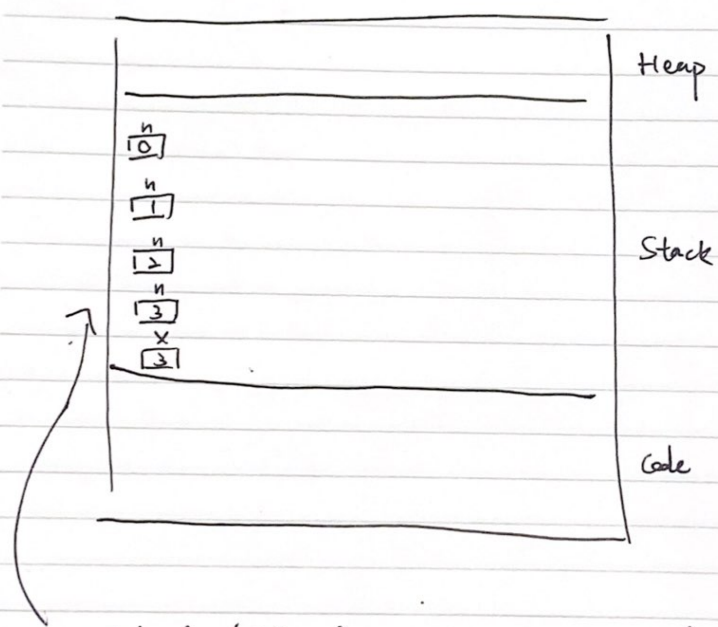
Recursion.

```
void fun(int n) {
    if (n > 0)
    {
        1. calling      (ascending)
        2. fun(n-1)
        3. returning    (descending)
    }
}
```

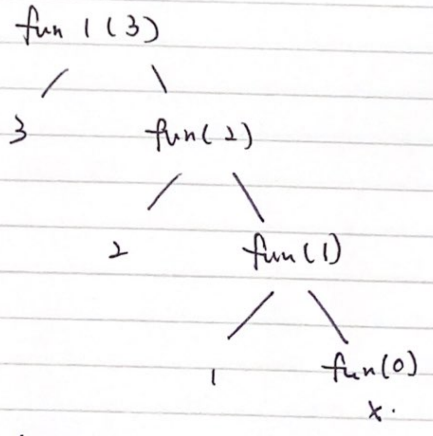
Loop only has ascending.

Recursion has ascending & descending

memory



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



o/p : 3 2 1

time complexity = O(n)

tail Recursion

```
# include <stdio.h>
  void fun (int n)
  {
       if (n>0)
       {
         printf ("%d", n);
         fun (n-1)
       }
  }

 int main ( ) {
   int x=3;
   fun (x);
   return 0;
 }
```

Static variable in Recursion

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

   main ()
   {
    int a=5;
    printf ("%d", fun(a));
   }
```

$\uparrow$ done at returning time

$$\begin{tikzcd}[row sep=small]
& \text{fun}(5) = 15 \\
\text{fun}(4) + \underline{5} = 15 \arrow[ur, "10"] & \\
\text{fun}(3) + \underline{4} \arrow[u, "6"] & \\
\underline{3} \arrow[u] & \\
\vdots \arrow[u] & \\
\text{fun}(0) + \underline{1} \arrow[u] &
\end{tikzcd}$$

local variable

```
int fun(int n)
{
 static int x=0;
   if(n>0)
   {
    x++;
    return fun(n-1)+x;
   }
   return (0);
 }.

 main(){
  int a=5;
  print ..
 }
```

created at code section

← will not have multiple copy like n.
only 1 copy.
every call uses same copy.

x
$\boxed{\cancel{0}}$ ~~1~~ ~~2~~ ~~3~~ ~~4~~ 5

```
            fun(5) = 25
           /
        fun(4) + 5
       /
     fun(3) + 5
    /
   fun(2) + 5
  /
 fun(1) + 5
 /
fun(0) + 5
/
0
```

f.

```
int x=0
int fun(int n)
{
 .
 :
```

global variable behaves same as s.v.

single one copy