21/12/24

D D M M Y Y Y Y

* Section 5: Recursion

46. How Recursion works (Tracing)

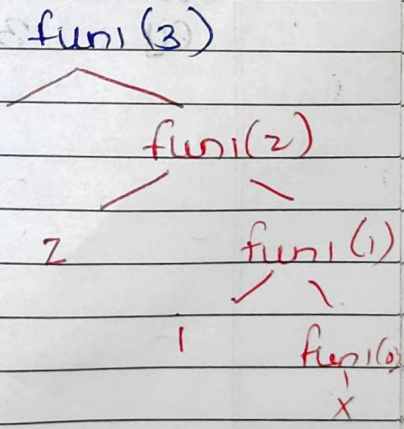
* A function which calls itself is called Recursion

```
Type fun (parameter) {
    if (<base condition>) {
        ...
        fun(parameter);
        ...
    }
}
```

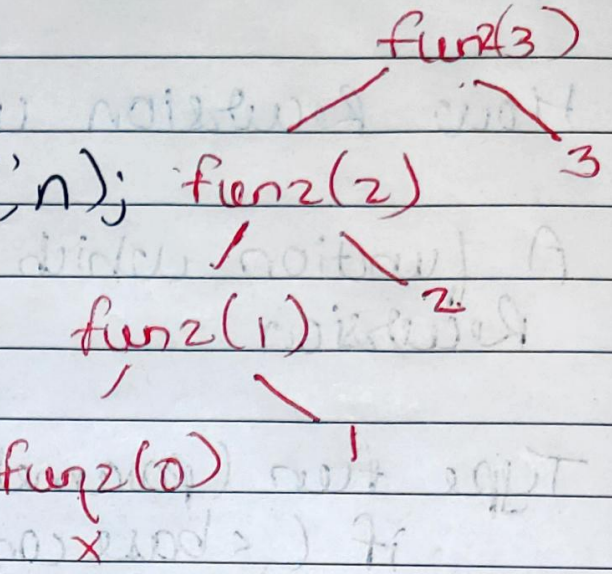
Example:

1)
```
void fun1 (int n){
    if(n>0){
        printf("%d", n);
        fun1 (n-1);
    }
}
void main(){
    int x = 3;
    fun1(x);
}
```

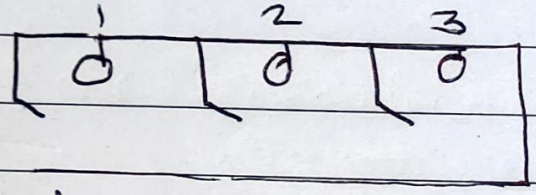
Output: 3 2 1



```
void fun2(int n){
    if (n>0){
        fun2 (n-1);
        printf ("%d", n);
    }
}
void main(){
    int x = 3;
    fun2(x);
}
```



Output : 1 2 3



1. Switch on bulb
2. go to next Room.

eg : 1 2 3

1. got to next Room
2. Switch on bulb

eg : 3 2 1

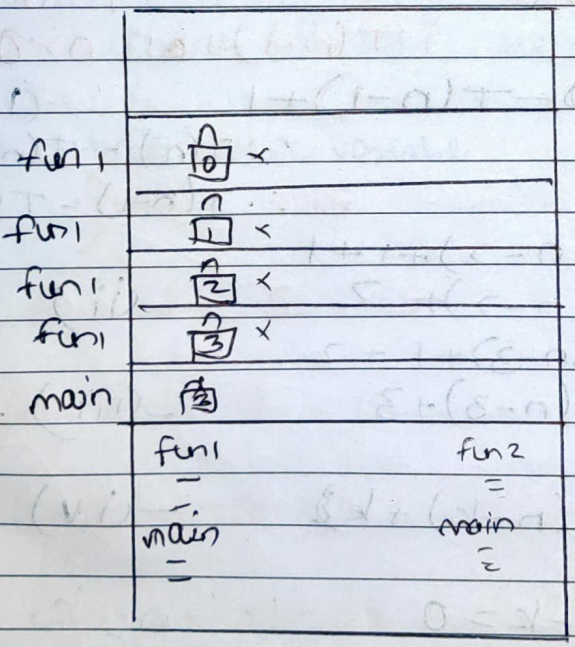
17. Generalising Recursion

```
void fun (int n){
    if (n>0) {
        calling          → Ascending
        fun(n-1) *2;
        Returning        → Descending
    }
}
```

Loop only has Ascending phase whereas Recursion has Ascending and Descending phase

18. How Recursion uses Stack

* First example



Calling n+1 → O(n)

