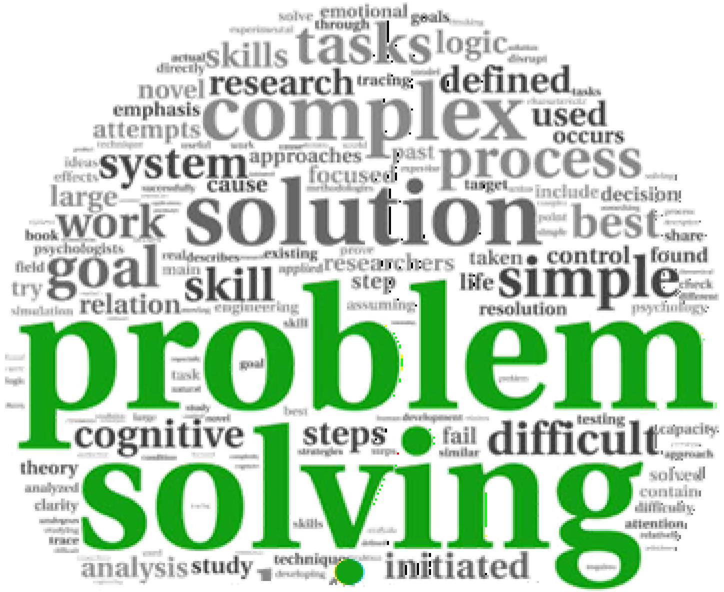


problem solving using computers

CSE 1051

20.1 RECURSION EXAMPLES

Objectives:

To learn and understand the following concepts:

- ✓ To design a recursive algorithm
- ✓ To solve problems using recursion
- ✓ To understand the relationship and difference between recursion and iteration

Session outcome:

At the end of session one will be able to :

- Understand recursion
- Write simple programs using recursive functions

Steps to Design a Recursive Algorithm

- Base case:
  - It prevents the recursive algorithm from running forever.

- Recursive steps:
  - Identify the base case for the algorithm.
  - Call the same function recursively with the parameter having slightly modified value during each call.
  - This makes the algorithm move towards the base case and finally stop the recursion.

Factorial - Recursive implementation

At each step, with time moving left to right:

