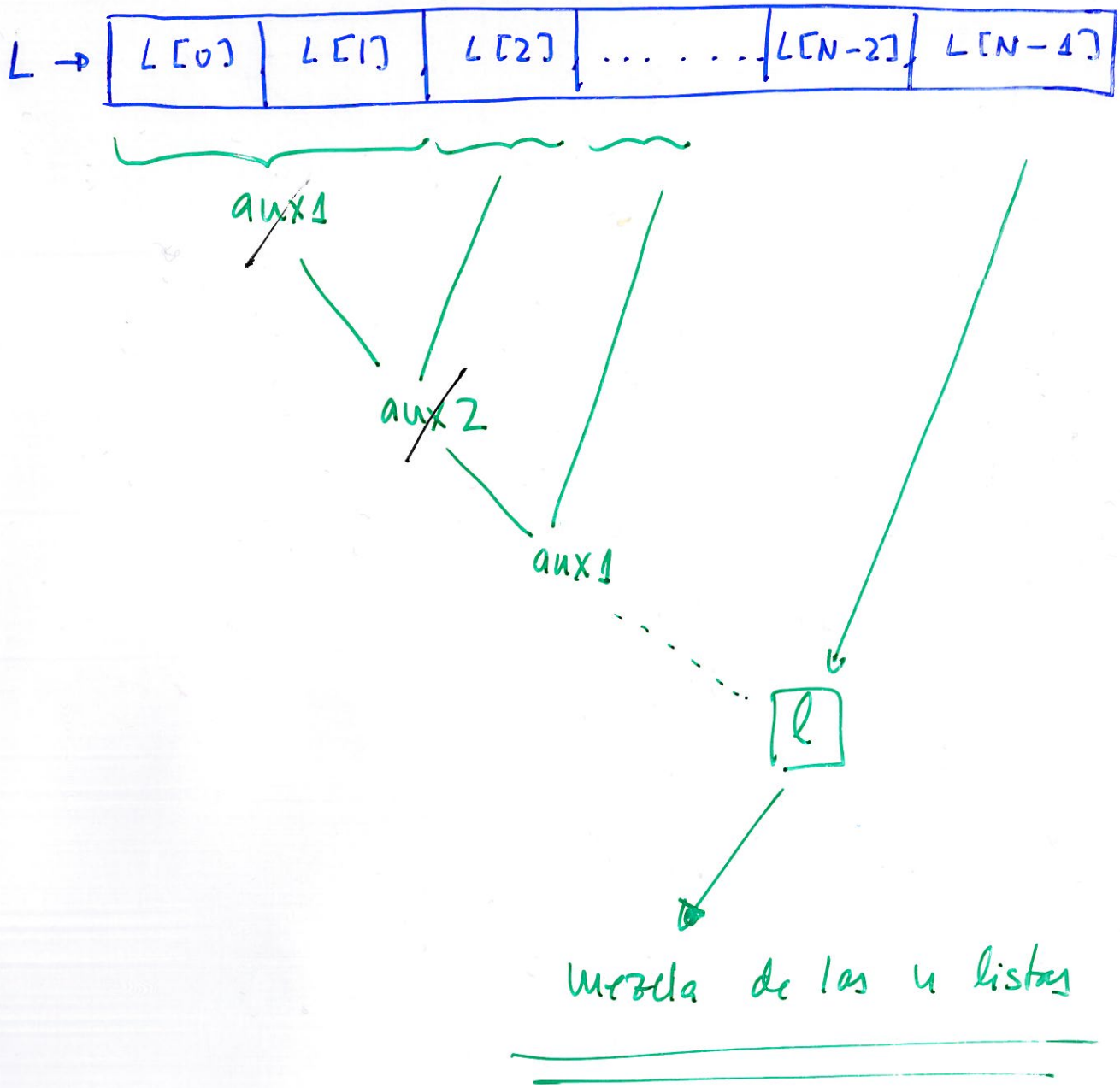

L →
L[0]
L[1]
L[2]
. . . . . .
L[N-2]
L[N-1]
aux1
aux2
aux1
l
Mezcla de las 4 listas

```
tlista mezcla (tlista * listas, int n)
{
  tlista l; tlista aux1, aux2; int i;
  if (n==2) l = mezcla2 (listas[0], listas[1]);
  else {
      aux1 = mezcla2 (listas[0], listas[1]); /* mezcla de los 2 primeros */
      for (i=2; i<n-1; i++) /* mezcla de los restantes menos el último */
        if (i%2) {   /* impar */
                  aux1 = mezcla2 (listas[i], aux2);
                  destruir (aux2);
                }
        else {  /* par */
