Algoritmos y Estructuras de Datos 2

Práctica 6: Divide & Conquer

1er cuatrimestre 2022

Índice

1. Ejercicio 1: más a la izquierda **2**

2. Ejercicio 2: índice espejo **2**

3. Ejercicio 3: potencia **3**

4. Ejercicio 4 **3**

5. Ejercicio 5: suma de potencias **3**

6. Ejercicio 6: máxima distancia entre nodos **3**

7. Ejercicio 7 **4**

8. Ejercicio 8 **4**

9. Ejercicio 9: i-ésimo merge **4**

10. Ejercicio 10 **5**

11. Ejercicio 11 **5**

12. Ejercicio 12 **5**

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: índice espejo

ÍndiceEspejo(**in** A: arreglo(nat), low: nat, high: nat) → **out** res: bool

Pre ≡ $\{\forall i : (1 \leq i < tam(A)) \Rightarrow a_i < a_{i+1} \wedge \text{low} = 0 \wedge \text{high} = tam(A)\}$

```
1: if low > high then
2:     res ← false                                  ▷ Nos pasamos y no encontramos nada
3: end if
4: mid ← (low + high) div 2                         ▷ O(1)
5: pivote ← A[mid]                                  ▷ O(1)
6: if mid − pivote > 0 then
7:     res ← ÍndiceEspejo(A, mid+1, high)           ▷ Buscamos a la derecha
8: end if
9: if mid − pivote < 0 then
10:    res ← ÍndiceEspejo(A, low, mid-1)            ▷ Buscamos a la izquierda
11: end if
12: res ← true                                      ▷ Caso mid-pivote == 0, encontramos una solución
```

Complejidad:
Sea $n = tam(A)$, $a = 1$, $b = 2$, $f(n) = cte \in \Theta(1)$
$T(n) = T(n/2) + cte$
$f(n) \in \Theta(n^{log_b(a)}) = \Theta(n^{log_2(1)}) = \Theta(1)$
$\Rightarrow T(n) = \Theta(n^{log_b(a)} log(n)) = \Theta(log(n))$

Implementación en Go: https://play.golang.org/p/DdY_WfUMLZD

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: máxima distancia entre nodos

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: i-ésimo merge

IesimoMerge(**in** A: arreglo(nat), **in** B: arreglo(nat), **in** i: nat) → **out** res: nat

```
1: izq ← 1
2: der ← tam(A)
3: res ← 0
4: while izq < der ∧ res = 0 do
5:     mid ← izq + floor((der - izq) / 2)
6:     posInsercion ← BuscarPosInsercion(B, A[mid])
7:     iMerge ← mid + posInsercion
8:     if iMerge = i then
9:         res ← mid
10:    else if iMerge < i then
11:        der ← mid - 1
12:    else
13:        izq ← mid + 1
14:    end if
15: end while
16: if res ≠ 0 then
17:     res ← A[res]
18: else
19:     res ← IesimoMerge(B, A, i)
20: end if
```

Complejidad:

BuscarPosInsercion(**in** A: arreglo(nat), **in** e: nat) → **out** res: nat

```
1: izq ← 1
2: der ← tam(A)
3: while izq < der do
4:     mid ← izq + floor((der - izq) / 2)
5:     if e < A[mid] then
6:         der ← mid - 1
7:     else
8:         izq ← mid + 1
9:     end if
10: end while
11: res ← izq
```

Complejidad:

Ejemplo de ejecución

```
IesimoMerge([1, 3, 5], [2, 4, 6], 5)
    M = Merge([1, 3, 5], [2, 4, 6]) = [1, 2, 3, 4, 5, 6]
    M[5] = 5
IesimoMerge([1, 2, 3, 4, 5], [6, 7, 8, 9, 10], 2)
    M = Merge([1, 2, 3, 4, 5], [6, 7, 8, 9, 10]) = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    M[2] = 2
```

i = 5 [1, 2, 3, 4, 5] [6, 7, 8, 9, 10]

10. Ejercicio 10

Pendiente.

11. Ejercicio 11

Pendiente.

12. Ejercicio 12

Pendiente.