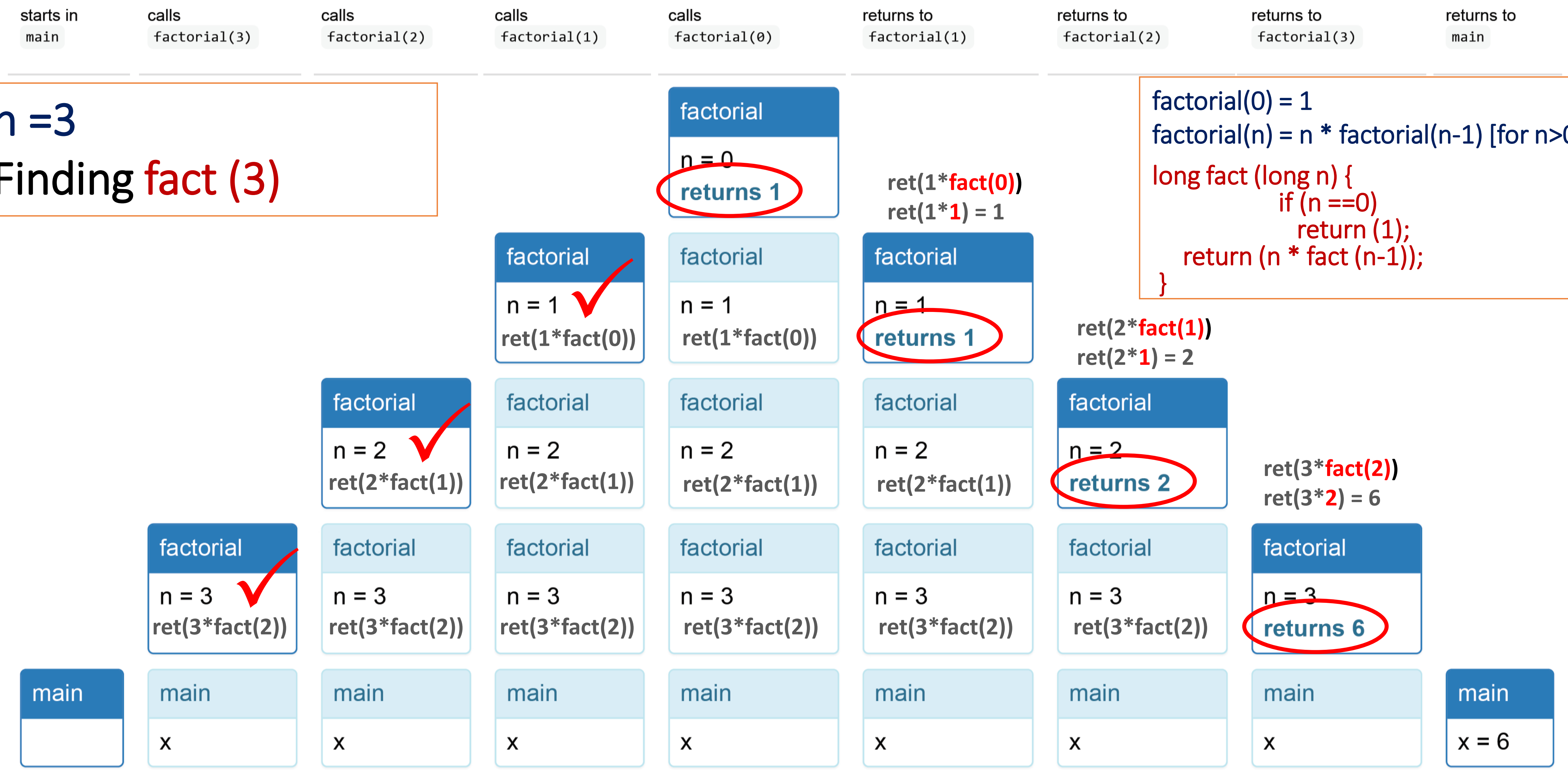
n =3

Finding fact (3)

factorial(0) = 1

factorial(n) = n * factorial(n-1) [for n>0]

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



Fibonacci Numbers: Recursion

```
fib(0) = 0
fib(1) = 1
fib(n) = fib(n-1) + fib(n-2) [for n>1]

So fib(4)
  = fib(3) + fib(2)
  = (fib(2) + fib(1)) + (fib(1) + fib(0))
  = ((fib(1) + fib(0)) + 1) + (1 + 0)
  = ( 1 + 0 ) + 1) + (1 + 0)
  = 3
```

Fibonacci Numbers: Recursion

Fibonacci series is 0,1, 1, 2, 3, 5, 8 …

```
int fib(int n)
{
  if (n <= 1)
      return n;
          else
      return (fib(n-1) + fib(n-2));
 }
```

