Agenda

1) what is recursion
2) some easy questions

Why Recursion is Important:

i) Trees, graphs

ii) Backtracking, DP

Recursion: function calling itself.

$$sum(N) = 1 + 2 + 3 + \dots + N$$

$$sum(N-1) = 1 + 2 + 3 + \dots + N-1$$

$$\underbrace{sum(N)}_{\text{main problem}} = \underbrace{sum(N-1)}_{\text{Sub-problem}} + N$$

How to apply recursion?

1) **Assumption:** what the function is going to do

2) **Main logic:** solving **main problem** using just smaller **Sub-problem** of same type.

3) **Base condition:** when the recursion should stop.

```
int sum (int n) {
    if (n==1) {
        return 1;
    }

    int temp = sum(n-1);
    return temp+n;
}

void main( ) {
    int n= scn.nextInt( );
    sOPln (sum(n));
}
```

Assumption : Given n, returning sum of n natural no.

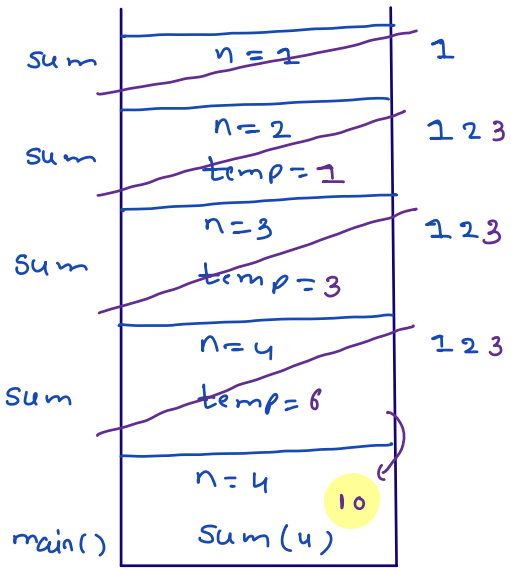
Main logic :

Sum(n) = Sum(n-1) + n

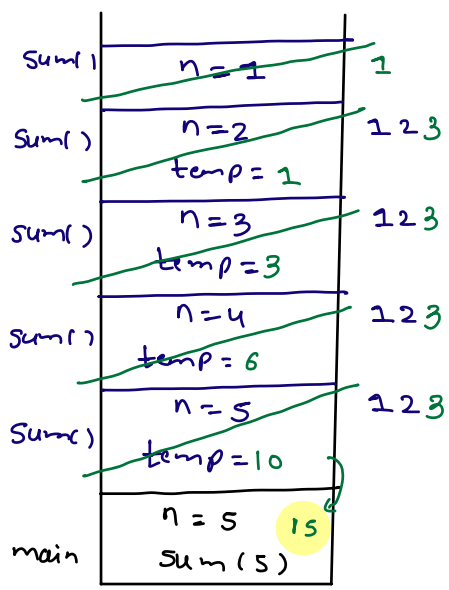
Base condition: smallest problem whose ans is known.

if(n==1) → 1

```
int sum (int n) {
    if (n==1) {
1       return 1;
    }

2   int temp = sum(n-1);
3   return temp+n;
}

void main() {
    int n= scn.nextInt();
    sopln (sum(n));
}
```



```
int sum (int n) {
    if (n==1) {
1       return 1;
    }

2   int temp = sum(n-1);
3   return temp+n;
}

void main() {
    int n= scn.nextInt();
    sopln (sum(n));
}
```



Q.2 Given n, find factorial of n.

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

Assumption : Given n, return factorial of N.

