Agenda

i) Pow(a,n)

ii) How to find TC of recursive code.

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

N = 1104 sum = 1+1+0+4 = 6

N = 128 sum = 1+2+8 = 11

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

Assumption: Given N, sum() returning sum of all digits.

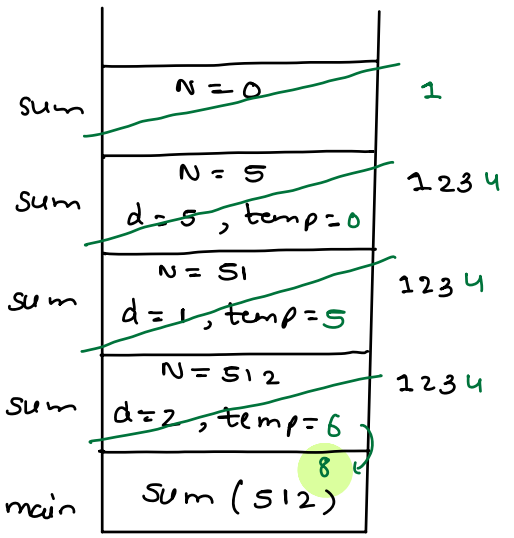
Main logic:

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

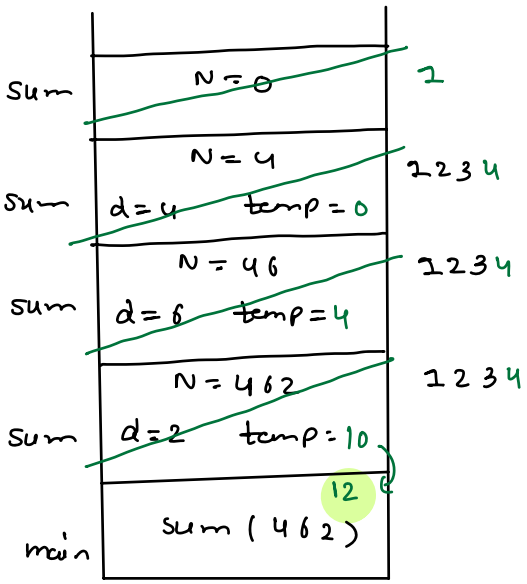
Base condition:

```
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 temp+d;
}
```




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



Q.2 Given a, n. Calculate a^n.

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

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

a = 1 n = 100 | ans = $1^{100} = 1$

a = 20 n = 0 | ans = $20^0 = 1$

$a^0 = 1$

```
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 returns a^n.

Main logic:

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

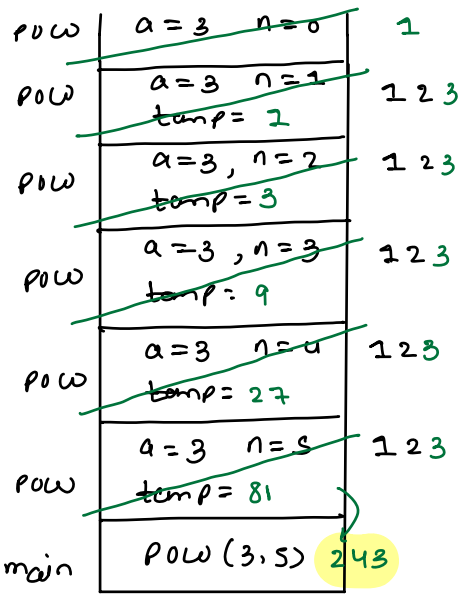
Base condition:

