Agenda

1) what is recursion

2) Some basic questions

Importance

1) Trees and graphs

2) DP

Recursion : Function calling itself.

How to apply recursion

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

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

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

Solving the main problem using just smaller problem of same type.

3 magical steps to apply recursion

1) **Assumption** : what is the function doing.

sub-problem

2) **Main logic** : solving same problem using just smaller problem of same type

3) **Base condition** : when to stop recursion.

Q.1 Given N, find sum of N natural numbers.

```
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, find sum of N natural no.

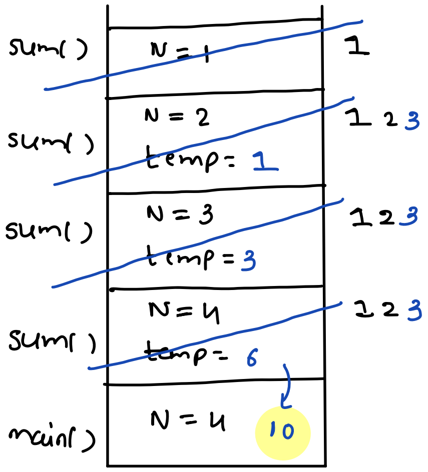
Main logic :

sum(N) = sum(N-1) + N

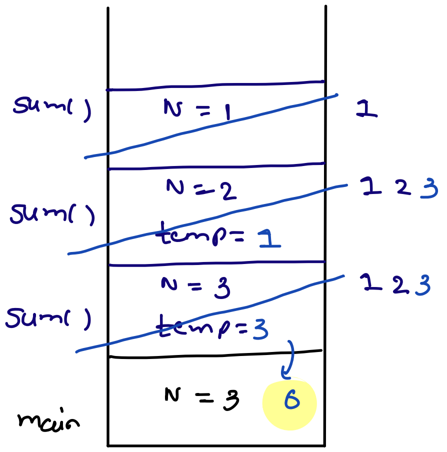
Base condition : smallest problem whose answer is known.

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, find factorial of N.

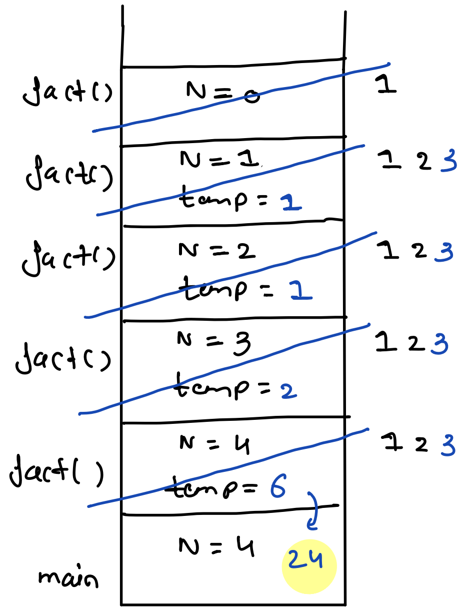
Main logic:

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

Base condition:

if (N==0) → 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.

0	1	2	3	4	5	6	7	8	....
0	1	1	2	3	5	8	13	21	

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

Assumption : Given N, find N^{th} fibonacci no.

Main logic :

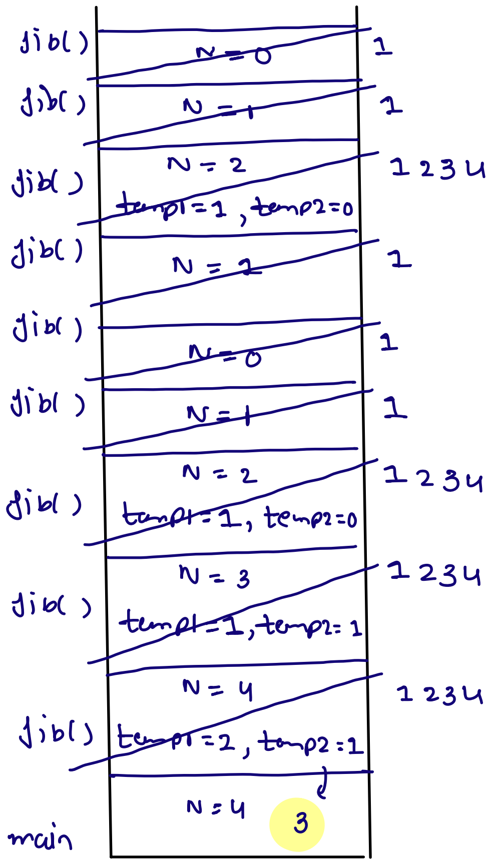
fib(N) = fib(N-1) + fib(N-2)

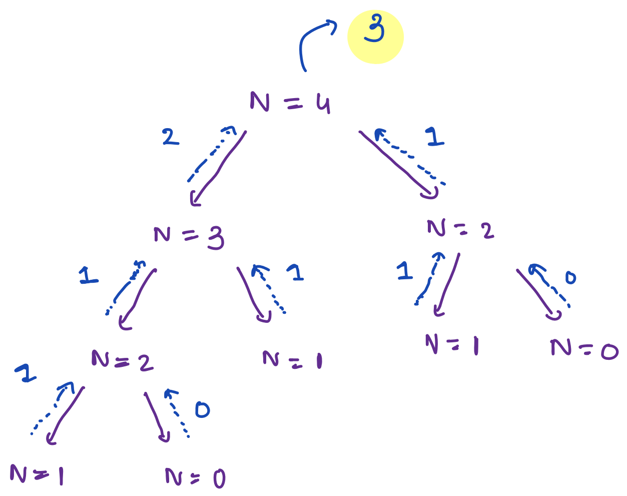
Base condition:

if (N==0) → 0

if (N==1) → 1