Main logic :

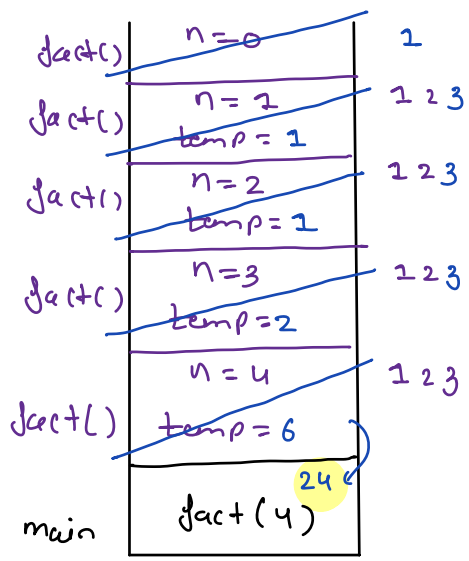
factorial (N) = factorial (N-1) * N

Base condition :

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

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

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



Q.3 Given n, find n^{th} fibonacci number (using recursion)

0	1	2	3	4	5	6	7	8	9
0	1	1	2	3	5	8	13	21	34

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

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

Assumption: Given n, find n^{th} fibonacci term.

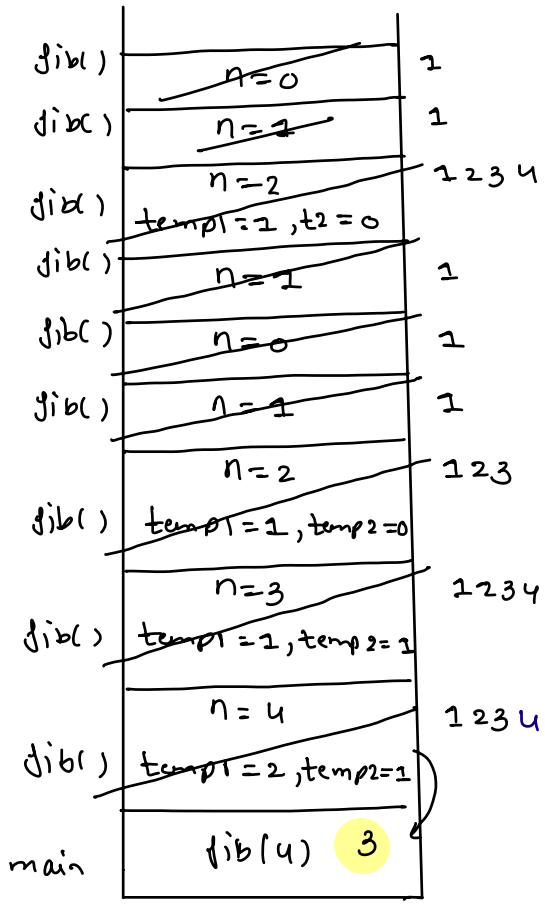
Main logic:

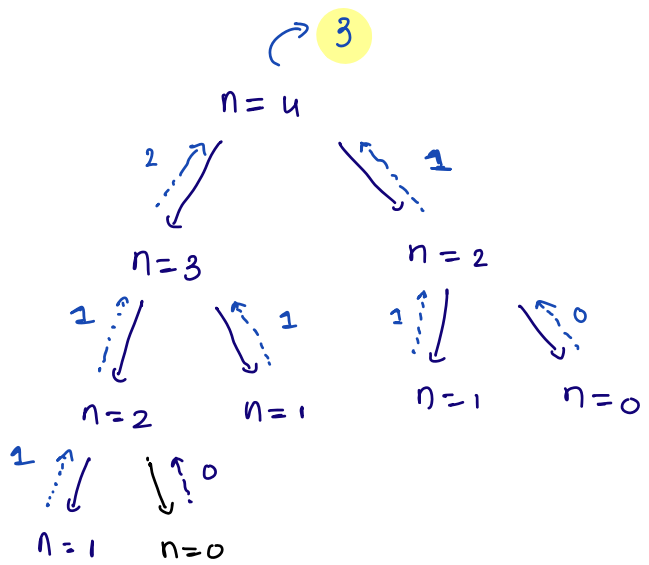
fib(n) = fib(n-1) + fib(n-2)

