Recursion

function calling itself.

when a function calls itself directly or indirectly

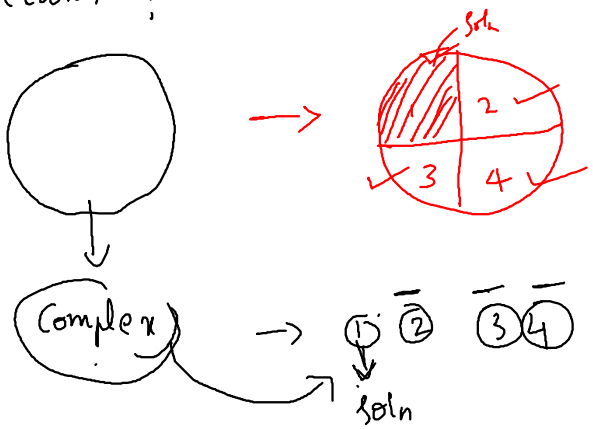
```
void update(){
}
int main(){
  update();
}
```

normal function call.

```
void updak(){
  updak();
}
```

recursion

Why recursion ?



Ek problem
bachi kee mai kardungi

→ Ek badi problem soln → same type of choti problem

Problem: Finding 5!

5! => 5X4X3X2X1

=> 5X4!

bigga problem

n! = n x (n-1) !

$F(n) = n \times f(n-1)$ → smaller problem

Bigger problem

$5! \Rightarrow 5 \times 4!\ (24) \Rightarrow 120$ //

$4! \Rightarrow 4 \times 3!\ (6) = 24$

$3! \Rightarrow 3 \times 2!\ (2) = 6$

$2! = 2 \times 1!\ = 1\ (2)$

$1! = 1$

Example:

4^n

$n \Rightarrow 5$

$4^5 \Rightarrow 4 \times 4 \times 4 \times 4 \times 4$

$4^5 \Rightarrow 4 \times 4^4$

$4^n \Rightarrow 4 \times 4^{(n-1)}$

$F(n) = 4 \times f(n-1)$ — Recursive relation.

```
void display(){
   cout<<"Recursion"<<endl;
  display();
}
int main(){
  display();
}
```

output
Recursion
Recursion
Recursion
⋮

→ Segmentation fault

?

⇒ When to stop

infinite loop

1) n to 1

```
n=5
print( ){
    print(n-1) }

print(n)
```

5 → n=5
4 n=(n-1) = 4
3 n-1 ⇒ 3
2 n-1 ⇒ 2
1 n-1 = 1

```
void print(int n){
    cout<<n<<endl;
    print(n-1);
}
int main(){
 int n=5;
 print(n);
}
```

```
void print(int n){
    cout<<n<<endl;
    print(n-1);
}
```

```
void print(int n){
    cout<<n<<endl;
    print(n-1);
}
```

```
void print(int n){
    cout<<n<<endl;
    print(n-1);
}
```

```
void print(int n){
    cout<<n<<endl;
    print(n-1);
}
```

output

5
4
3
2
1
0
-1
...

↳ segmentation fault.

Base case / condition
stop recursion
① ✕ return mandatory

for(i, ⓒ, i/d)
{
}

Base condition
if(n==0) return;
≡ ✕

```
void print(int n){
    //base condition
    if(n==0) return;
    cout<<n<<endl;
    print(n-1);
}
int main(){
 int n=5;
 print(n);
}
```

```
void print(int n){
    //base condition
    if(n==0) return;
    cout<<n<<endl;
    print(n-1);
}
```

```
void print(int n){
    //base condition
    if(n==0) return;
    cout<<n<<endl;
    print(n-1);
}
```

```
void print(int n){
    //base condition
    if(n==0) return;
    cout<<n<<endl;
    print(n-1);
}
```

```
void print(int n){
    //base condition
    if(n==0) return;
    cout<<n<<endl;
    print(n-1);
}
```

```
void print(int n){
    //base condition
    if(n==0) return;
    cout<<n<<endl;
    print(n-1);
}
```

5
4
3
2
1

Base

* 2 cheezen
  ↳ Base Condition → Rukna
  ↳ Recursive relation → calling itself.

Example : 1 to 6

6 > Stop

1 (n+1)
2
3
4
5
6

```
void print(int n){
    //base condition
    if(n>6) return;
    cout<<n<<endl;
    print(n+1);
}
```

components for recursion

1) Base condition (return)
2) Processing
3) Recursive relation

(1)

1) BC
2) P
3) RR → Tail recursion

(2)