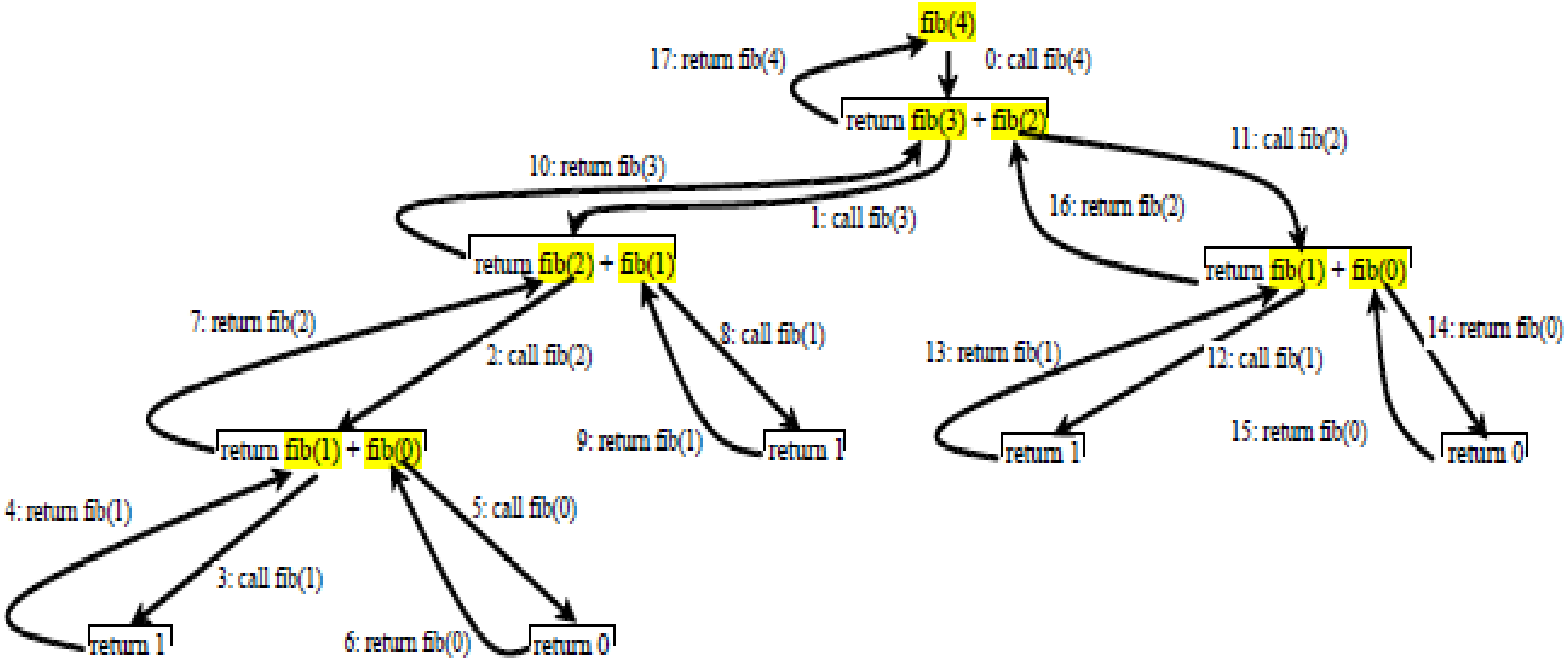
$$\mathrm{fib}(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ \mathrm{fib}(n-1) + \mathrm{fib}(n-2) & \text{if } n >= 2 \end{cases}$$

First, the terms are numbered from 0 onwards like this:

n =	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	…
x_n =	0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	…

Output:
n = 4
fib = 3

Recursive Calls initiated by Fib(4)



Fibonacci Series using Recursion

```
int fibo(int);

int main(void){
  int n,i, a[20], fibo;
  printf("enter any num to n\n“);
  scanf(“%d”, n);
  printf(“Fibonacci series “);
  for (i=1; i<=n; i++)
  {
    fibo = fib(i);
    printf(“%d     ”, fibo);
  }
  return 0;
}
```

```
int fib(int n)
{
 if (n <= 1)
  return n;
 else
  return (fib(n-1) + fib(n-2));
 }
```

Static Variable

The value of static variable persists until the end of the program.

Static variables can be declared as

```
static int x;
```

A static variable can be either an internal or external type depending on the place of declaration.

```
void fnStat( );
int main() {
int  i;
for( i= 1; i<=3; i++)
fnStat( );
 return 0;
}
```

```
void fnStat( ){
static int x = 0;
x = x + 1;
printf("x=%d", x);
}
```

Output:

x = 1

x = 2

x = 3

Reversing a Number

```
#include <stdio.h>
int rev(int);

int main() {
  int  num;
  printf("enter number)";
  scanf("%d",num);
  printf("%d", rev(num));
  return 0;
}
```

```
int  rev(int num){
    static int n = 0;
    if(num > 0)
      n = (n* 10) + (num%10) ;
    else
      return n;
    return  rev(num/10);
}
```