49. Recurrence Relation - Time complexity of Recursion

```
T(n) - void fun1(int n){
1    -    if(n>0){
1    -    ⇒   printf("%d", n);
T(n-1) -      fun1(n-1);
T(n) = T(n-1)+2 }
    }
    void main(){
        int x=3;
        fun1(x);
    }
```

fun1(3)
fun1(2) - 3 - 1
fun1(1) - 2 - 1
fun1(0) - 1 - 1
3 units

n units
O(n)

Time complexity : O(n)

Recurrence Relation $T(n) = \begin{cases} 1 & n=0 \\ T(n-1)+1 & n>0 \end{cases}$

$T(n) = T(n-1)+1 \quad -(i)$

$\therefore T(n) = T(n-1)+1$

$\therefore T(n-1) = T(n-2)+1$

$\therefore T(n) = T(n-2)+1+1$

$T(n) = T(n-2)+2 \quad -(ii)$

$T(n) = T(n-3)+1+2$

$\therefore T(n) = T(n-3)+3 \quad -(iii)$

$\vdots$

$\therefore T(n) = T(n-k)+k \quad -(iv)$

Assume $n-k=0$ $\quad \therefore n=k$

$T(n) = T(n-n) + n$

$T(n) = T(0) + n$

$\therefore T(n) = 1 + n \Rightarrow O(n)$

∴

5. Static and global variables in Recursion

```
int fun (int n) {
    if (n>0) {
        x++;
        return fun(n-1)+x;
    }
    return 0;
}
main() {
    int a = 5;
    printf("%d", fun(a));
}
```

Heap: 0, 1, 2, 3, 4, 5

Stack: 5

Code: fun, main

x: 0

// with static

x: 0 + 2 3 4 5

fun(5) = 25

// without static variable

fun(5) = 15

fun(4) + 5 = 15

fun(3) + 4 = 10

fun(2) + 3 = 6

fun(1) + 2 = 3

fun(0) + 1 = 1

0

fun(4) + 5 = 25

fun(3) + 5 = 20

fun(2) + 5 = 15

fun(1) + 5 = 10

fun(0) + 5 = 5

0

53. Tail Recursion

* Types of Recursion

1. Tail Recursion
2. Head Recursion
3. Tree Recursion
4. Indirect Recursion
5. Nested Recursion

* Tail Recursion

```
void fun(int n){
    if(n>0){
        printf("%d",n);
        fun(n-1);
    }
}
fun(3);
```

→ Tail Recursion
function calls itself at last

3 2 1

Space complexity : O(n)
Time complexity : O(n)

* Comparing loop to Recursion

It can be easily converted to loops directly

```
void fun(int n){
    while(n>0){
        printf("%d",n);
        n--;
    }
}
fun(3);
```

3 2 1

Time complexity : O(n)
Space complexity : O(1)
More efficient in Space

Tail < Loop

54. Head Recursion

```
void fun(int n) {
    if (n>0) {
        fun(n-1);
        printf("%d", n);
    }
}
fun(3);
```

1 2 3

→ Head Recursion
Processing is done at returning time

* Comparing loop to Recursion

```
void fun(int n) {
    while (n>0) {
        printf("%d", n);
    }
}
fun(3);
```

1 2 3

It cannot be directly converted to loops easily

```
void fun(int n) {
    int i=1;
    while (i<=n) {
        printf("%d", i);
        i++;
    }
}
```

1 2 3

55. Tree Recursion

* Linear Recursion

```
fun(n){
  if(n>0){
    --
    -
1 -> fun(n-1);
    -
    -
  }
}
```

Function which calls itself only one time is Linear recursion

* Tree Recursion

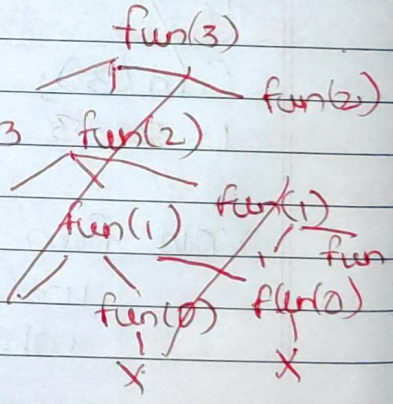
```
fun(n){
  if(n>0){
    -
    -
1 -> fun(n-1);
    -
2 -> fun(n-1);
    -
  }
}
```