```
int fib (int N) {

    if (N == 0 || N == 1) {

            return N;

    }

    int temp1 = fib (N-1);

    int temp2 = fib (N-2);

    return temp1 + temp2;

}
```





Euler diagram

(cover it

in next

sessions)

Q.4 Given N, print Increasing from 1 to N.

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 increasing 1 to N.

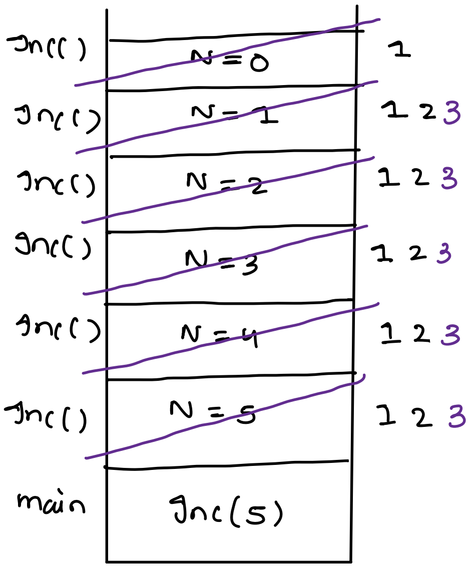
Main logic :

$\underbrace{Inc(N)}_{1 \text{ to } N} \Rightarrow \underbrace{Inc(N-1)}_{1 \text{ to } N-1} + SOPln(N)$

Base condition

if(N==0)

```
void  Inc ( int N ) {

      if (N == 0) {
1  |      return;
      }
2     Inc ( N-1 );
3     SOPln (N);
}
```



O/P : 1 2 3 4 5

```
void    fun ( int  N ) {

    if ( N == 0 ) {
        return ;
    }

    sopln ( N );

    fun ( N - 1 ) ;

}
```

do tracing yourself

Q.5 Given string, check if it is palindromic or not.

str : bata   ans = false

str : abcba   ans = true

str :

ch(s) != ch(e)
return false

ch(s) == ch(e)
if string from s+1 to e-1 is palindromic or not.

question

```
boolean palindrome (string str) {
        return check (str, 0, str.length()-1);
}
```

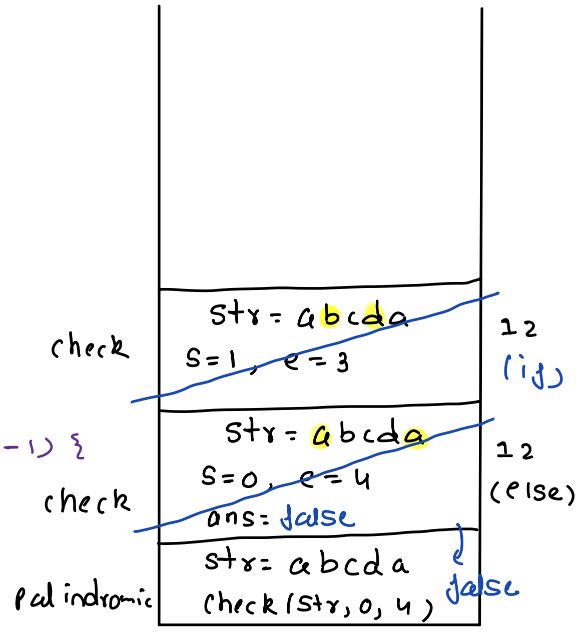
helper

