Todays Content

→ Recursion

→ How to write recursive code / trace

→ TC / SC of recursive code in Next class.

Why Recursion ?

→ Merge Sort / Quick Sort

→ Binary Tree / BST / Balanced BST / Segment Trees / Tries

→ Dynamic Programming

→ Back tracking

→ Graphs

Recursion ? ⇒ Function calling itself

↳ Solving a problem using smaller instance of same problem

⇓

sub-problem

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

$$\boxed{sum(N) = sum(N-1) + N}$$

$$sum(4) = sum(3) + 4$$

↳ sub-problem

Qn: How to write recursive code?

Assumption: Fix what your function should do

Main Logic: Solve the assumption using the problem

Base Condition: Inputs for which you want to stop the recursion.

Problems

$(N >= 1)$

```
int sum(N) {
    if(N==1) {return 1}

    return sum(N-1) + N
}
```

Assumption: Given N, calc & return the sum of first N natural numbers.

Main Logic:

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

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

$sum(1) = sum(0) + 1$
$= 1.$

Qn: fact(N) ? (N >= 1)

fact(3) = 3x2x1 = 6 , fact(4) = 4x3x2x1 = 24.

```
int fact(N) {
    if(N==1) {return 1}

    return N x fact(N-1)
}
```

Assumption: Given N, find & return N!

Main Logic:

fact(N) = N x N-1 x N-2 x ... x 3 x 2 x 1

fact(N) = N x fact(N-1)

Function Call Tracing

```
int add (N, M) {
    return N+M
}

int mul (x, y) {
    return x*y
}

int sub(a, b) {
    returns a-b
}
```

```
main() {
    x = 10, y = 20
    print(sub(mul(add(x,y), 30), 75))
}
```

sub(mul(add(x,y), 30), 75) : returns 825 (900)

↳ mul(add(x,y), 30) : return 900 (30)

↳ add(x,y) : return 30 (10, 20)

Output: 825

// Data Structure

LIFO structure

Last-In
First Out

add(x, y) : returns 30 [remove]

mul(add(x, y), 30) : returns 900 [remove]

sub(mul(add(x, y), 30), 75) : returns 825 [removed]

DS: Stack
↳ System handles the implementation

Sum Tracing

returns 10

```
int sum(N=4) {
    if (N==1) { return 1 }
    return sum(N-1) + N
}
```

6 + 4

```
int sum(N=3) {
    if (N==1) { return 1 }
    return sum(N-1) + N
}
```

3 + 3

```
int sum(N=2) {
    if (N==1) { return 1 }
    return sum(N-1) + N
}
```

1 + 2

```
int sum(N=1) {
    if (N==1) { return 1 }
    return sum(N-1) + N
}
```

Stack

sum(1) :	return 1
sum(2) : return	sum(1) + 2 (1+2)
sum(3) : return	sum(2) + 3 (3+3)
sum(4) : returns	sum(3) + 4 = 10

What happens if base condn not there?

a) TLE

b) MLE ✓

Note: In recursion, if you get MLE, verify base condition.

Qn: $N >= 0$ (Sum of prev 2 numbers) : Fibonacci

# input (N)	0	1	2	3	4	5	6	7	8	9	10
Fib( )	0	1	1	2	3	5	8	13	21	34	55

fib(5) = 5

fib(9) = 34

Assumption: Given N, Calc & return N^{th} fibonacci no.

```
int fib(N) {
    if(n<=1) {return n}

    return fib(N-1) + fib(N-2)
}
```

Main Logic

fib(N) = sum of prev 2 terms

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

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

Note: How to figure out base condition.

↳ For which valid inputs, expression is invalid.

fib(0) = fib(-1) + fib(-2) ✗

fib(1) = fib(0) + fib(-1) ✗

fib(2) = fib(1) + fib(0) ✓

// Trace out:

```
int fib(N=4)
  if(N<=1) {return N}
  return fib(N-1) + fib(N-2)
}
```

→ returns 3

2 1

```
int fib(N=2)
  if(N<=1) {return N}
  return fib(N-1) + fib(N-2)
}
```

1 0

```
int fib(N=1)
  if(N<=1) {return N}
  return fib(N-1) + fib(N-2)
}
```

```
int fib(N=0)
  if(N<=1) {return N}
  return fib(N-1) + fib(N-2)
}
```

```
int fib(N=3)
  if(N<=1) {return N}
  return fib(N-1) + fib(N-2)
}
```

