Recursion (Basic)

1) fn calling itself directly or indirectly.

2) 2 Parts → Base case.
→ Recursive call.

3) Every recursive problem can be solved with iteration and vice versa.

4) Multiple Activation records are formed for recursion due to fn calls.

<)

Disadv

→ Uses more space and memory.
→ Also has greater time req.

Adv

→ Is clean and simple.

Basic 1 Finding a number.

```
bool recursiveSearch(int arr[],
                     int l, int r,
                     int x)
{
  if(r<l) { return false; }
  if(arr[l]==x) { return true; }
  if(arr[r]==x) { return true; }
  return recursiveSearch(arr,
                         l+1, r-1,
                         x);
}
```

Basic 2 Palindrome

```
bool isPalindrome(char str[], int s, int e)
{
  if(s==e) { return true; }
  if(str[s] != str[e]) { return false; }
  if(s<e) { return isPalindrome
                   (str, s+1, e-1); }
  return true;
}
```

7) Tail Recursion

→ when the recursive call is at the end of the fn. and doesn't use the current fn anymore. } Easier for compiler to correct.

Eg:- Factorial

```
fact(int N, int a)
{ if(N==0) return a;
  return fact(N-1, N*a);
}
```

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

Tail

Non-Tail.

8) Application

→ Dynamic Prog
Backtracking
Divide and Conquering (Quick Sort, Merge Sort) Binary Search. etc.

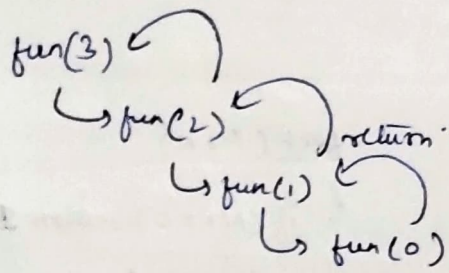
Inherently Recursive, → Tower of Hanoi
Traversals
DFS / BFS
Johnpens problem.

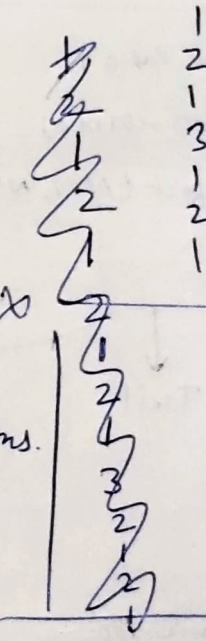
Problems

Guess the output →

```
void fun(int n)
{ if(n==0) return;
  fun(n-1);
  cout<<n<<endl;
  fun(n-1);
}
int main(){
   fun(3);
   return 0;
}
```

→





Always try to draw a tree to solve recursion problems.

① Print n to 1 using recursion.

```
Print_n_to_1(int n){
    if(n==0){return;} ——→ Base case.
    Print(n);
    Print_n_to_1(n-1);
}
```

Space → O(n) but if we use tail recursion i.e use modern, it can reduce to compiler O(1).

② Natural numbers sum.

First n natural num sum.

```
FirstN(int n)
{
    if(n==0) return 0;
    return n + FirstN(n-1);
}
```

③ Find nth Fibonacci number (where n ≥ 0).

```
int a
Fibo(int n){
if(a==n) return a;
a = fib(a-1) + fib(a-2);
}
```

(6M, 2/6)

→ if(n==1) return 1;

My trial.

Ans

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

④ Palindrome already discussed.