                  aux2 = mezcla2 (listas[i], aux1);
                  destruir (aux1);
                }
      if ((n-1)%2)      /* mezcla del último */
        {
          l = mezcla2 (listas[(n-1)], aux2);
          destruir (aux2);
        }
      else {
          l = mezcla2 (listas[(n-1)], aux1);
          destruir (aux1);
        }
  }
  return l;
}
```

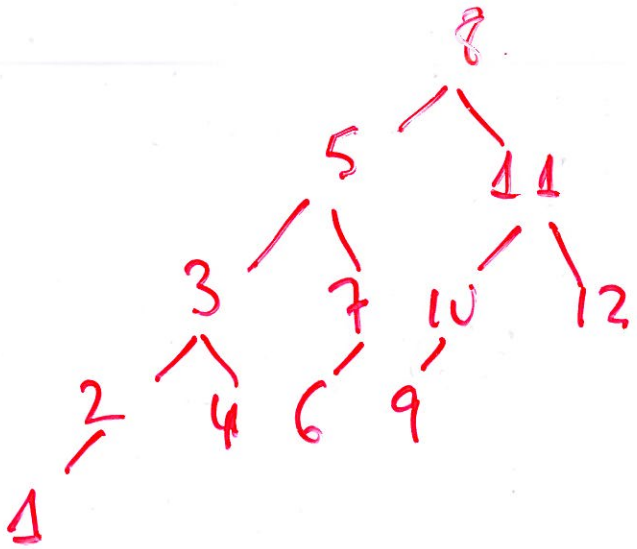
¿Son válidas estas funciones hash?

$h(k) = [k/n]$

$h(k) = 1$

$h(k) = (k + random(n)) \% n$

Construir un AVL que contenga como etiquetas los enteros del 1 al 12 y que tenga la mayor profundidad posible.