Base condition:

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

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

2   int temp1 = fib(n-1);

3   int temp2 = fib(n-2);

4   return temp1 + temp2;

}
```





euler diagram

Q.4 Given n, print values from 1 to n (Increasing)

n = 5, O/P: 1 2 3 4 5

n = 4, O/P: 1 2 3 4

```
void Inc (int n) {
    if (n==0) {
        return;
    }
    Inc(n-1);
    sopln(n);
}
```

Assumption: given n, print 1 to n.

Main logic:

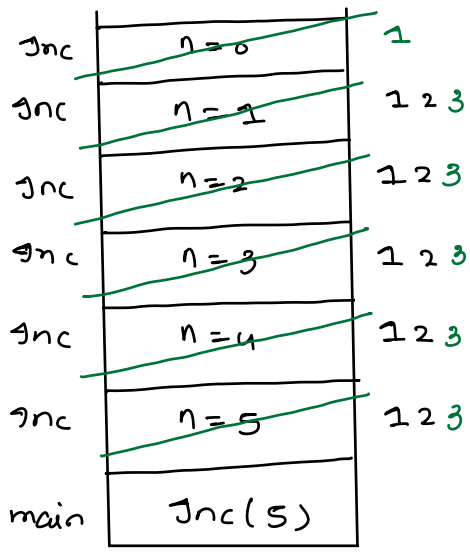
Inc(n) => Inc(n-1) + sopln(n)

Base condition:

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

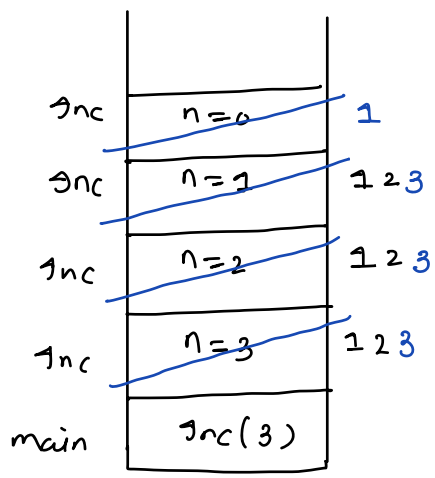
```
if(n==0){
    return;
}
```

```
void  fnc (int n) {
      if (n==0) {
1 |       return;
  }

2 fnc(n-1);
3 sopln(n);
}
```



O/P: 1 2 3 4 5

```
void  fnc (int n) {
      if (n==0) {
1 |       return;
  }

2 fnc(n-1);
3 sopln(n);
}
```



O/P: 1 2 3

```
void fun(int n) {
    if(n==0) {
        return;
    }
    sopdn(n);
    fun(n-1);
}
```

todo: trace and confirm your answer.

Q.5 Given a char[] and s, e. Find if substring from s to e is palindromic or not.

A → a b t t b c a

0 1 2 3 4 5 6

1 4 → true

2 5 → false

A →


```
if(A[s] != A[e]) {
    return false;
}
```

```
if(A[s] == A[e]) {
    pal(A, s+1, e-1);
    ans
}
```

a	b	c	b	t	b	c	d	c	b
0	1	2	3	4	5	6	7	8	9

s = 3

e = 6

```
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;
    }
}
```

a	b	c	b	a
		s		
		e		

s == e

a	b	b	a
e	s		

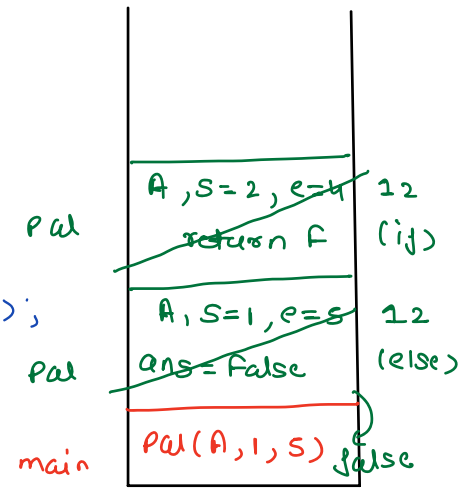
s > e

