = 32$

a = 1, n = 1000 ans = $1^{1000} = 1$

a = 3, n = 4 ans = $3^4 = 81$

```
int pow (int a, int n) {
    if (n == 0) {
        return 1;
    }
    int temp = pow(a, n-1);
    return temp * a;
}
```

Assumption : given a, n pow returning $\Rightarrow a^n$.

Main logic :

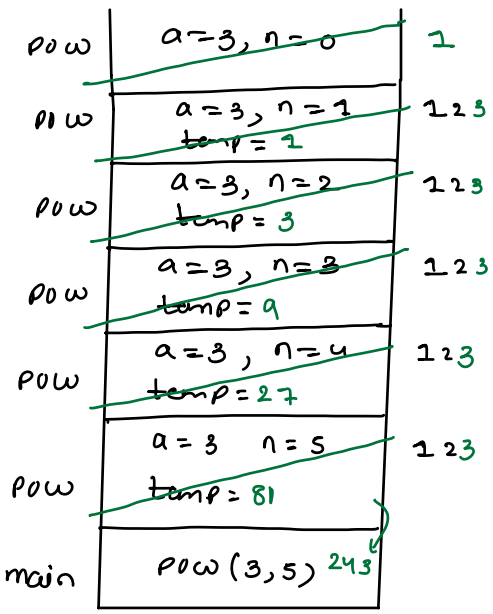
$a^n = a^{n-1} * a$

Base condition :

✓ n == 0 → return 1 ($a^0 = 1$)

n == 1 → return a ($a^1 = a$)

```
int pow (int a, int n) {
    if (n == 0) {
1       return 1;
    }
2   int temp = pow (a, n-1);
3   return temp * a;
}
```

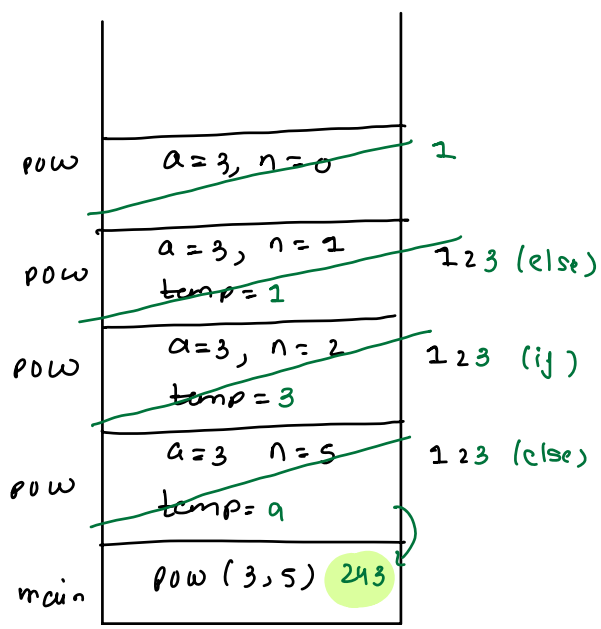


Something better

$$a^n = \begin{cases} a^{n/2} * a^{n/2} & (n \text{ is even}) \\ a^{n/2} * a^{n/2} * a & (n \text{ is odd}) \end{cases}$$

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

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



```
int pow (int a, int n) {
    if (n == 0) {
        return 1;
    }
    int temp = pow (a, n-1);
    return temp * a;
}
```

3, 20
↓
3, 19
↓
3, 18
↓
3, 17
↓
3, 16
⋮
..., $3^4 \rightarrow 3^3 \rightarrow 3^2 \rightarrow 3^1 \rightarrow 3^0$

20 calls

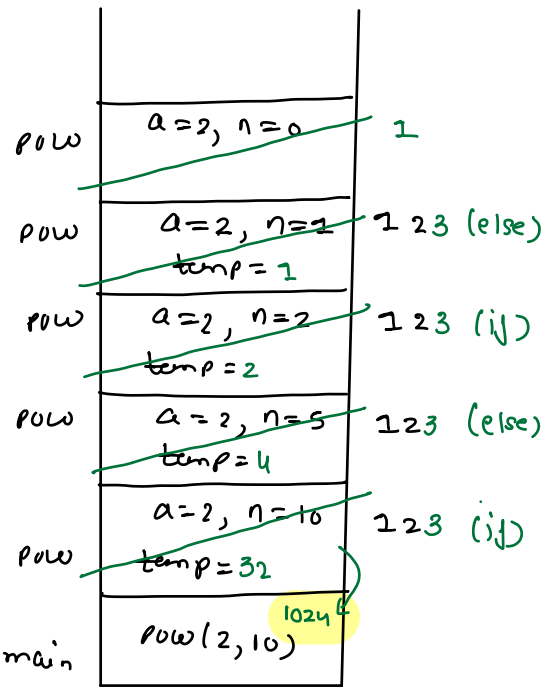
```
int pow (int a, int n) {
    if (n == 0) {
        return 1;
    }
    int temp = pow (a, n/2);
    if (n % 2 == 0) {
        return temp * temp;
    }
    else {
        return temp * temp * a;
    }
}
```

