Algoritmos y Estructuras de Datos 2

Práctica 6: Divide & Conquer

1er cuatrimestre 2022

Índice

1. Ejercicio 1: más a la izquierda

MasALaIzquierda(**in** A: arreglo(nat)) → **out** res: bool

Pre ≡ {tam(A) = 2^k para algún k: nat}

```
1: n ← tam(A)
2: m ← n div 2
3: izq ← SubArreglo(A, 1, m)                          ▷ O(1)
4: der ← SubArreglo(A, m + 1, n)                      ▷ O(1)
5: sumaIzq ← Sumar(izq)                               ▷ O(n/2)
6: sumaDer ← Sumar(der)                               ▷ O(n/2)
7: if sumaIzq > sumaDer then
8:     if n = 2 then
9:         res ← true
10:    else
11:        res ← MasALaIzquierda(izq) ∧ MasALaIzquierda(der)
12:    end if
13: else
14:    res ← false
15: end if
```

Complejidad:
$T(n) = 2T(n/2) + n$
Sea $a = 2$, $b = 2$, $f(n) = n$
$f(n) \in \Theta(n^{log_b(a)}) = \Theta(n^{log_2(2)}) = \Theta(n)$
$\Rightarrow T(n) = \Theta(n^{log_b(a)} log(n)) = \Theta(nlog(n))$

2. Ejercicio 2

Pendiente.

3. Ejercicio 3: potencia

Potencia(**in** a: nat, **in** b: nat) → **out** res: nat

```
1: if b = 0 then
2:     res ← 1
3: else if b mod 2 = 0 then
4:     c ← Potencia(a, b / 2)
5:     res ← c * c
6: else
7:     c ← Potencia(a, (b - 1) / 2)
8:     res ← c * c * a
9: end if
```

Complejidad:
$T(b) = T(b/2) + cte$
Sea $a = 1$, $c = 2$, $f(b) = cte \in O(1)$
$f(b) \in \Theta(n^{log_c(a)}) = \Theta(n^{log_2(1)}) = \Theta(n^0) = \Theta(1)$
$\Rightarrow T(b) = \Theta(n^{log_c(a)} log(b)) = \Theta(log(b))$

4. Ejercicio 4

Pendiente.

5. Ejercicio 5: suma de potencias

SumaDePotencias(**in** A: arreglo(arreglo(nat)), **in** n: nat) → **out** res: arreglo(arreglo(nat))

Pre ≡ {n = 2^k para algún k: nat ≥ 1}

```
1: if n = 1 then
2:     res ← A
3: else
4:     B ← SumaDePotencias(A, n / 2)
5:     res ← Potencia(A, n / 2) * B + B
6: end if
```

Complejidad:
$T(n) = T(n/2) + cte$
Sea $a = 1$, $b = 2$, $f(n) = cte \in O(1)$
$f(n) = \Theta(n^{log_b(a)}) = \Theta(n^{log_2(1)}) = \Theta(n^0) = \Theta(1)$
$\Rightarrow T(n) = \Theta(n^{log_b(a)} log(n)) = \Theta(log(n))$
Asumimos que la función Potencia = O(1).

6. Ejercicio 6

ab es tupla ⟨ izq: ab, der: ab ⟩

mdr es tupla ⟨ maxDistancia: nat, altura: nat ⟩

MaxDistancia(**in** nodo: ab) → **out** res: nat

```
1: res ← MaxDistanciaAux(nodo).maxDistancia
```

Complejidad:

MaxDistanciaAux(**in** nodo: ab) → **out** res: mdr

```
1: if nodo = nil then
2:     res ← ⟨ maxDistancia: 0, altura: 0 ⟩
3: else
4:     resIzq ← MaxDistanciaAux(nodo.izq)
5:     resDer ← MaxDistanciaAux(nodo.der)
6:     res ← ⟨
7:       maxDistancia: max(resIzq.maxDistancia, resDer.maxDistancia, resIzq.altura + resDer.altura + 1),
8:       altura: max(resIzq.altura, resDer.altura) + 1
9:     ⟩
10: end if
```

Complejidad:

Revisar porque este algoritmo cuenta la cantidad de nodos en vez de ejes.

7. Ejercicio 7

Pendiente.

8. Ejercicio 8

Pendiente.

9. Ejercicio 9

Pendiente.

10. Ejercicio 10

Pendiente.

11. Ejercicio 11

Pendiente.

12. Ejercicio 12

Pendiente.