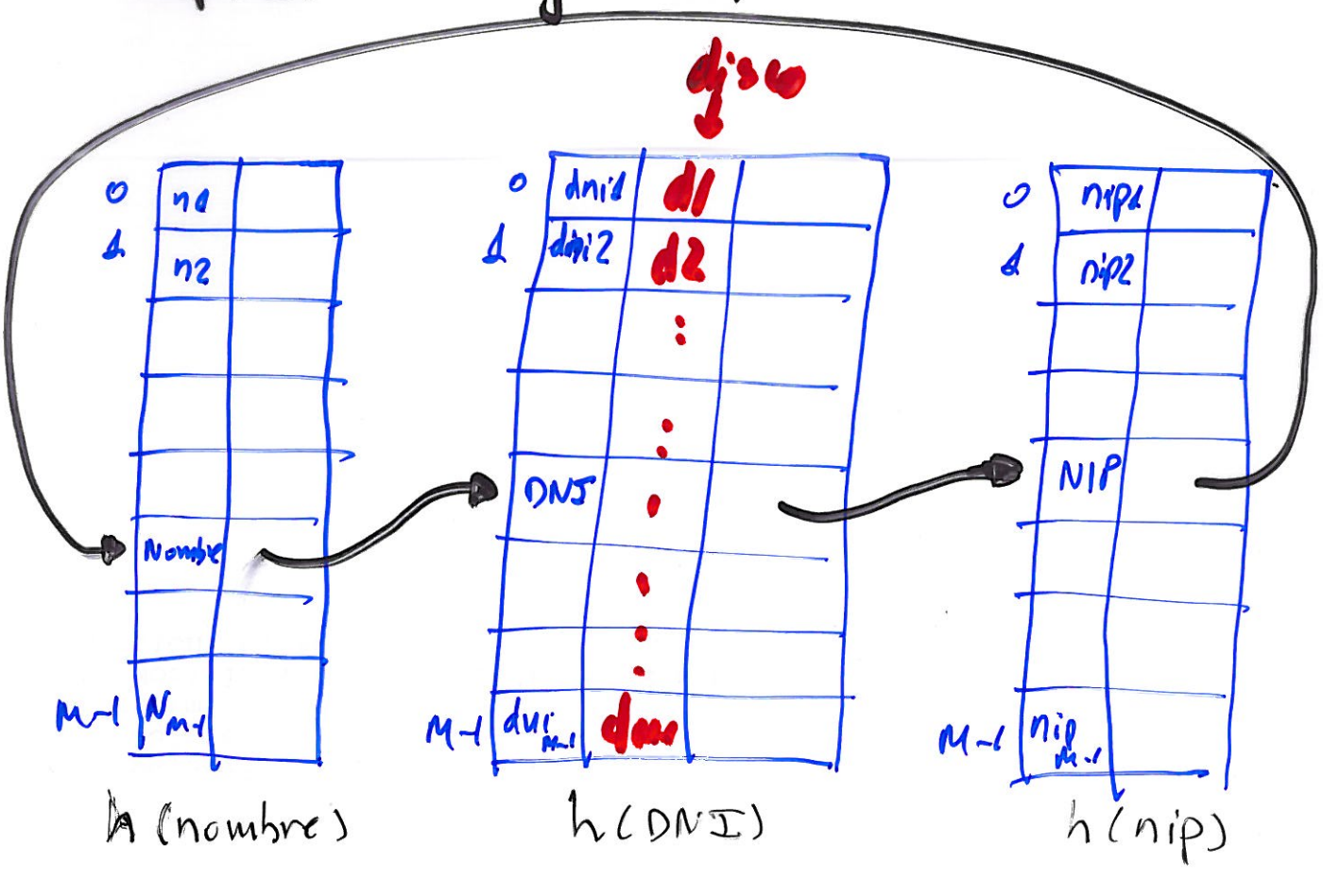


Prof = 4

Junio 2010

5 b)

Los empleados de una empresa se representan en una base de datos por su nombre, DNI y número de identificación personal como identificadores únicos junto con la información adicional del empleado. Construir una estructura de tablas hash que permita acceder en un tiempo O(1) en media al registro de un empleado por cualquiera de esos 3 campos teniendo en cuenta que no hay espacio para duplicar los registros.