3, 20
↓
3, 10
↓
3, 5
↓
3, 2
↓
3, 1
↓
3, 0

5 calls

Dry run

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



Q.3 Calculate $a^n \% p$.

$1 <= a <= 10^9$

$1 <= n <= 10^5$

$1 <= p <= 10^9$

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

TC of Recursive code

Recursive code : a function getting called multiple no. of times.

TC of single function => x

total no. of functions calls => y

Overall TC = x * y

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

TC of single function = O(1)

Total no. of function calls = N

TC: O(N)

N = 5
↓
N = 4
↓
N = 3
↓
N = 2
↓
N = 1

```
int factorial (int N) {
    if (N == 0) {
        return 1;
    }
    int temp = factorial (N-1);
    return temp * N;
}
```

TC of single function = O(1)

Total no. of function calls = N

N = 5
↓
N = 4
↓
N = 3
↓
N = 2
↓
N = 1
↓
N = 0

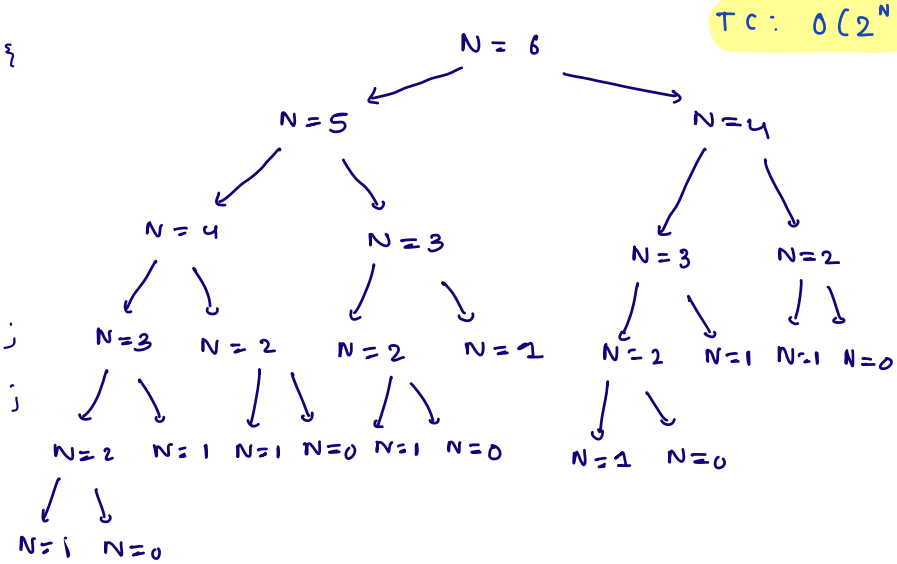
TC: O(N)