Output:
num = 234
rev = 432

GCD: Recursion

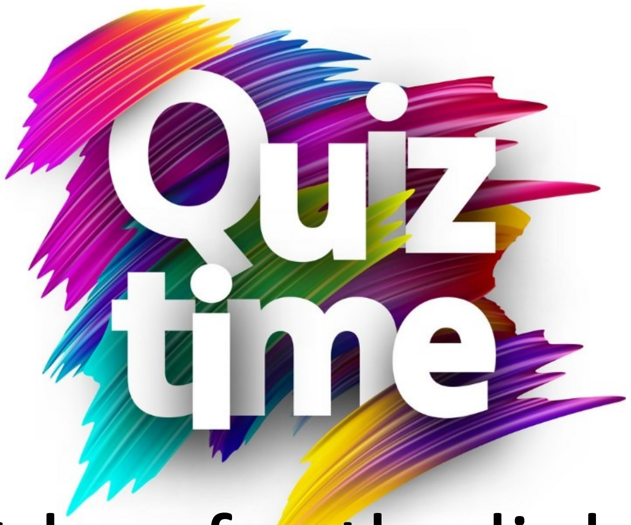
$$\gcd(x,y) = \begin{cases} x & \text{if } y = 0 \\ \gcd(y, \text{remainder}(x,y)) & \text{if } x \geq y \text{ and } y > 0 \end{cases}$$

```
int gcd(int x, int y)
{
 if (x == 0)
          return (y);
   if (y==0)
          return (x);
   return gcd(y, x % y);
}
```

Output:
x= 24 , y = 9
gcd = 3

gcd(24,9) ← *control in gcd( ) on call*

gcd(9,24%9)   gcd(9, 6)
gcd(6,9%6)   gcd(6, 3)
gcd(3,6%3)   gcd(3, 0)
return values   return 3
return 3
return 3
return 3



Go to posts/chat box for the link to the question

submit your solution in next 2 minutes

The session will resume in 3 minutes

Finding product of two numbers

```
#include <stdio.h>
int product(int, int);
int main()
{
    int a, b, result;
    printf("Enter two numbers to find their product: ");
    scanf("%d%d", &a, &b);
    result = product(a, b);
    printf("%d * %d = %d\n", a, b, result);
    return 0;
}
```

Output:
Enter two numbers to find their product: 10 20
10*20=200

```
int product(int a, int b)
{
    if (a < b)
    {
        return product(b, a);
    }
    else if (b != 0)
    {
        return (a + product(a, b - 1));
    }
    else
    {
        return 0;
    }
}
```

Division of two numbers

```
#include <stdio.h>
int divide(int a, int b);

int main()
{
   int a,b;

   printf("Enter two numbers for
division");
   scanf("%d%d", &a,&b);
   printf("%d/%d=%d",a,b, divide(a,b));
return 0;
}
```

```
int divide(int a, int b)
{
    if(a - b <= 0)
    {
        return 1;
    }
   else
   {
     return divide(a - b, b) + 1;
   }
}
```

Output:
Enter two numbers for division: 20 10
20/10=2

Recursion - Should I or Shouldn't I?

- Pros
  - Recursion is a natural fit for recursive problems.
- Cons
  - Recursive programs typically use a large amount of computer memory and the greater the recursion, the more memory used.
  - Recursive programs can be confusing to develop and extremely complicated to debug.

Recursion *vs* Iteration

RECURSION	ITERATION
Uses more storage space requirement	Less storage space requirement
Overhead during runtime	Less Overhead during runtime
Runs slower	Runs faster
A better choice, a more elegant solution for recursive problems	Less elegant solution for recursive problems

Recursion – Do's

- You must include a termination condition or Base Condition in recursive function; Otherwise your recursive function will run "forever" or infinite.
- Each successive call to the recursive function must be nearer to the base condition.

Summary

- Definition
- Recursive functions
- Problem Solving Using Recursion