```
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;
    }
}
```

```
a  b  c  b  a          a  b  b  a
0  1  2  3  4          0  1  2  3
      s                   e  s
      e
```

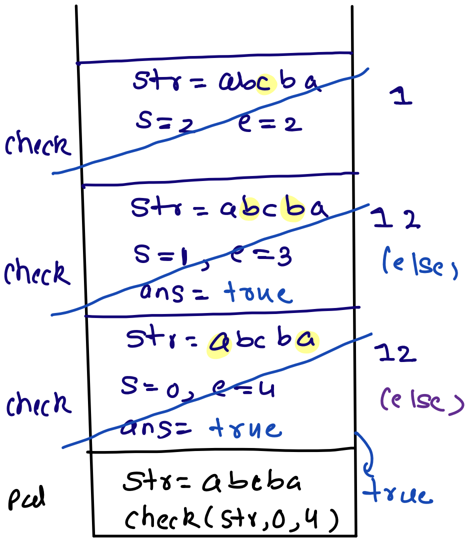
```
if ( s==e || s > e) {
        return true;
}
```

```
boolean check (string str, int s, int e) {

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



str: a b c d a
0 1 2 3 4

```
boolean check (string str, int s, int e) {

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



str: a b c b a
0 1 2 3 4

Doubts

longest consecutive sequence

A = 100 1 3 15 99 97 16 2 4 5 98

Cons. Seq: 1 2 3 4 5

97 98 99 100

15 16

idea1: using sorting

```
Arrays.sort(A);    // nlogn
```

$$A = \underbrace{1 \quad 2 \quad 3 \quad 4 \quad 5}_{5} \quad \underbrace{15 \quad 16}_{2} \quad \underbrace{97 \quad 98 \quad 99 \quad 100}_{4}$$

A = 100 1 3 15 99 97 16 2 4 5 98

100 → ~~T~~ F

1 → T

3 → ~~T~~ F

15 → T

99 → ~~T~~ F

97 → T

16 → ~~T~~ F

2 → ~~T~~ F

4 → ~~T~~ F

5 → ~~T~~ F

98 → ~~T~~ F

HashMap

ele vs are you starting pt. of any sequence

i) In map put A[i] vs true

ii) put false infront of those A[i]'s which can't be start of your seq.

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

(iii) travel the map and try to create ans from real start points.

```
for( int sp : map.keySet( ) ) {

    if ( map.get(sp) = = true) {
        // sp is a correct starting point
        int len = 0 ;
        while (map.containsKey(sp+len)) {
                len++;
        }
        ans= Math.max(ans, len);
    }
}
```

O(n)

100 → ~~T~~ F

1 → T

3 → ~~T~~ F

15 → T

99 → ~~T~~ F

97 → T

16 → ~~T~~ F

2 → ~~T~~ F

4 → ~~T~~ F

5 → ~~T~~ F

98 → ~~T~~ F

sp= 1

len=0 ,	1+0	⇒ T
len= 1,	1+1	⇒ T
len= 2,	1+2	⇒ T
len=3,	1+3	⇒ T
len=4,	1+4	⇒ T
len= 5,	1+5	⇒ F

ans= 5

sp= 15

len=2

(15, 16)

sp= 97

len = 4

(97, 98, 99, 100)

A = 2 3 4 1 5 7 10 8 9

2 → ~~T~~ F

3 → ~~T~~ F

4 → ~~T~~ F

1 → T

5 → ~~T~~ F

7 → ~~T~~ F

10 → ~~T~~ F

8 → ~~T~~ F

9 → ~~T~~ F

sp = 1

len=10

$\boxed{\text{ans}=10}$