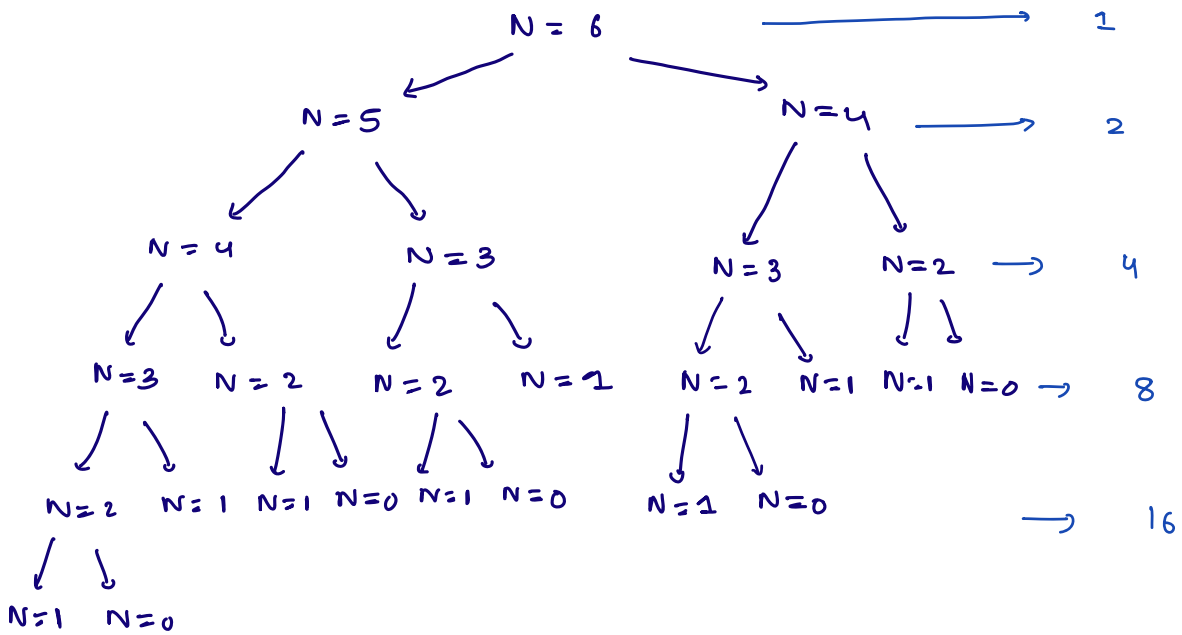
```
int fib (int N) {
    if (N == 0 || N == 1) {
        return N;
    }
    int temp1 = fib (N-1);
    int temp2 = fib (N-2);
    return temp1 + temp2;
}
```

TC of single function = O(1)

Total no. of function calls = 2^N

TC: $O(2^N)$





total calls = $1+2+4+8+16+\cdots\cdots$

$$S_t = \frac{a(r^t - 1)}{r-1}$$

$a = 1$

$r = 2$

$t = N$

$$= \frac{1(2^N - 1)}{1} \approx 2^N$$

```
boolean check (string str, int s, int e) {
    if (s==e || s>e) {
        return true;
    }
    if (str.charAt(s) != str.charAt(e)) {
        return false;
    }
    else {
        boolean ans = check (str, s+1, e-1) {
        return ans;
    }
}
```

TC of single func = O(1)

total no. of function = $\frac{n}{2}$

n = 10

0, 9
↓
1, 8
↓
2, 7
↓
3, 6
↓
4, 5
↓
5, 4

$\frac{n}{2}$

TC : O(n)

0, 20
↓
1, 19
↓
2, 18
↓
3, 17
↓
4, 16
↓
5, 15
↓
6, 14
↓
7, 13 → 8, 12 → 9, 11 → 10, 10

```
int pow (int a, int n) {
    if (n==0) {
        return 1;
    }
    int temp= pow(a,n-1);
    return temp * a;
}
```

TC of single function = O(1)

total no. of function calls = n

TC: O(n)

3, 7
↓
3, 6
↓
3, 5
↓
3,4 → 3,3 → 3,2 → 3,1 → 3,0

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

TC of single function = O(1)