```
if (n == 1) {
    return a;
}
```

✓

```
if (n == 0) {
    return 1;
}
```

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



something better

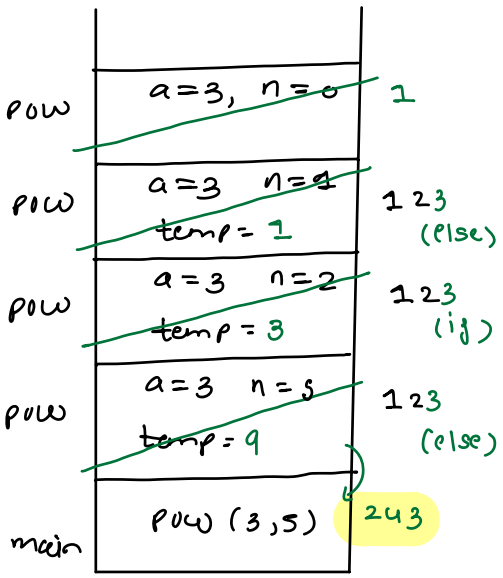
$$a^n = \begin{cases} a^{n/2} * a^{n/2} & (n \text{ is even}) \\ a^{n/2} * a^{n/2} * a & (n \text{ is odd}) \end{cases}$$

$$5^8 = 5^4 * 5^4$$

$$5^9 = 5^4 * 5^4 * 5$$

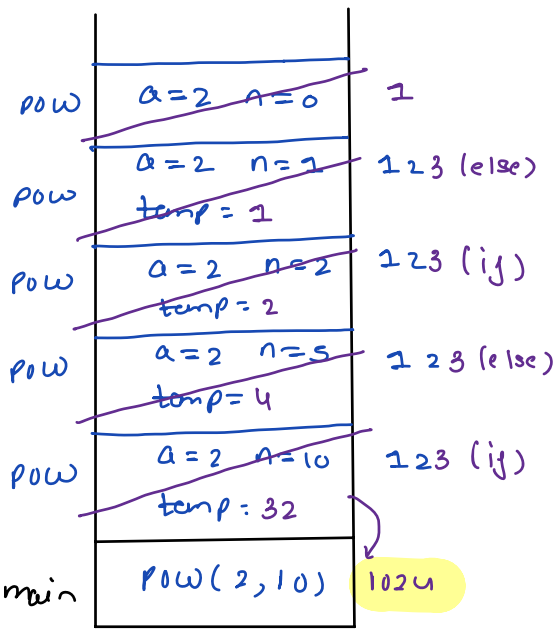
```
int pow (int a, int n) {
    if(n==0) {
1       return 1;
    }
2   int temp = pow(a,n/2);

    if (n%2==0) {
3           return temp* temp;
    }
    else {
            return temp * temp * a;
    }
}
```




```
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;
    }
}
```



```
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
↓
3,3
↓
3,2
↓
3,1
↓
3,0

n 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

$\log_2 n$ calls

Q.3 Given a, n, p. 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 times.

TC of single function : x

total no. of function 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 calls : n

n=5
↓
n=4
↓
n=3
↓
n=2
↓
n=1

TC: O(n)

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

TC of single func = O(1)

total no. of calls = n

n=5
↓
n=4
↓
n=3
↓
n=2
↓
n=1
↓
n=0

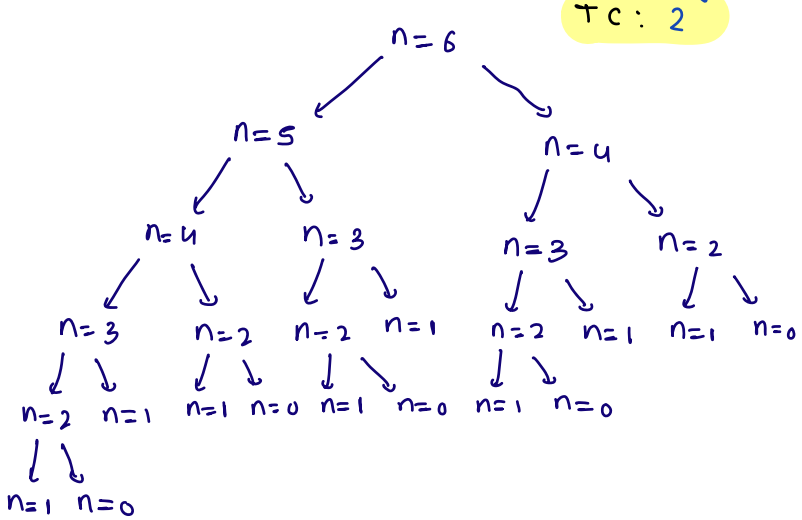
TC: O(n)

