Computer Programming Lab # 04

Recursion, Searching & Sorting Algorithms and Vectors

Recursion:

A subprogram is recursive when it contains a call to itself. Generally, recursive solutions are simpler than (or as simple as) iterative solutions. There are some problems in which one solution is much simpler than the other.

Examples:

Factorial

```
int factorial(int n)
{                                   // iterative solution
int f = 1;
int i= 0;
while(i < n)
{
i= i+ 1;
f = f*i;
}
return f;
}
```

Definition of factorial: $! = \cdot -1\cdot -2\cdots 2\cdot 1$

Recursive Definition:

$$n! = \begin{cases} n \cdot (n-1)!, & n > 0 \\ 1, & n = 0 \end{cases}$$

```
int factorial(int n)
{                              //recursive solution
if(n == 0) return 1;
else return n*factorial(n -1);
}
 void Array_Filling(int a[],int size)
{
if(size==0) {
a[size]=size;
return;
}
else {
Array_Filling(a,size-1);
a [size]=size; }
}


int sum_fun(int a[ ],int size)
{
if(size==0) {
return a[size];
}
else {
return a[size]+sum_fun(a,size-1); }
}
```

Recursive design

In the design of a recursive program, we usually follow a sequence of steps:

1. Identify the basic cases (those in which the subprogram can solve the problem directly without recurring to recursive calls) and determine how they are solved.

   For example, in the case of factorial, the only basic case used in the function is n=0. Similarly, we could have considered a more general basic case (e.g., $n \leq 1$). In both cases, the function should return 1.

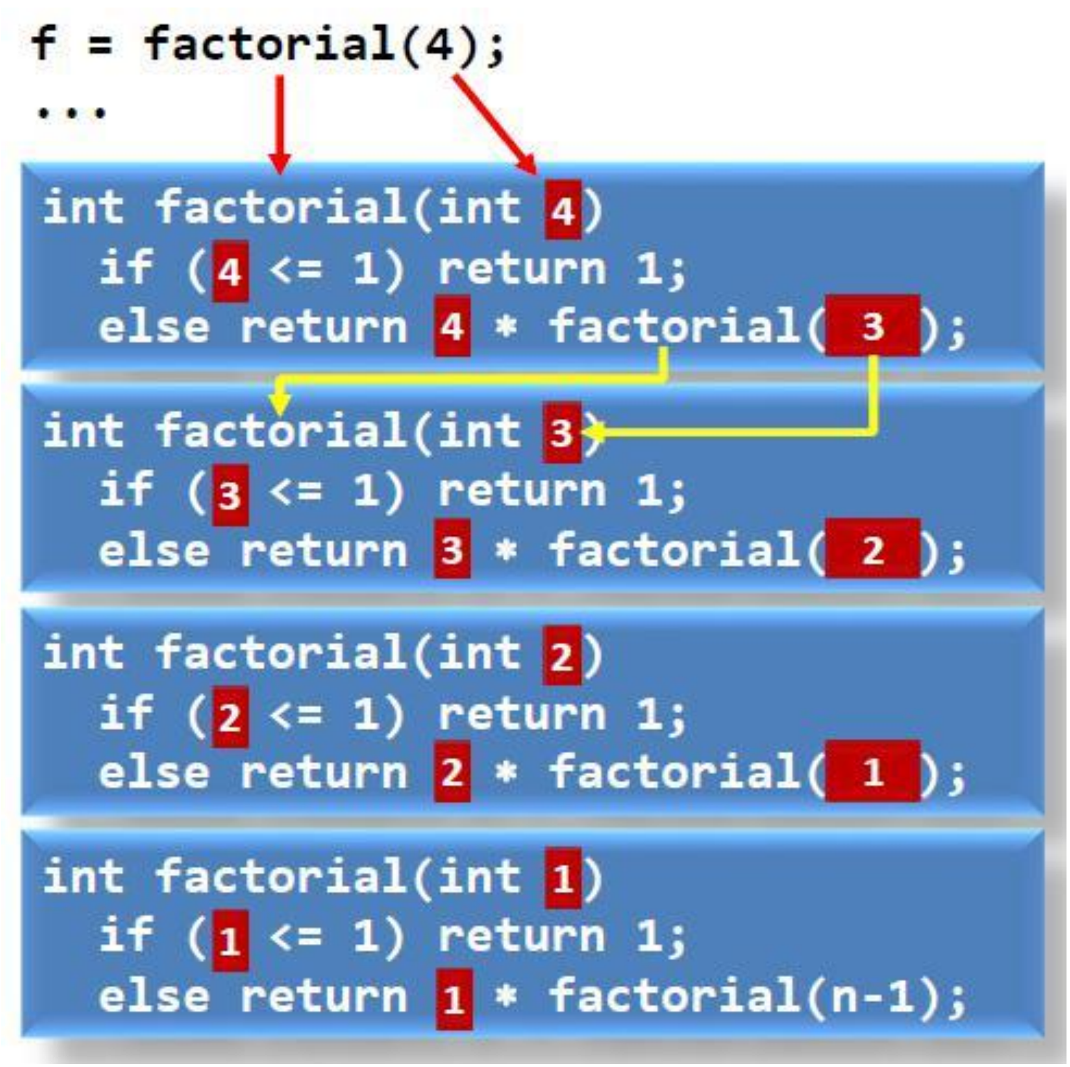
2. Determine how to resolve the non-basic cases in terms of the basic cases, which we assume we can already solve.