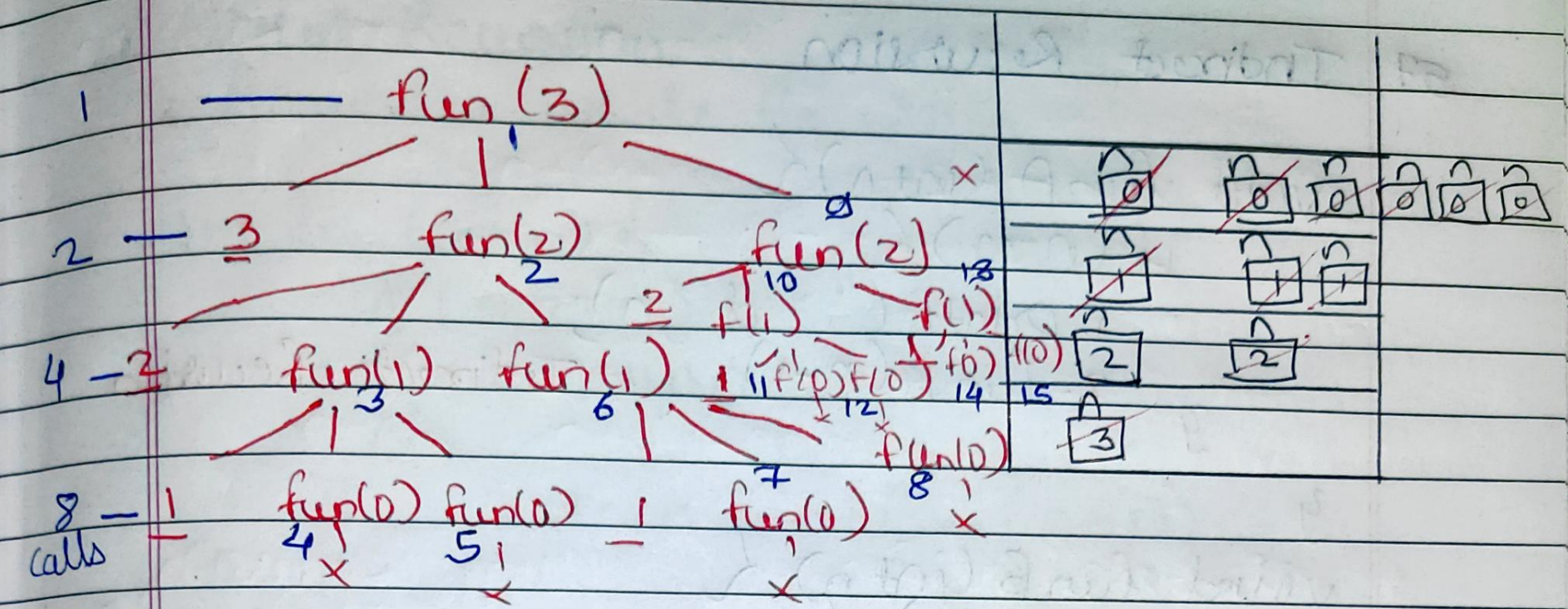
Function which calls itself more than one time is tree recursion

* Tree Recursion

```
void fun(int n){
  if(n>0){
    1. printf("%d",n);
    2. fun(n-1);
    3. fun(n-1);
  }
}
fun(3);
```