total no. of calls = $\log_2 n$

3,20
↓
3,10
↓
3,5
↓
3,2
↓
3,1
↓
3,0

$\log_2 n$

TC: $\log_2 n$

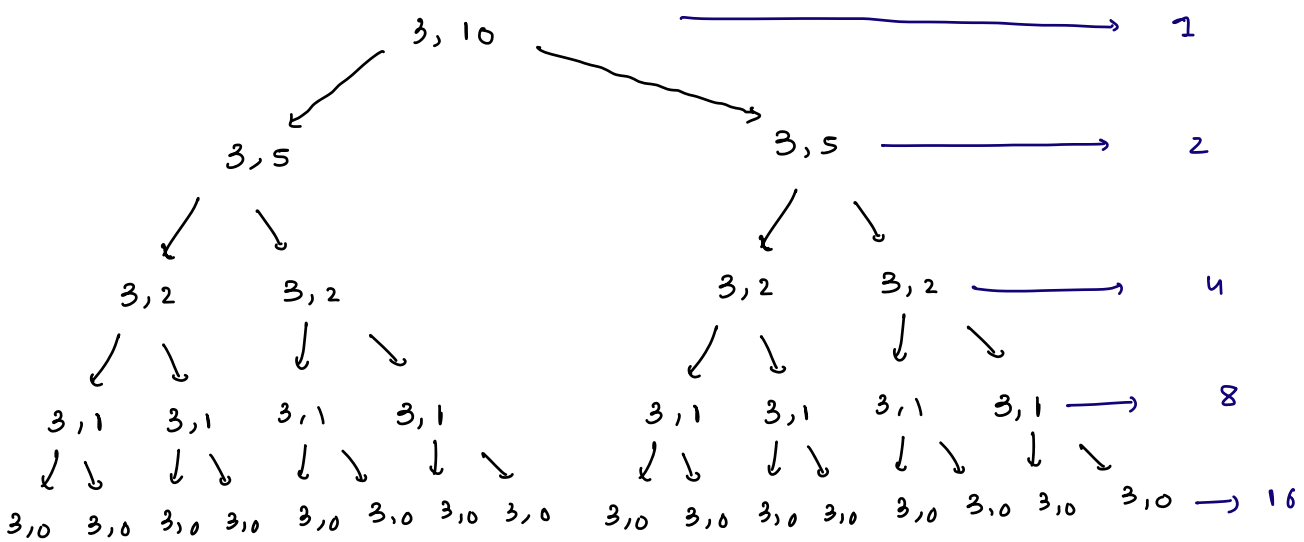
```
int pow(int a, int n) {
    if(n==0) {
        return 1;
    }
    if(n%2==0) {
        return pow(a, n/2) * pow(a, n/2);
    }
    else {
        return pow(a, n/2) * pow(a, n/2) * a;
    }
}
```

TC of single function = O(1)

total no. of calls = n

TC = O(n)

→ Pseudo Smart



total calls = $1+2+4+8+16+\dots$

$$S_t = \frac{a(r^t-1)}{r-1}$$

$a = 1$

$r = 2$

$t = \log_2 n$

$$= 1 \frac{(2^{\log_2 n} - 1)}{2 - 1} = 2^{\log_2 n} - 1 \quad \{2^{\log_2 n} = n\}$$

$$= n - 1 \approx n$$

Doubts

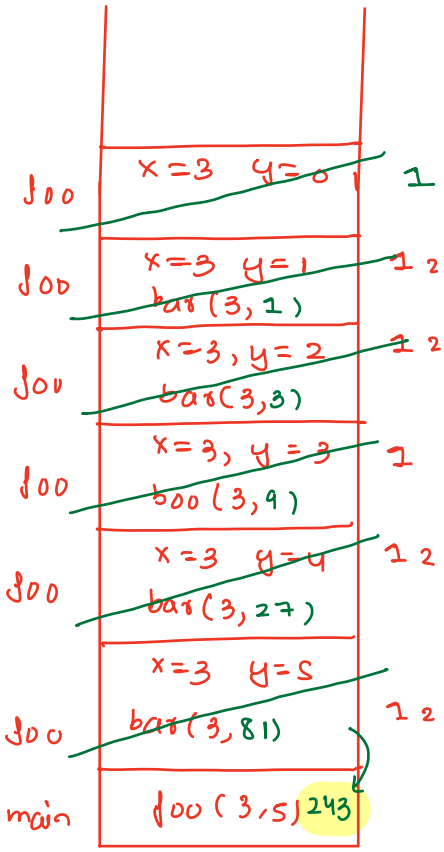
=

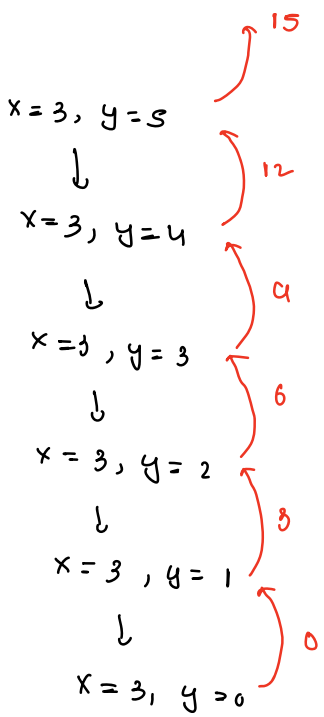
```
#include
using namespace std;

int bar(int x, int y){
   1 if (y == 0)   return 0;
   2 return (x + bar(x, y-1));
}

int foo(int x, int y){
   1 if (y == 0) return 1;
   2 return bar(x, foo(x, y-1));
}

int main(){
    cout << foo(3, 5);
}
```





bar(x, y) ⇒ x * y