```
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;
    }
}
```

$s=1, e=5$

A = a b d c b b t

0 1 2 3 4 5 6

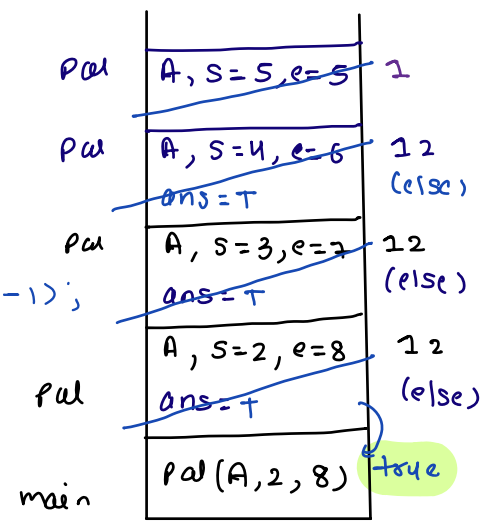



```
boolean palindrome (char [] A, int s, int e) {
    if (s==e || s>e) {
        return true;
    }
    if (A[s] != A[e]) {
        return false;
    }
    else {
        boolean ans = palindrome(A, s+1, e-1);
        return ans;
    }
}
```

$s=2, e=8$

A = a b c n i t i n c a

0 1 2 3 4 5 6 7 8 9



Doubts

Colorful number

3245 ⇒ [3, 2, 4, 5]

3 = 3	2 = 2	4 = 4	5 = 5
3 2 = 6	2 4 = 8	4 5 = 20	
3 2 4 = 24	2 4 5 = 40		
~~3 2 4 5~~			

Subarrays product with hash set.

$O(n^2)$: TC

```
for ( s=0; s<n; s++) {
    int prod = 1;
    for (e=s; e<n; e++) {
        if (s==0 && e=A.length-1) { continue; }
        prod = prod*A[e];
        if (hs.contains(prod)) {
            return false;
        }
        hs.add(prod);
    }
}
return true;
```

longest consecutive sequence

100 2 4 99 5 3 98 1

cons. seq : 1 2 3 4 5

98 99 100

↓

100 2 4 99 5 3 98 1

100 → ~~T~~ F

2 → ~~T~~ F

4 → ~~T~~ F

99 → ~~T~~ F

5 → ~~T~~ F

3 → ~~T~~ F

98 → T

1 → T

HashMap

ele vs can this ele be a starting pt. of any cons. seq.

i) travel the array and put true infront of every ele in your map.

ii) travel the array and put false infront of every invalid start pt.

if A[i]-1 is present then A[i] can't be a start pt.

iii) now travel map and the sp in front of which true is present create cons. seq. using that sp.

100 → ~~T~~ F 5 → ~~T~~ F

2 → ~~T~~ F 3 → ~~T~~ F

4 → ~~T~~ F * 98 → T

99 → ~~T~~ F * 1 → T

ans = ~~0~~ ~~3~~ 5

sp = 98 len = ~~0~~
~~1~~
~~2~~
3

```
for ( int sp : map.keySet() ) {
    if (map.get(sp) == true) {
        int len = 0;
        while (map.containsKey(sp+len)) {
            len++;
        }
        ans = Math.max(len, ans);
    }
}
```

sp = 1 len = ~~0~~
~~1~~
~~2~~
~~3~~
~~4~~
5

⇒ O(n)

2 14 3 5 9 14 5 9 14

max = 14

i) find max

ii) find freq of max

freq of max = 3

ans = n − freq of max

= 9 − 3 = 6