Estructuras de Datos y Algoritmos

CURSO 2009 – PRÁCTICO 2

SOLUCIÓN

1-

```
Iguales: LNat*LNat->boolean
```

- `Iguales([],[])=True`
- `Iguales(x,[])=False`
- `Iguales([],y)=False`
- `Iguales(x.Xs,y.Ys)=`

```
            si x==y -> Iguales(Xs,Ys)
            sino    -> False
```

2-

```
IsIn: Nat*LNat->Boolean
```

- `IsIn(x,[])=False`
- `IsIn(x,y.Ys)=`

```
      si x==y -> True
      sino    -> IsIn(x,Ys)
```

3-

```
Count: Nat*LNat->Nat
```

- `Count(x,[])=0`
- `Count(x,y.Ys)=`

```
      si x==y -> 1 + Count(x,Ys)
      sino    -> Count(x,Ys)
```

4-

```
Remove: Nat*LNat->LNat
```

- `Remove(x,[])=[]`
- `Remove(x,y.Ys)=`

```
-x==y -> Remove(x,Ys)
-x!=y -> y.Remove(x,Ys)=Cons(y,Remove(x,Ys))
```

5-

```
InsOrd: Nat*LNat->LNat
```

- `InsOrd(x,[])=[x]=x.[]=[].x`
- `InsOrd(x,y.Ys)=`

```
      si x<=y -> x.y.Ys
      sino    -> y.InsOrd(x,Ys)
```

6-

```
Ordenar: LNat->LNat
```

- `Ordenar([])=[]`
- `Ordenar(x.[])=[x]`
- `Ordenar(x.y.Xs)=`

```
            si x<y -> Merge(x.y,Ordenar(Ys))
            sino   -> Merge(y.x,Ordenar(Ys))
```

Otra solución:

```
Ordenar: LNat->LNat
```

- `Ordenar([])=[]`
- `Ordenar(y.Ys)= InsOrd(Ordenar(Ys),y)`

7-

```
Merge--Mezcla dos listas ordenadas y devuelve una lista ordenada
Merge: LNat*LNat->LNat
```

- `Merge([],[])=[]`
- `Merge(x.Xs,[])= x.Xs`
- `Merge([],y.Ys)= y.Ys`
- `Merge(x.Xs,y.Ys)=`

```
            si x<y -> x.Merge(Xs,y.Ys)
            sino   -> y.Merge(x.Xs,Ys)
```

8-

```
Reverse: LNat->LNat
```

- `Reverse([])=[]`
- `Reverse(x)=x`
- `Reverse(x.Xs)=Reverse(Xs).x`

9-

```
Max: LNat->Nat
//Pre: la lista no es vacia
```

- `Max(x.[])=x`
- `Max(x.y.Ys)=`

```
      si x>Max(y.Ys) -> Max(x.Ys)
      sino           -> Max(y.Ys)
```

10-

El cálculo del máximo común divisor de dos números se puede realizar de forma recursiva mediante el algoritmo de Euclides.
La implementación recursiva se basa en la siguiente propiedad:

```
mcd(a,b)= mcd(b, a % b), con a >= b, b!=0.
```

Por ejemplo: mcd(96, 36)= mcd(36, 24)= mcd(24, 12)= mcd(12, 0).
Cuando b se hace cero, no es necesario continuar, ya que mcd(a, 0)= a.

11-

```
Fib: Nat ->Nat
```

- Fib(0)= 1
- Fib(1)= 1
- Fib(n)= Fib (n-1) + Fib(n-2)

```
// funcion recursiva para fibonacci
int fibonacci(int n) {
      switch (n) {
            case 0: return 1;
            case 1: return 1;
            default: return (fibonacci(n-1) + fibonacci(n-2));
      }
}
```

12-

```
Fact: Nat ->Nat
```

- Fact(0)= 1
- Fact(n)= n * Fact(n-1)

```
// funcion recursiva para factorial
int factorial(int n) {
      if (n == 0)
            return 1;
      else
            return n*factorial(n-1);
}
```

13-

```
Potencia: Nat*Nat->Nat
// base*exponente->resultado
```

- Potencia(b,0)= 1
- Potencia(b,e)= b * Potencia (b,e-1)

```
// funcion recursiva para para calcular la potencia
int potencia (int base, int exponente) {
      if (exponente == 0){
            return 1;
      }
      else{
            return (base * potencia(base, exponente-1));
      }
}
```

14-

```
Suma: LNat->Nat
```

- Suma([])= 0
- Suma(x.Xs)= x + Suma(Xs)