Febrero 2013

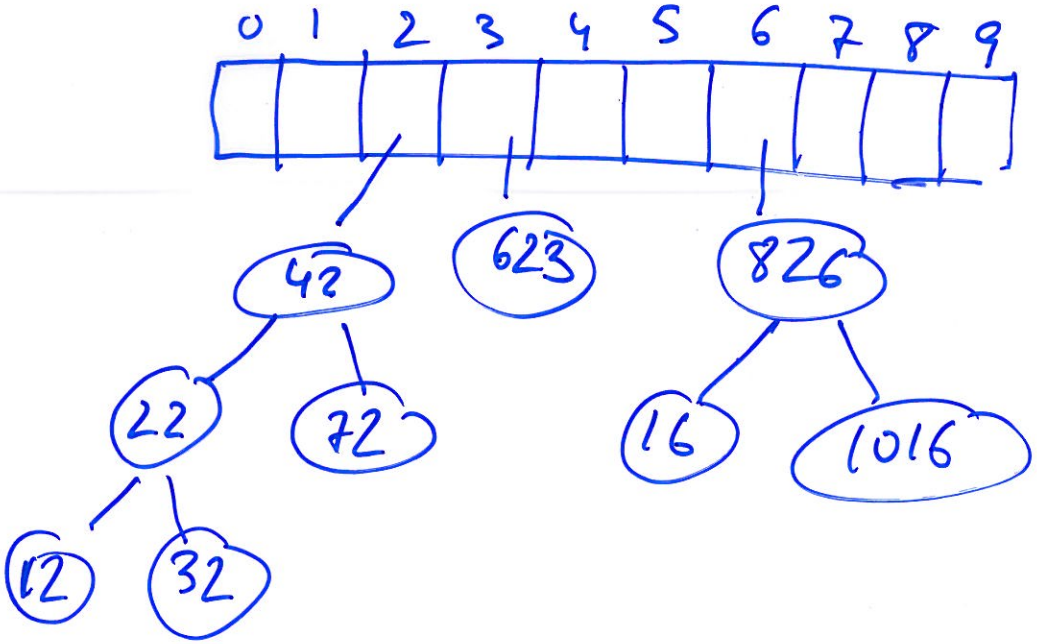
Problema 5

Supongamos que en una Tabla Hash hacemos uso en cada cubeta de AVL en lugar de listas.

* ¿Cual es la eficiencia (peor caso) de la función buscar?
* Mostrar la tabla hash resultante de insertar los elementos:

x	16	72	826	1016	12	42	623	22	32
h(x)	6	2	6	6	2	2	3	2	2

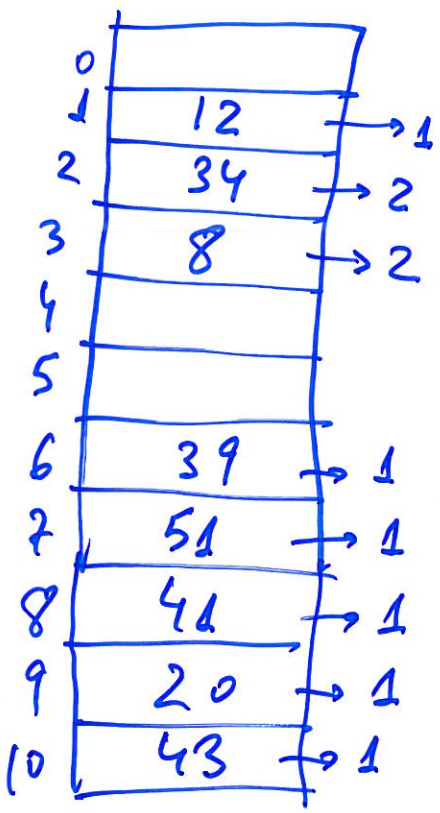
¿h(x)?



Insertar las claves {12, 41, 8, 34, 51, 20, 43, 39} en una tabla hash cerrada de tamaño 11 usando como función hash $h(k) = k \% 11$ y para resolver colisiones, rehashing doble usando como función hash secundaria:

$h_2(k) = 7 - (k \% 7)$ ¿Rendimiento?

	12	41	8	34	51	20	43	39
$h_1(k)$	1	8	8	1	7	9	10	6
$h_2(k)$	2	1	6	1	5	1	6	3



$Rend = \frac{10}{8} = 1.25$ int/clau

Rellenar la tabla con la eficiencia de las operaciones de las filas en las estructuras de datos de las columnas. (T → int)

	Vect. ord	List. ord	ABB	AVL	Pila	T. Hash
Encontrar mínimo						
Encontrar máximo						
#[a,b] a<b						
Imprimir ordenad.						

	Vector Ordenado	Lista ordenada	ABB	AVL	Pila	Tabla Hash
Encontrar mínimo	$O(1)$	$O(1)$	$O(n)$	$O(\log_2 n)$	$O(n)$	$O(n)$
Encontrar máximo	$O(1)$	$O(1)$	$O(n)$	$O(\log_2 n)$	$O(n)$	$O(n)$
#[a,b] a<b	$O(\log_2 n)$	$O(n)$*	$O(n)$	**	$O(n)$	$O(n)$
Imprimir ordenad	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n \log_2 n)$***	$O(n \log_2 n)$***

* Sin algoritmo especial

** $O(\max(\log_2 n, \#[a,b]))$

*** Con heapsort p. ej.

Junio 2010

5.b 2

¿Cuál de las siguientes funciones hash te parece más apropiada para insertar elementos en una tabla hash abierta / cerrada? ¿Por qué?