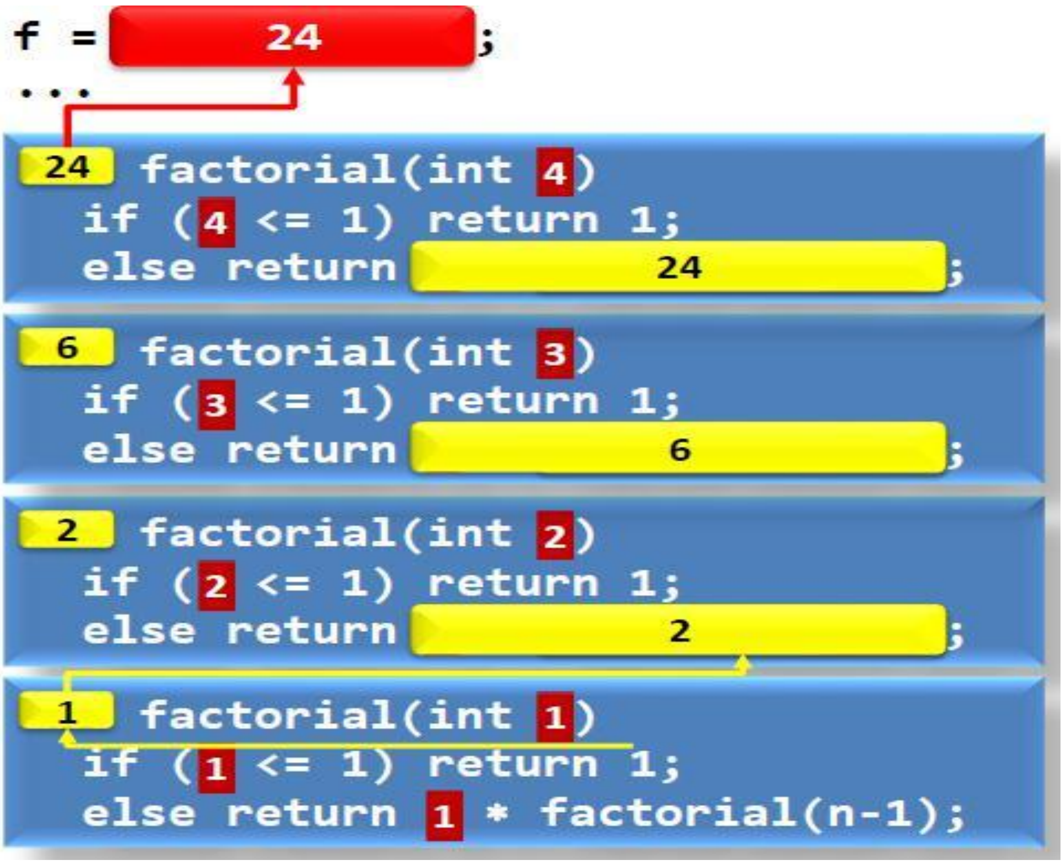
   In the case of a factorial, we know that the factorial of a number n greater than zero is n*factorial (n-1).

3. Make sure that the parameters of the call move closer to the basic cases at each recursive call. This should guarantee a finite sequence of recursive calls that always terminates.

   In the case of a factorial, n-1 is closer to 0 than n. Therefore, we can guarantee that this function terminates.

Recursion: behind the scenes





Example:

Design a procedure that, given a number *n*, writes its binary representation.

Void base2 (int n)
{

- Basic case (n=1) ->write "1"
- General case (n>1) -> write n/2 and then write n%2

```
void base2(int n) {
if(n == 1) cout<< n;
else{
base2 (n/2);
cout<< n%2;
}
}
```

The procedure always terminates since n/2 is closer to 1 than n. Note that n/2 is never 0 when n > 1. Therefore, the case n = 1 will always be found at the end of the sequence call.