3 2 1 1 2 1 1





$1 + 2 + 4 + 8 = 15$

$2^0 + 2^1 + 2^2 + 2^3 = 2^{3+1} - 1$

$\therefore 2^0 + 2^1 + 2^2 \ldots + 2^n = \underline{2^{n+1}} - 1$

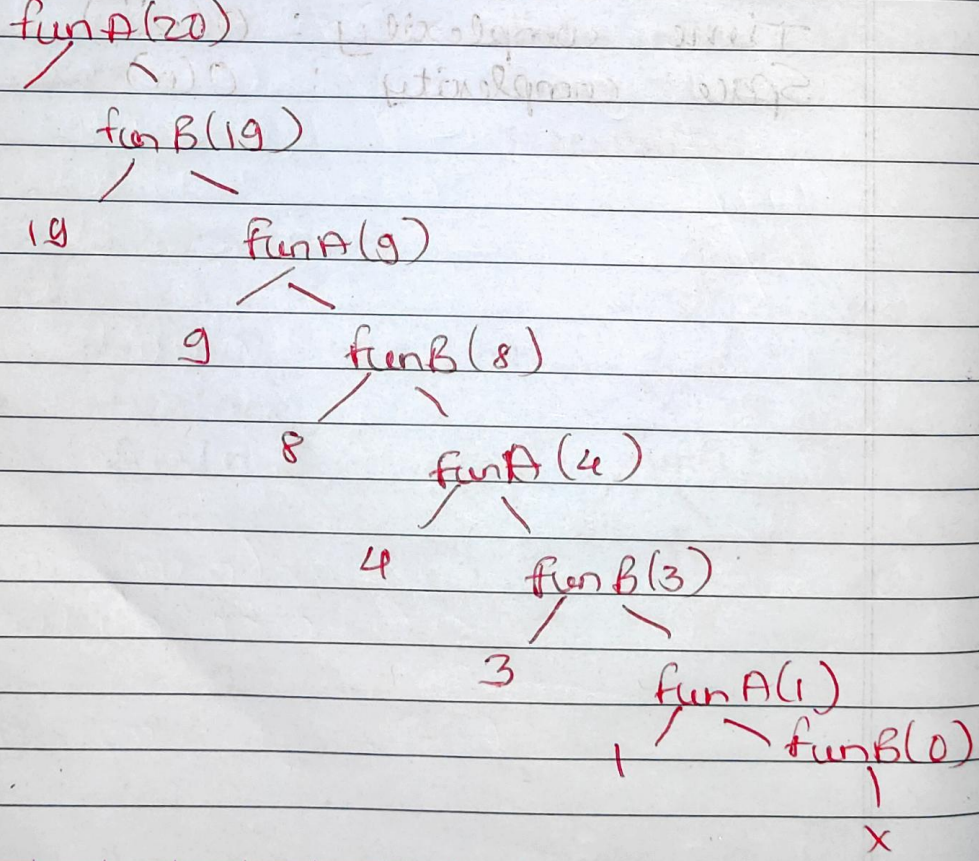
Time complexity : $O(2^n)$

Space complexity : $O(n)$

57. Indirect Recursion

```
void funA(int n){
    if(n>0){
        1. printf("%d", n);
        2. funB(n-1);   → Indirect Recursion
    }
}
void funB(int n){
    if(n>1){
        1. printf("%d", n);
        2. funA(n/2);   → Indirect Recursion
    }
}
```

20 19 9 8 4 3 1



59 Nested Recursion
Recursion inside Recursion

```
int fun (int n) {
    if (n > 100)
        return n - 10;
    else
        return fun(fun(n+1));
}
```

fun(95): 91

fun(95) = 91

↓

fun(fun(95+1)) 96 = fun(106)
fun(96)

↓

fun(fun(107)) 97 = fun(107)
fun(97)

↓

fun(fun(108)) 98 = fun(108)
fun(98)

↓

fun(fun(109)) 99 = fun(109)
fun(99)

↓

fun(fun(110)) 100 = fun(110)
fun(100)

↓

fun(fun(111)) 101 = fun(111)
fun(101)

↓

91

61. Sum of Natural Number using Recursion

$$Sum(n) = 1+2+\dots+(n-1)+n$$
$$= Sum(n-1) + n$$

$$Sum(n) = \begin{cases} 0 & n=0 \\ Sum(n-1)+n & n>0 \end{cases}$$

* Using Recursion

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

Sum(5) = 15
Sum(4) + 5 = 10 + 5 = 15
Sum(3) + 4 = 6 + 4 = 10
Sum(2) + 3 = 3 + 3 = 6
Sum(1) + 2 = 1 + 2 = 3
Sum(0) + 1 = 0 + 1

Time complexity: O(n)
Space complexity: O(n)

* Using loops

```
int sum(int n){
    int i, s=0;          // -1
    for(i=1; i<=n; i++){ // -n+1
        s=s+i;           // -n
    }
    return n;            // -1
}
```

n+3

Time complexity: O(n)

* Using Formula

```
int sum(int n){
    return n*(n+1)/2;
}
```

Time complexity: O(1)

13. Factorial using Recursion

$0! = 1$
$1! = 1$

$n! = 1 * 2 * \ldots * n$

$5! = 1 * 2 * \ldots * 5 = 120$

$fact(n) = 1 * 2 * \ldots * (n-1) * n$

$fact(n) = fact(n-1) * n$

$$fact(n) = \begin{cases} 1 & n = 0 \\ fact(n-1) * n & n > 0 \end{cases}$$

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

Time complexity: $O(n)$
Space complexity: $O(n)$

65. Power using Recursion

Exponent $(m)^n$

$2^5 = 2*2*2*\ldots*2$

$m^n = m*m*\ldots*$ n times

pow(m,n) = (m*m*m.... *n-1 times) * m

pow(m,n) = pow(m,n-1) * m

$$pow(m,n) = \begin{cases} 1 & n=0 \\ pow(m,n-1)*m & n>0 \end{cases}$$

```
* int pow(int m, int n){
    if (n == 0)
        return 1;
    return pow(m, n-1) * m;
}
```

Time complexity : O(n)

Space complexity : O(n)

6 Multiplication

```
* int pow(int m, int n){
    if(n == 0)
        return 1;
    if (n%2 == 0)
        return pow(m*m, n/2);
    else
        return m* pow(m*m, (n-1)/2);
}
```