1) BC
2) RR → Head recursion.
3) P

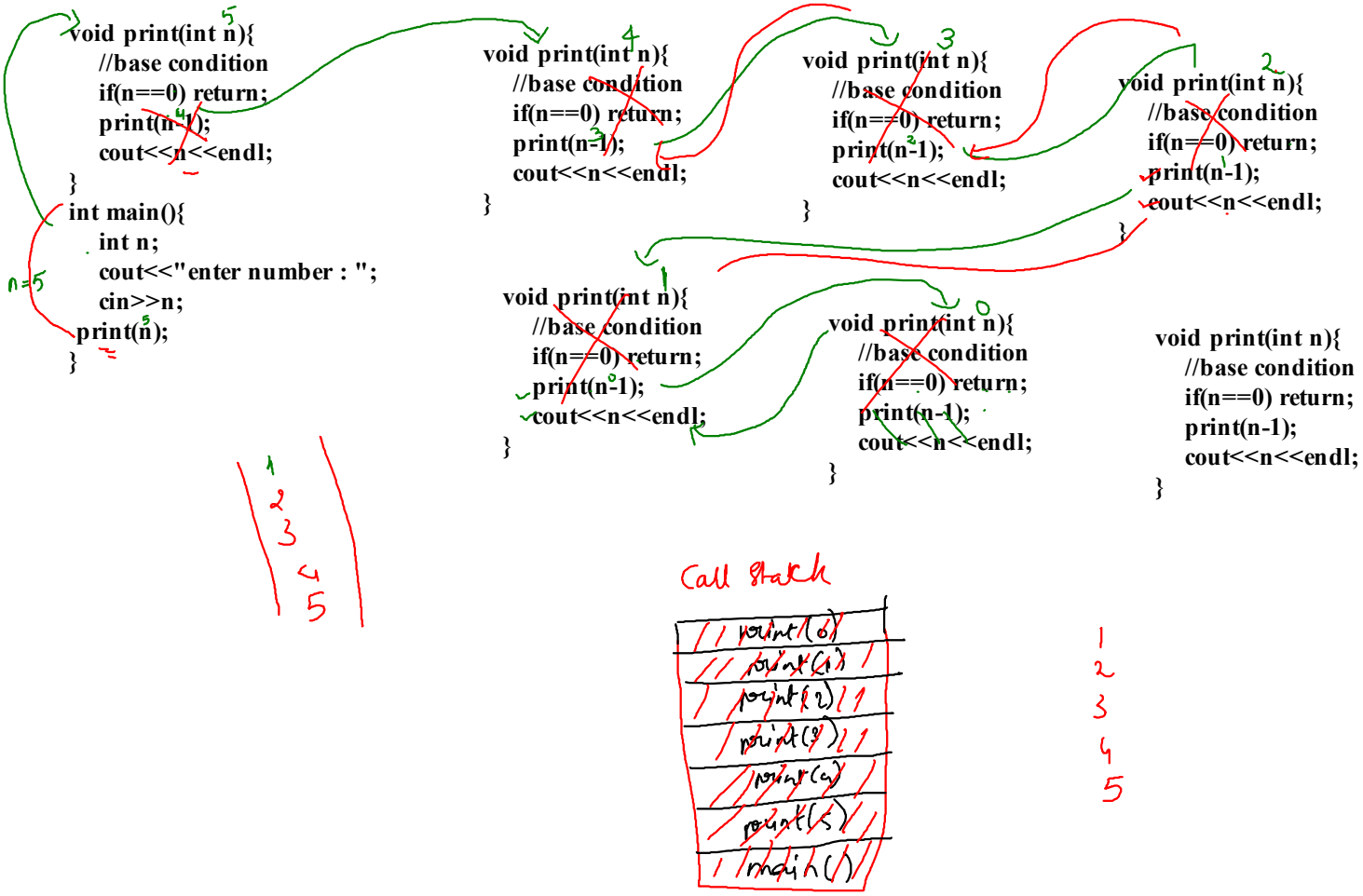
Example : Head recursion 1 to n

BC (n==0)
RR → (n-1)
Processing . . n

n=5

1
2
3
4
5

(n-1)



Factorial of number

(n==0) return 1;

n=5

5! => 5X4X3X2X1

=> 5X4!

f(n) => n x f(n-1)

5 x f(4) = 120

4 x f(3)

3 x f(2)

2 x f(1)

1 x f(0)

```
int factorial(int n){
    if(n==0) return 1;
    int fact=n*factorial(n-1);
    return fact;
}
int main(){
    int n;
    cout<<"enter number : ";
    cin>>n;
    int result=factorial(n);
    cout<<"Result is "<< result;
}
```

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

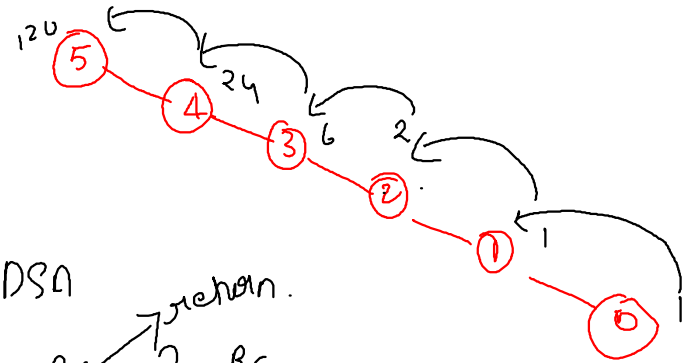
```
int factorial(int n){
    if(n==0) return 1;
    int fact=n*factorial(n-1);
    return fact;
}
```

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

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

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

-> Recursion Tree



Recursion

DSA

BC

RR

recursion.

BC

RR -> Head

P

BC

P

RR -> Tail Recursion