Problema 1. Escribir una función que calcule el factorial de un entero positivo. Por ejemplo, 4!=24=1*2*3*4 y 0!=1

Solución 1: `x! = 1*2*...*x  (0!=1)`

```
static public int factorial(int x)
{
   int producto=1, i=1;
   while( i <= x )
   {
        producto = producto * i;
        i = i + 1;
   }
   return producto;
}
```

Solución 2: `x! = x * (x-1)!  (0!=1)`

```
static public int factorial(int x)
{
   if( x == 0 )
      return 1;
   else
      return x * factorial(x-1);
}
```

Notas

- más simple
- 4 líneas
- sin iteración (while)
- sólo uso de parámetro
- sin variables adicionales

Método (Función) recursivo

- se invoca a sí mismo
  `ej: factorial(x-1)`
- debe tener una salida no recursiva (caso base)
  `ej: if( x == 0 ) return 1;`
- las llamadas a sí mismo deben acercarse (converger) al caso base (deben disminuir el tamaño del problema)
  `ej: factorial(x-1)`

Operación

```
f(4)=4*f(3)→
f(3)=3*f(2)→
f(2)=2*f(1)→
f(1)=1*f(0)→
f(0)=1→
f(1)=1*1=1→
f(2)=2*1=2→
f(3)=3*2=6→
f(4)=4*6=24
```

Problema. Función que cuente dígitos de un entero positivo. Ej: dígitos(245) entrega 3, dígitos(4) entrega 1.

solución iterativa

```
static public int digitos(int x)
{
  int n = 1;
  while( x >= 10 )
  {
   n = n + 1;
   x = x/10;//elimina último dígito
  }
  return n;
}
```

x	245	24	2
n	1	2	3

solución recursiva:

- 1 si x<10
- 1 + digitos(x/10) si x >= 10

```
static public int digitos(int x)
{
  if( x < 10 )
    return 1;
  else
    return 1 + digitos(x/10);
}
```

digitos(245)=1+digitos(24)
digitos(24)=1+digitos(2)
digitos(2)=1
digitos(24)=1+1=2
digitos(245)=1+2=3

Problema. Método que reciba un entero y lo escriba al revés
Ejemplo: **invertir(345);** escribe 543.

```
static public void invertir(int x){...}
```

iterativa

```
while( x>=10 ){
  U.print(x%10);
  x = x/10;
}
U.print(x);
```

recursiva

```
if( x>=10 ){
  U.print(x%10);
  invertir(x/10);
}else
  U.print(x);
```

recursiva abreviada

```
U.print(x%10);    //escribir último dígito
if( x >= 10 )     //si tiene más dígitos
   invertir(x/10);//  invertirlos
```

Problema . Función que calcule el máximo común divisor
Ejemplos: mcd(18,24)=6, mcd(9,4)=1

Solución 1: "fuerza bruta"

```
static public int mcd(int x,int y){
  int max=1, i=2;
  while( i <= Math.min(x,y) ){
    if( x%i==0 && y%i==0 ) max=i;
    i = i + 1;
  }
  return max;
}
```

Nota.
mcd(18,24) realiza 17 iteraciones.
mcd(9,4) realiza 3 iteraciones.
mcd(x,y) realiza Math.min(x,y)-1 iteraciones.

Solución 2. fuerza bruta mejorada

```
static public int mcd(int x,int y)
{
  int i=Math.min(x,y);
  while( i > 1 ){
    if( x%i==0 && y%i==0 ) return i;
    i = i - 1;
  }
  return 1;
}
```

Nota.
mcd(18,24) realiza 13 iteraciones
mcd(9,4) realiza 3 iteraciones
mcd (x,y) realiza máximo Math.min(x,y)-1 iteraciones

Solución 3. Algoritmo de Euclídes

```
static public int mcd(int x,int y)
{
  while( x != y )
    if( x > y )
      x = x - y;
    else
      y = y - x;
  return x;
}
```