pow(2,9) = 2^9

1

6 multiplication $2 * pow(2^3, 4) = 2*2^8 = 2^9$

1

$pow(2^4, 2) = 2^8$

1

$pow(2^8, 1) = 2^8$

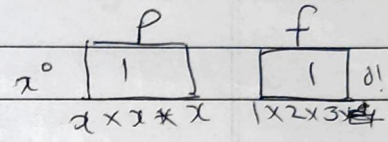
1

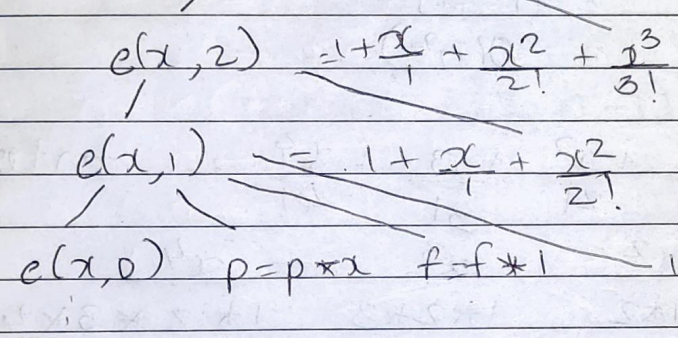
$2^8 * pow(2^{16}, 0) = 2^8 * 1 = 2^8$

67. Taylor series using Recursion e^x

$$e^x = 1 + \frac{x}{1} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} \ldots + n \text{ terms}$$

sum(n-1)+n sum(n) = 1+2+...+n
fact(n-1)*n fact(n) = 1*2*3*....*n
pow(x,n-1)*x pow(x,n) = x*x*x*...n times






```
int e (int x, int n) {
    static int p=1, f=1;
    int r;
    if (n==0)
        return(1);
    else {
        r=e(x,n-1);
        p=p*x;
        f=f*n;
        return r+p/f;
}
```

69. Taylor series using Horner's Rule

$$e^x = 1 + \frac{x}{1} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} \cdots + n \text{ terms}$$

$$= 0 + 0 \quad \frac{x*x}{1*2} \quad \frac{x*x*x}{1*2*3} \quad \frac{3}{3}$$

$$2 + 4 + 6 + \ldots$$

$$2[1 + 2 + 3 + \ldots]$$

$$2 \frac{n(n+1)}{2}$$

$$= n(n+1) = \textcircled{8}$$

$$= n^2 + n$$

$$= O(n^2)$$

* $$e^x = 1 + \frac{x}{1} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} \cdots + n \text{ terms}$$

$$= 1 + \frac{x}{1} + \frac{x^2}{1*2} + \frac{x^3}{1*2*3} + \frac{x^4}{1*2*3*4}$$

$$= 1 + \frac{x}{1}\left[1 + \frac{x}{2} + \frac{x^2}{2*3} + \frac{x^3}{2*3*4}\right]$$

$$= 1 + \frac{x}{1}\left[1 + \frac{x}{2}\left[1 + \frac{x}{3} + \frac{x^2}{3*4}\right]\right]$$ $O(n^2)$ Quadratic

$$= 1 + \frac{x}{1}\left[1 + \frac{x}{2}\left[1 + \frac{x}{3}\left[1 + \frac{x}{4}\right]\right]\right]$$ $O(n)$ Linear

```
int e(int x, int n){ // using loops
    int s=1;
    for(n>0 ; n--){
        s = 1 + x/n * s;
    }
    return s;
}
```

```
int e(int x, int n){ // using recursion
    static int s=1;
    if(n==0)
        return s;
    s = 1 + x/n * s;
    return e(x, n-1);
}
```

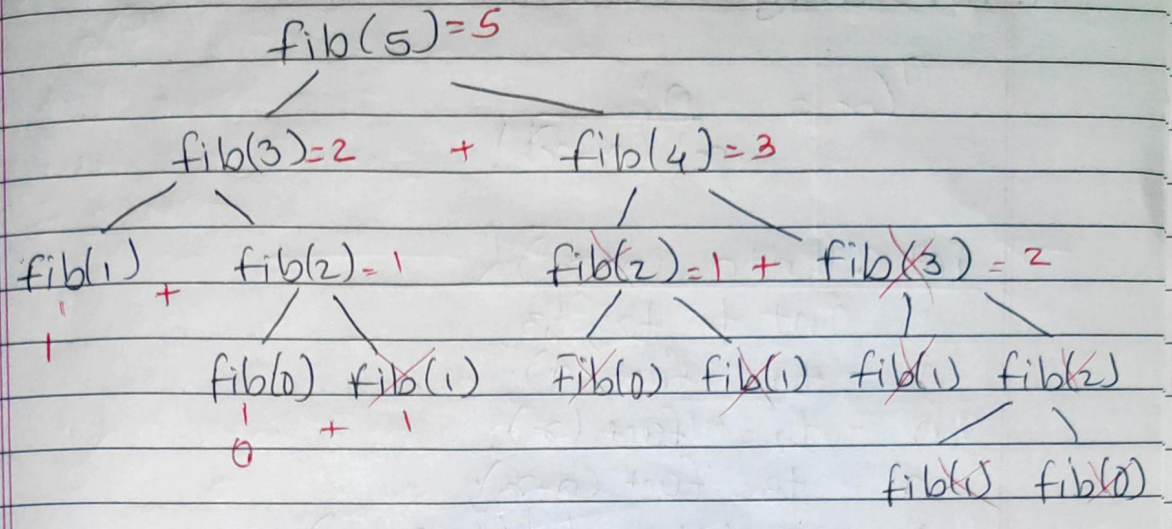
72. Fibonacci series using Recursion-Memoization