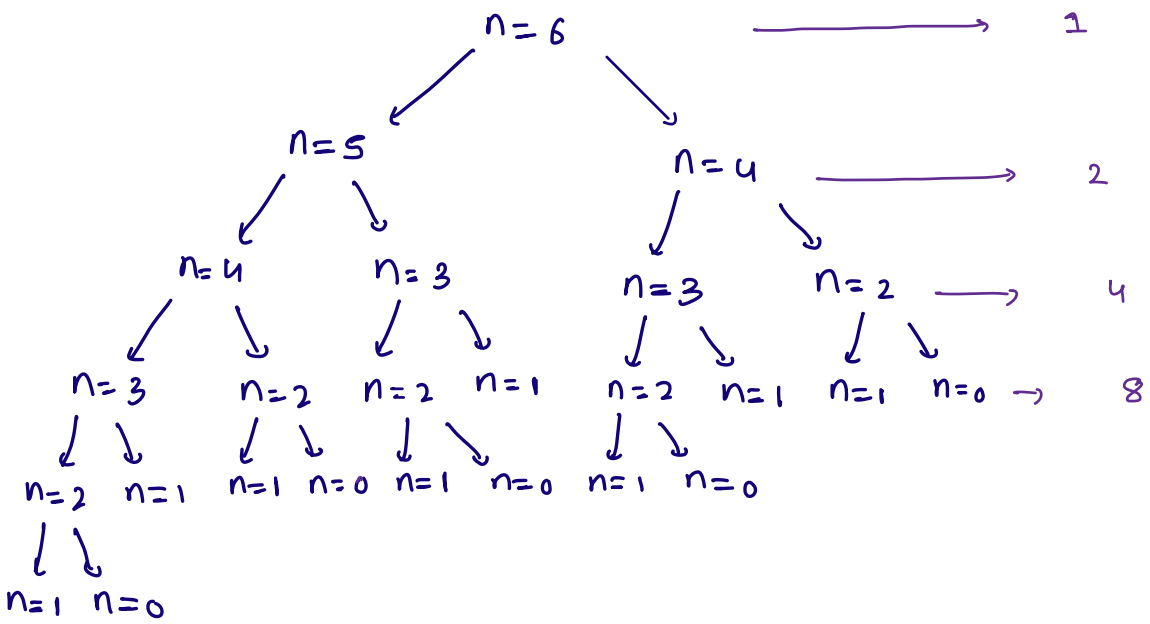
```
int fib (int n) {
    if(n==0 || n==1) {
        return n;
    }
    int temp1 = fib(n-1);
    int temp2 = fib(n-2);
    return temp1 + temp2;
}
```

TC of single function = O(1)

total no. of calls = 2^n

TC: 2^n





total calls = $1 + 2 + 4 + 8 \ldots\ldots$

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

$a = 1$

$r = 2$

$t = N$

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

```
boolean palindrome (char [] A, int s, int e) {
    if (s == e || s > e) {
        return true;
    }
    if (A[s] != A[e]) {
        return false;
    }
    else {
        boolean ans = palindrome(A, s+1, e-1);
        return ans;
    }
}
```

Tc of single function: O(1)

total no. of calls: n/2

TC: O(n)

s=0, e=9

↓

s=1, e=8

↓

s=2, e=7

↓

s=3, e=6

↓

s=4, e=5

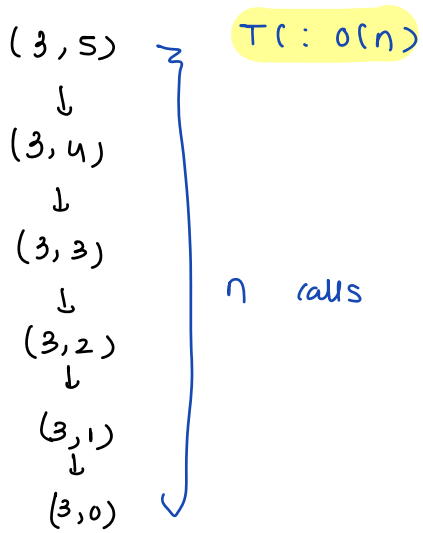
↓

s=5, e=4

```
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 calls = n

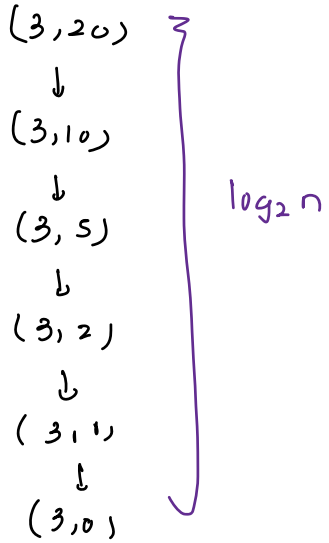