Nota.
mcd(18,24) realiza 3 iteraciones: (18,24),(18,6),(12,6)
mcd(9,4) realiza 5 iteraciones: (9,4),(5,4),(1,4),(1,3),(1,2)
mcd(x,y) realiza máximo Math.min(x,y)+1 iteraciones

Solución 4. Algoritmo de Euclídes recursivo

```
static public int mcd(int x,int y){
  if( x == y ) return x;
  if( x > y )
    return mcd(y,x-y);
  else
    return mcd(x,y-x);
}
```

Nota.
`mcd(18,24) mcd(18,6) mcd(12,6) mcd(6,6) 6`
3 llamadas recursivas

`mcd(9,4) mcd(5,4) mcd(1,4) mcd(1,3)`
`mcd(1,2) mcd(1,1) 1`
5 llamadas recursivas

Solución 5. Euclídes optimizado (menos invocaciones)

```
static public int mcd(int x,int y)
{
  int max=Math.max(x,y), min=Math.min(x,y);
  if( min==0 ) return max;
  return  mcd(min, max % min);
}
```

`mcd(18,24)→mcd(18,6)→mcd(6,0)→6`
2 llamadas recursivas

`mcd(9,4)→mcd(4,1)→mcd(1,0)→1`
2 llamadas recursivas

Problema.
//calcular x^y (y:entero≥0) **recursivamente** sin usar Math.pow(x,y)
//Ejs**:** potencia(2.0,3)=8.0 potencia(2.0,0)=1.0

```
static public double potencia(double x,int y){
…
}
//calcular potencias enteras de 2
static public void main(String[]x)throws IOException{
…
}
n?3
2^3=8.0
n?-3
2^-3=0.125
…
n?0 (fin de datos)
```

```
static public double potencia(double x,int y){
  if(y==0)
    return 1.0;
  else
    return x * potencia(x,y-1);
}
static public void main(String[]x)throws IOException{
  int n=U.readInt("n?");
  while(n!=0)
  {
    if(n>0)
      U.println("2^"+n+"="+potencia(2,n));
    else
      U.println("2^"+n+"="+1/potencia(2,-n));
    n=U.readInt("n?");
  }
}
```

Solución 2

```
static public double potencia(double x,int y){
  if(y==0) return 1.0;
  return x * potencia(x,y-1);
}
static public void main(String[]x)throws IOException{
  while(true){
    int n=U.readInt("n?");
    if(n==0) break;
    U.print("2^"+n+"=");
    double p=potencia(2,Math.abs(n));
    if(n>0)
      U.println(p);
    else
      U.println(1/p);
  }
}
```

Solución 3

$x^y = x\ x^{y-1}$ si y es impar
$x^y = x^{y/2}\ x^{y/2}$ si y es par

```
static public double potencia(double x,int y)
{
  if(y==0)
    return 1.0;
  else if(y%2==1) //impar?
    return x * potencia(x,y-1);
  else{
    double aux=potencia(x,y/2);
    return aux*aux;
  }
}
```

Solución 4

```
static public double potencia(double x,int y)
{
  if(y==0) return 1.0;

  double aux=potencia(x,y/2);

  if(y%2==0)
    return aux * aux;
  else
    return x * aux * aux;
}
```

Nota. Realiza $\log_2 y$ -1 llamadas recursivas (y no y-1)
Ej: f(x,17)-->f(x,8)-->f(x,4)-->f(x,2)-->f(x,1)-->f(x,0)
5 llamadas (y no 16)

Problemas propuestos

- int permutaciones(int x,int y) //x!/(x-y)!
- int combinaciones(int x,int y) //x!/(y!(x-y)!)
- int fibonacci(int i)//entrega i-ésimo nº de Fibonacci
  nros: 0, 1, 1, 2, 3, 5, 8, 13, 21, …
- void invertir(int x)//invertir(123) escribe 321
  Un método void no es una función, es decir,
  no entrega resultado y por lo tanto no termina
  con la instrucción return expresión
- int inverso(int x)//inverso(123) entrega 321