1 1

```
int fib(N=2)
  if(N<=1) {return N}
  return fib(N-1) + fib(N-2)
}
```

1 + 0

```
int fib(N=1)
  if(N<=1) {return N}
  return fib(N-1) + fib(N-2)
}
```

```
int fib(N=1)
  if(N<=1) {return N}
  return fib(N-1) + fib(N-2)
}
```

```
int fib(N=0)
  if(N<=1) {return N}
  return fib(N-1) + fib(N-2)
}
```

* Try out stack tracing ⇒ TODO

Break till: 8:30am

Qn: Given N, print all numbers from 1 to N in increasing order

inc(3) → 1 2 3

inc(4) → 1 2 3 4

```
void Inc(N) {
    if(N==1) {print(1), return;}
    Inc(N-1)
    print(N)
}
```

Assumption: Given N, print all numbers from 1 to N in inc. order.

Main Logic:

Inc(N): 1 2 3 4 ... N-1 N

↓

Inc(N-1)

print(N)

Trace:

```
void Inc(N=4)
    if(N==1) {print(1), return}
    Inc(N-1)
    print(N)
}
```

```
void Inc(N=3)
    if(N==1) {print(1), return}
    Inc(N-1)
    print(N)
}
```

```
void Inc(N=2)
    if(N==1) {print(1), return}
    Inc(N-1)
    print(N)
}
```

```
void Inc(N=1)
    if(N==1) {print(1), return}
    Inc(N-1)
    print(N)
}
```

Output:

print(1)

print(2)

print(3)

print(4)

* **Note:** Once the function is completely executed, it'll go back to the parent function.

* Stack trace ⇒ To do

* We can return from a void fn.

Qn: Given N, print all numbers from N to 1 (dec. order)

dec(N)

H.W.

Qn: Given a substring, check if its a palindrome or not.

Eg:

0 1 2 3 4 5 6

good dad

$s = 4, e = 6 \Rightarrow$ true

$\boxed{s <= e}$

0 1 2 3 4 5 6

good dad

$s = 2, e = 5 \Rightarrow$ false

```
bool isPal (char ch[], int s, int e) {
    if(s > e) { return true }
    return (ch[s] == ch[e] && isPal(ch, s+1, e-1))
}
```

Assumption: Return if substring [s e] is a palindrome.

Main Logic:

ch[ ]: _ _ _ _ _ _ _ _

s, e, s+1, e-1

① if(ch[s] == ch[e])

② substring [s+1, e-1] should be a palindrome

isPal(ch, s+1, e-1)

s, e ✓

m a d a m

m a l y a l a m

0	1	2	3	4
m	a	d	a	m

s = 0, e = 4

return true

```
bool isPal(ch[], s=0, e=4)
{
    if(s>e) {return true}
    return (ch[s] == ch[e] && isPal(ch, s+1, e-1))
}
```

T

T

```
bool isPal(ch[], s=1, e=3)
{
    if(s>e) {return true}
    return (ch[s] == ch[e] && isPal(ch, s+1, e-1))
}
```

T

T

```
bool isPal(ch[], s=2, e=2)
{
    if(s>e) {return true}
    return (ch[s] == ch[e] && isPal(ch, s+1, e-1))
}
```

T

T

```
bool isPal(ch[], s=3, e=1)
{
    if(s>e) {return true}
    return (ch[s] == ch[e] && isPal(ch, s+1, e-1))
}
```

Input:

0	1	2	3	4	5	6
a	n	m	e	t	n	a

s = 0, e = 6

return false

```
bool isPal(ch[], s=0, e=6)
{
    if(s>e) {return true}
    return (ch[s] == ch[e] && isPal(ch, s+1, e-1))
}
```

T

F

```
bool isPal(ch[], s=1, e=5)
{
    if(s>e) {return true}
    return (ch[s] == ch[e] && isPal(ch, s+1, e-1))
}
```

T

F

```
bool isPal(ch[], s=2, e=4)
{
    if(s>e) {return true}
    return (ch[s] == ch[e] && isPal(ch, s+1, e-1))
}
```

F