(2) How to write recursion code
(3) How recursion works
(4) TC/SC

→ Merge Sort and Quick Sort
→ Binary Trees, BST's, Segment Trees, Tries
→ Dynamic Programming
→ Backtracking

Recursion is not a data structure or algorithm, it is a way of writing code.

recursion: → recursion

Find the sum of first n natural numbers (n>0).

sum(5) = 15 # 5+4+3+2+1
sum(n) = 1+2+3+...+n-1+n
= sum(n-1) + n

sum(5) = sum(4) + 5 → 15
sum(4) → 10

→ same problem

sum(n) = sum(n-1) + n
→ smaller version of same problem (Subproblem)

Steps -

→ Assumption : What your function does
→ Base case : Terminating step
→ Main logic : Recursion step
self-work

int sum (int n) {
// Assumption :- sum(n) returns the sum of first n nos. V n>0
// Base case
if (n==1)
return 1;
// Main logic
return sum(n-1) + n ;
recursion step
self-work
}

Find factorial of a number n. (n>=0).

n! = 1*2*3*4*...n!
3! → 6, 5! → 120
0! → 1
n! = (n-1)! * n
fact(n) = fact(n-1) * n

int fact (int n) {
// Assumption :- fact(n) gives the factorial of n V n>=0
// Base case
if (n==1)
return 1;
// Main logic
return n* fact(n-1);
}

n = [C], 1, 2, 3, 4, 5, 6, 7

Find out the nth fibonacci number

Fib series = 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89
0 1 2 3 4 5 6 7 8 9 10
fib(1)=1 fib(2)=1
fib(3)=2
fib(4)=3
fib(5)=5
fib(6)=8
fib(7)=13
fib(8)=21
fib(9)=34
fib(10)=55
fib(n) = fib(n-1) + fib(n-2)
→ n>0.

int fib (int n) {
// Assumption :- fib(n) = nth fibonacci number
// Base case
if (n<=2)
return 1;
// Main logic
return fib(n-1) + fib(n-2);
}

int add(a,b) {
return a+b;
}
int sub (a,b) {
return a-b;
}
int mul (a,b) {
return a*b;
}

int main (a,b) {
int x = add(a,b);
int y = mul(x,b);
int z = sub (y,b);
return z;
}

stack frame
main
Memory stack

Tutorial frame :- usually ignore

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

Dry run :- sum(5)

→ 10

Recursion infinitely.

Dry run fact(4) :-

int {
a = f(b) + f(c)
}

(n-1)!
return fact(n-1)*n;
n*fact(n-1);

f(n-1) + f(n-2)
f(n-2) + f(n-3)

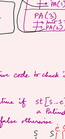
H(n) n=5 → How many recursion calls that fib(5)
Approx !

Q.1) Print numbers from 1 to n in increasing order using recursion.

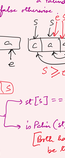
n = 5 :-
1 2 3 4 5

void PrintInc(n)
→ PrintInc(n-1) — 1
→ print(n) — 2
→ PrintInc(5)
→ PrintInc(4)
→ 5

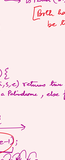
void PrintInc(int n)
// Assumption :- PrintInc(n) prints all no. 1 to n in increasing order.
// Base case
if (n==1){
print(1); return ;}
// Main logic
PrintInc(n-1);
print(n);
}



Q.2) Print all no. 1 to n in decreasing order.



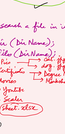
void PrintDec(int n)
// Assumption :- PrintDec(n) prints all no. 1 to n in decreasing order.
// Base case
if (n==1){
print(1); return ;}
// Main logic
print(n);
PrintDec(n-1);
}



Q.3) Given a string st, write a recursion code to check if it is a Palindrome.

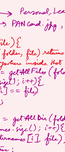
isPalin(st, s, e) → true if st[s..e] is a Palindrome
false otherwise.

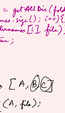
→ isPalin(st, 0, n-1)



→ st[s] == st[e]
→ isPalin(st, s+1, e-1)
Both have to be true.

Bottom :- isPalin(st, s, e) {
// Assumption :- isPalin(st, s, e) returns true if st[s..e] is Palindrome, else false.
// Base case :- st[s..e] is Palindrome, when s ≥ e.
return true;
→ if (st[s] != st[e])
return false;
// Main logic :- return isPalin(st, s+1, e-1);
}





Google, FB

Q.4) Given a directory structure, search a file in it.

Given :-
int <string> getAllDir (Dir Name);
int <string> getAllFile (Dir Name);

getAllDir (root) → Personal, Learning
getAllFile (root) → PPM Code Mgt, Office Mgt

bottom search (folder, file) {
// Assumption :- bool (folder, file) returns true if folder is located somewhere inside that folder.
int <string> filenames = getAllFile (folder);
for (int i=0; i<filenames[]; i++){
if (filenames[i] == file)
return true;
}
int <string> dirnames = getAllDir (folder);
for (int i=0; i<dirnames[]; i++){
if (search (dirnames[i], file))
return true;
}
return false ;
}

Base case
Main logic

dirnames → [A, B, C]

False ← ret search (A, file);