fib(n)	0	1	1	2	3	5	8	13
n	0	1	2	3	4	5	6	7

$$fib(n) = \begin{cases} 0 & n=0 \\ 1 & n=1 \\ fib(n-2) + fib(n-1) & n>1 \end{cases}$$

```
* int fib(int n){                    //Loop Iterative
    int t0 = 0, t1 = 1, s, i;          -1
    if (n<=1) return n;                -1
    for (i = 2; i<=n; i++){            -n
        s = t0 + t1;                   -n-1
        t0 = t1;                       -n-1
        t1 = s;                        -n-1
    }
    return s;                          -1
}                                      ---
                                       4n+1
Time complexity: O(n)

* int fib(int n){                    //using recursion
    if (n<=1)                          //Excessive recursion
        return n;
    return fib(n-2) + fib(n-1);
}                    2 fib(n-1)
Time complexity: O(2^n)
```



F | $-Y_0$ | $-Y_1$ | Y_1 | $-Y_2$ | $-Y_3$ | $-Y_5$ |

Memoization 0 1 2 3 4 5

Time complexity : O(n)

* int F[10];
int fib(int n){ // using memoization
if(n<=1){
F[n] = n;
return n;
}else{
if(F[n-2] == -1)
F[n-2] = fib(n-2);
if(F[n-1] == -1)
F[n-1] = fib(n-1);
return F[n-2] + F[n-1];
}
}

F[n] = F[n-2] + F[n-1];

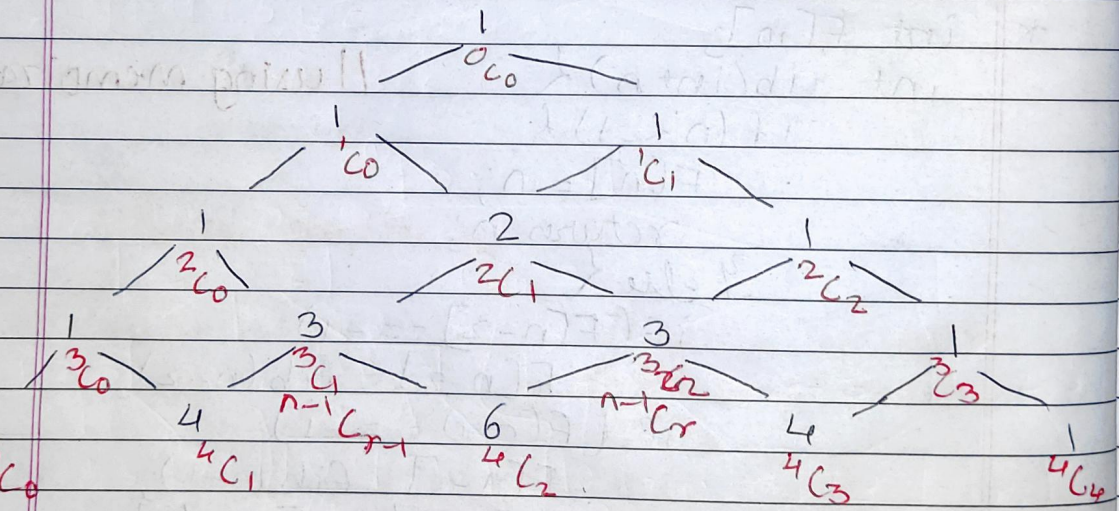
74. nCr using Recursion

$$^{n}C_{r} = \frac{n!}{r!(n-r)!}$$

```
int C(int n, int r){
    int t1, t2, t3;
    t1 = fact(n);
    t2 = fact(r);
    t3 = fact(n-r);
    return t1/(t2*t3);
}
```

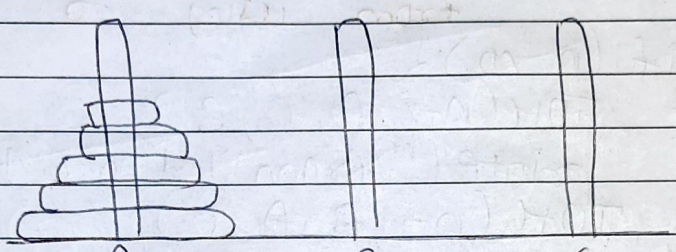
Time complexity: O(n)