```
int pow(int a, int n) {
    if(n==0) {
        return 1;
    }
    int temp = pow(a, n/2);
    if(n%2==0) {
        return temp*temp;
    }
    else {
        return temp*temp*a;
    }
}
```

Smart

TC of single function = O(1)

total no. of calls = $\log_2 n$



TC: $\log_2 n$

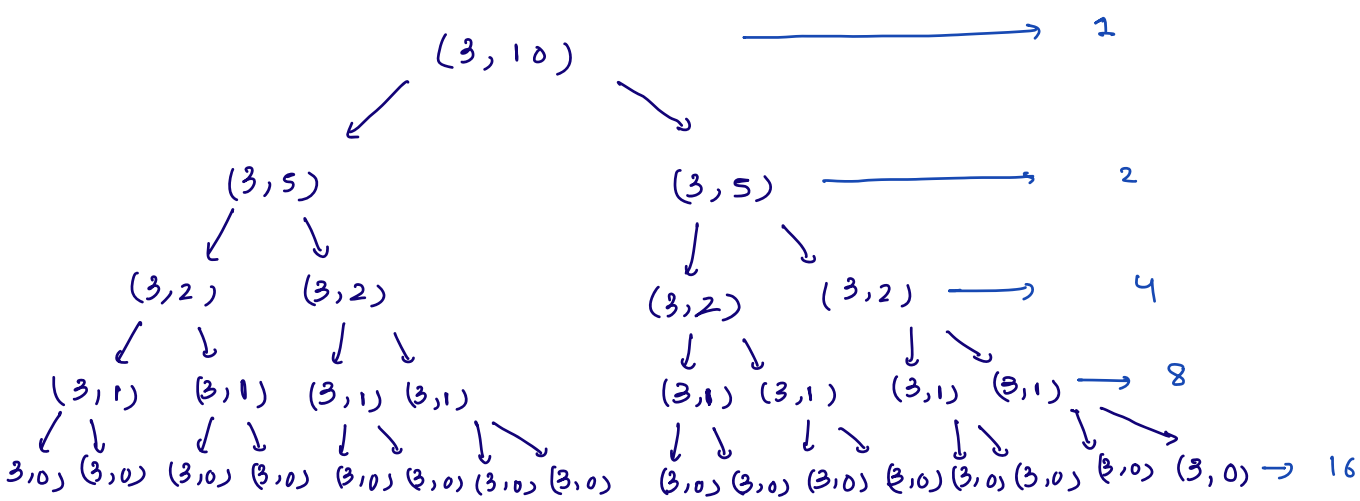
```
int pow (int a, int n) {
    if(n==0) {
        return 1;
    }
    if(n%2==0) {
        return pow(a,n/2) * pow(a,n/2);
    }
    else {
        return pow(a,n/2) * pow(a,n/2) * a;
    }
}
```

TC of single function = O(1)

total no. of calls = n

TC : O(n)

pseudo smart



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

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

$a = 1$

$r = 2$

$t = \log_2 n$

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

$$= n - 1 \approx n$$

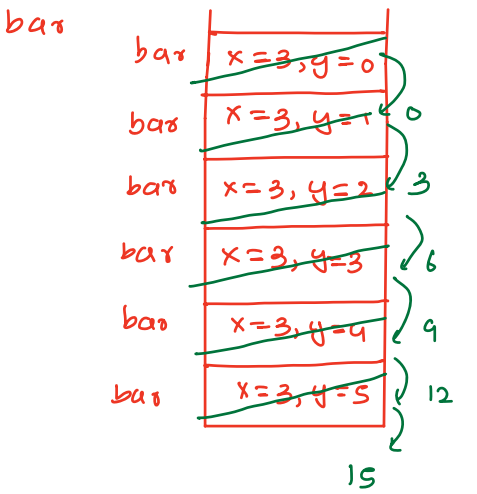
Doubts

```
#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;
    2return bar(x, foo(x, y-1));
}

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



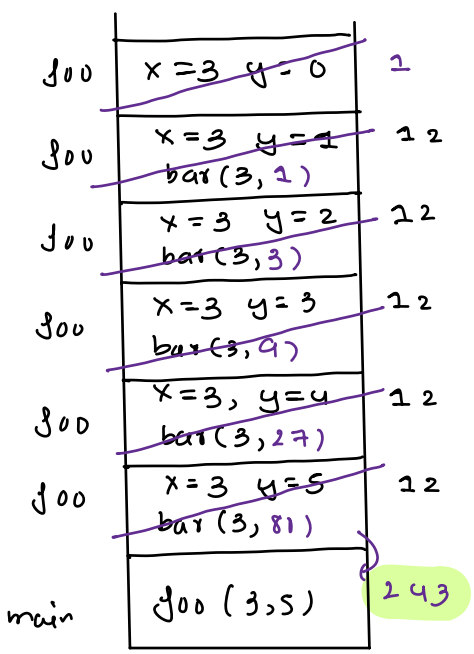
bar(x,y) => x * y

```
#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;
    2return bar(x, foo(x, y-1));
}

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



i) count total pair with sum = K

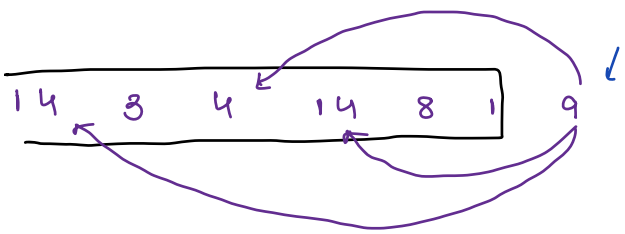
2 5 10 7 5 7 9 K = 12

K - A[i]

count = 1 + 1 + 1 + 2

2 → 1
5 → 2
10 → 1
7 → 2
9 → 1

ele vs freq (ds)



14 → 2

3 → 1

8 → 1

1 → 1

K = 5

A[i] = 9

A[i] - K ⇒ 9 - 5 = 4

A[i] + K ⇒ 9 + 5 = 14