* Pascal's Triangle



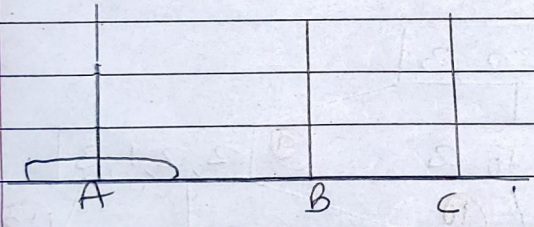
```
int c(int n, int r){          // using recursion
    if(r==0 || n==r){
        return 1;
    else
        return c(n-1, r-1) + c(n-1, r);
}
```

76. Tower of Hanoi Problem



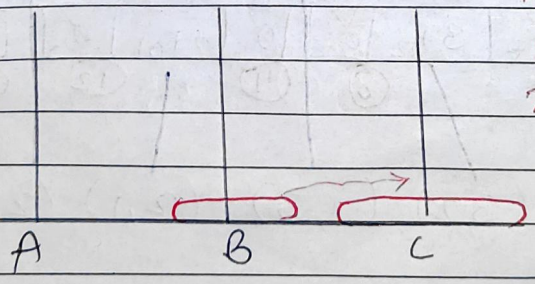
from ... using to

* 1 Disk



TOH(1, A, B, C)
move Disk from A to C using B

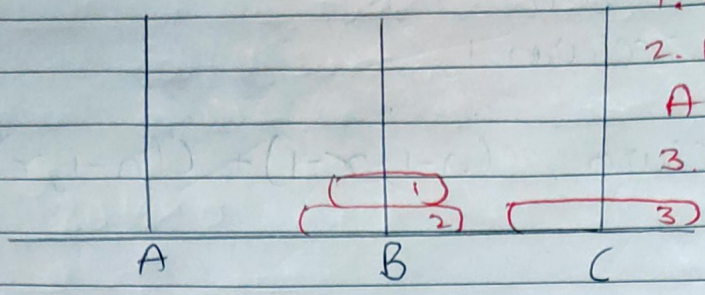
* 2 Disk // recursion



TOH(2, A, B, C)
1. TOH(1, A, C, B)
2. move Disk from A to C using B
3. TOH(1, B, A, C)

* 3 Disk

TOH(3, A, B, C)
1. TOH(2, A, C, B)
2. Move disk from A to C using B
3. TOH(2, B, A, C)



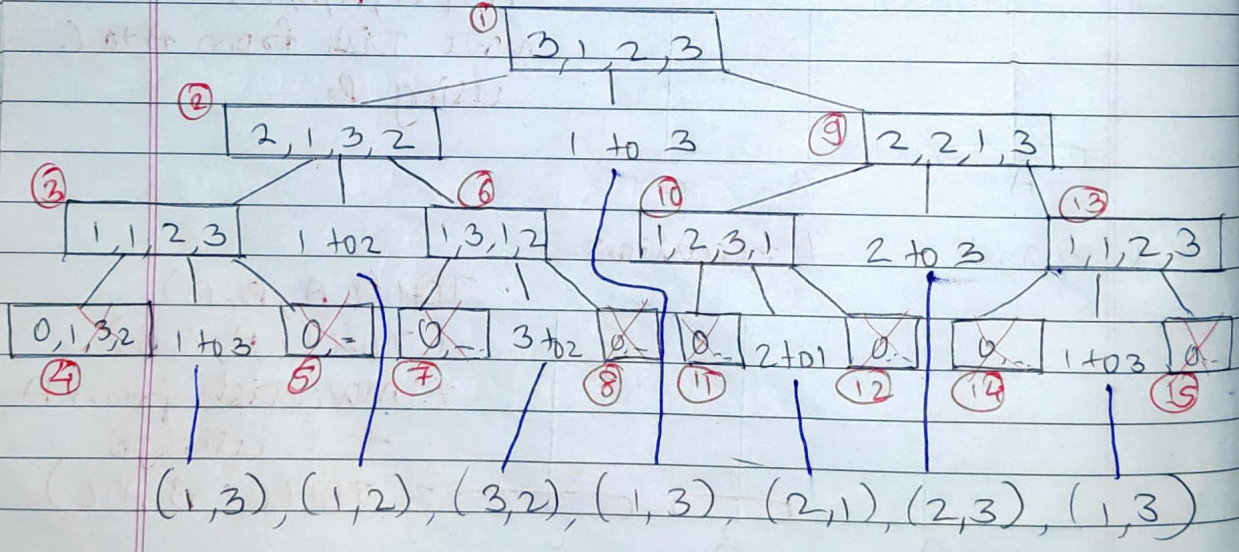

```
void TOH(int n, int A, int B, int C) {
                    from  using  to
    if (n>0) {
        TOH(n-1, A, C, B);
        printf("from %d to %d", A, C);
        TOH(n-1, B, A, C);
    }
```

$1+2+2^2+\dots+2^n = 2^{n+1}-1$

Time complexity: $O(2^n)$

calls

n=3 is TOH(3, 1, 2, 3);
n=2 is $1+2+2^2+2^3 = 2^4-1$ — A A B C



78. Quiz1

1) Question 1

```
int fun (int n) {
    static int i;
    if (n>=5) return n;
    n= n+i;
    i++;
    return f(n);
}
f(1);
```

i: 1 2 3 4

→ 7

f(1) = 7

n = n+i = 2 i++ f(2) = ?

n = 2+2 = 4 i++ f(3) = ?

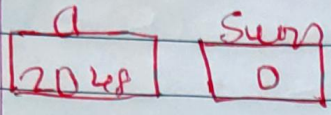
n = 4+3 = 7 i++ f(7)

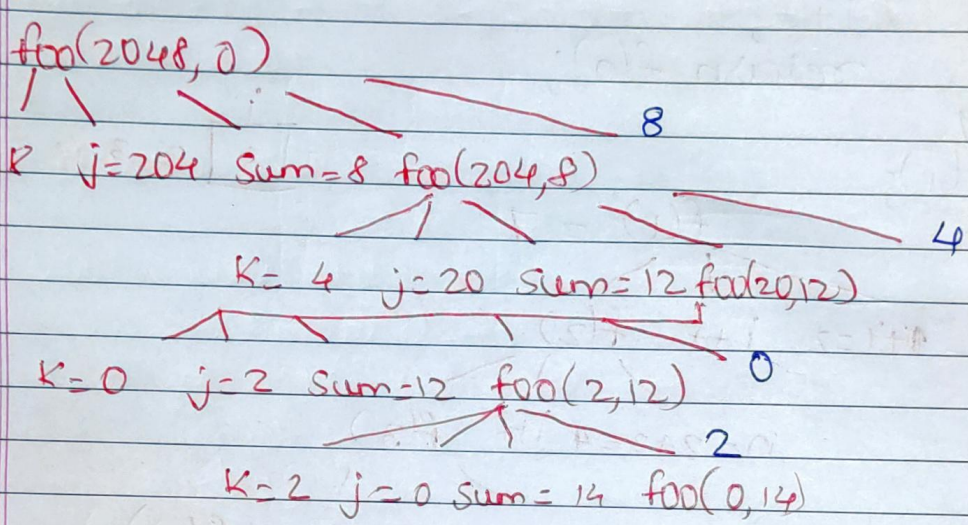
2) Question 2

```
void foo (int n, int sum) {
    int k=0, j=0;
    if (n==0) return;
    k = n % 10;
    j = n / 10;
    sum = sum + k;
    foo (j, sum);
    printf ("%d", k);
}
```

```
int main() {
    int a = 2048, sum = 0;
    foo(a, sum);
    printf("%d", sum);
}
```

→ 20480





3) Question 3

```
intf (int &x, int c) {
    c = c-1;
    if (c==0) return;
    x = x+1;
    return f(x, c) * x;
}
```

```
int main(){
    int p=5;
    printf("%d", f(p,p));
}
```

→ $g^4 = 6561$

reference → p

5	6	7	8	9

$f(x,5) = g^4$

$g^3 * g = g^4$

c=4 x=x+1 f(x,4)*x

$g^2 * g = g^3$

c=3 x=x+1 f(x,3)*x

$g * g = g^2$

c=2 x=x+1 f(x,2)*x

$1 * g = g$

c=1 x=x+1 f(x,1)*x

c=0

4) Question 4

```
int fun(int n){
    int x=1, k;
    if (n==1) return x;
    for(k=1; k<n; ++k){
        x = x + fun(k) * fun(n-k);
    }
    return x;
}
fun(5);
```

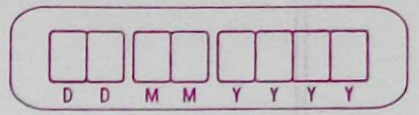
n	1	2	3	4	5
fun(n)	1	2	5	15	51

fun(2) = x + fun(1) * fun(2-1) = 1 + 1*1 = 1+1 = 2
fun(3) = x + fun(1) * fun(3-1) + fun(2) * fun(3-2)
= 1 + 1*2 + 2*1 = 1+2+2 = 5
fun(4) = 1 + f(1) * f(3) + f(2) * f(2) + f(3) * f(1)
= 1 + 1*5 + 2*2 + 5*1 = 15
fun(5) = 1 + 1*15 + 2*5 + 5*2 + 15*1
= 51

5) Question 5

```
void count(int n){
    static int d = 1;
    printf("%d", n);
    printf("%d", d);
    d++;
    if(n>1) count(n-1);
    printf("%d", d);
}
count(3)
```

→ 3 1 2 2 1 3 4 4 4



d

1 2 3 4

((3)

3 1 d++ 4

((2)

2 2 d++ 4

((1)

1 3 d++ 4