a) $h(K) = K \bmod (M \times N)$ M, N primos

b) $h(K) = K^2 \bmod M$ M primo

c) $h(K) = (M - (K \bmod M)) - 1$ M primo

(a) $M \times N$ no es primo

(b) OK

(c) OK

Aquí todas las posiciones son válidas

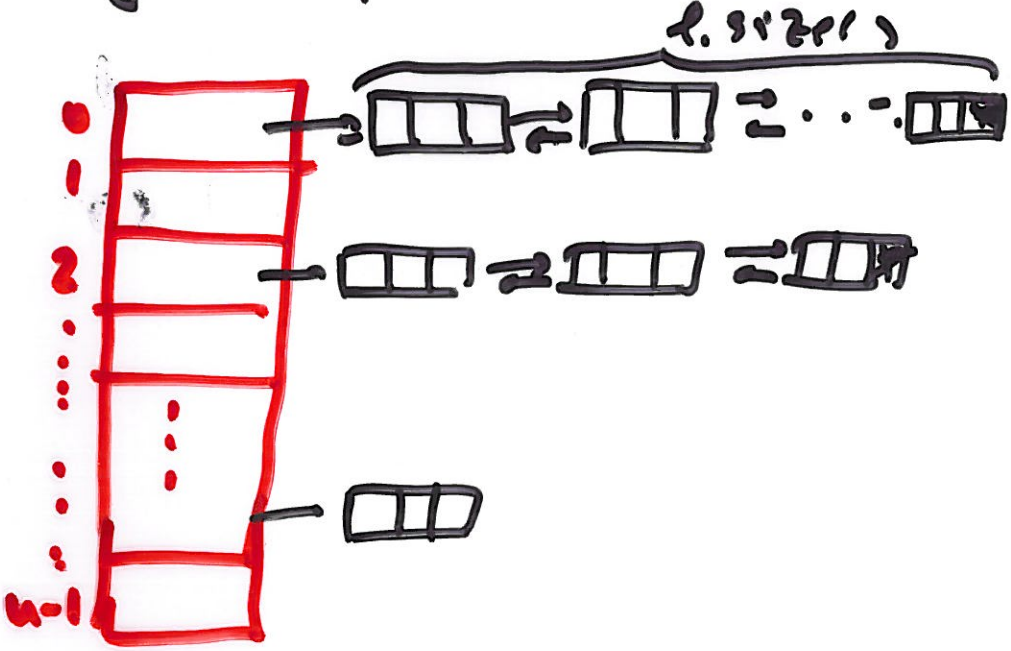
$K \,\%\, M = 0 \rightarrow h(K) = M - 1$

$K \,\%\, M = M - 1 \rightarrow h(K) = (M - (M-1)) - 1 = 0$

Rango $[0, M-1]$

Sep-2010

5.6 Se define el índice radial de una tabla hash abierta como el número de cubetas de la tabla multiplicado por el tamaño de la cubeta con mayor número de elementos. Construir un algoritmo para calcular dicho índice radial



Algoritmo

1. Determinar el tamaño del vector (número de cubetas): n (V.size())
2. Activar la función size() para cada una de las listas V[i] y encontrar el maxsize
3. Multiplicar n * maxsize

$\{3, 10, 29, 66, 127\}$

$h_1(k) = k \% 7$

$h_i(k) = [h_{i-1}(k) + h_0(k)] \% 7 \quad i = 2, 3 \ldots$

$h_0(k) = 1 + k \% 5$

	3	10	29	66	127
$h_1(k)$	3	3	1	3	1
$h_0(k)$	4	1	5	2	3

$h_1(3) = 3 \% 7 = 3$ $\quad h_0(3) = 1 + 3 \% 5 = 4$

$h_1(10) = 10 \% 7 = 3$ $\quad h_0(10) = 1 + 10 \% 5 = 1$

$h_1(29) = 29 \% 7 = 1$ $\quad h_0(29) = 1 + 29 \% 5 = 5$

$h_1(66) = 66 \% 7 = 3$ $\quad h_0(66) = 1 + 66 \% 5 = 2$

$h_1(127) = 127 \% 7 = 1$ $\quad h_0(127) = 1 + 127 \% 5 = 3$

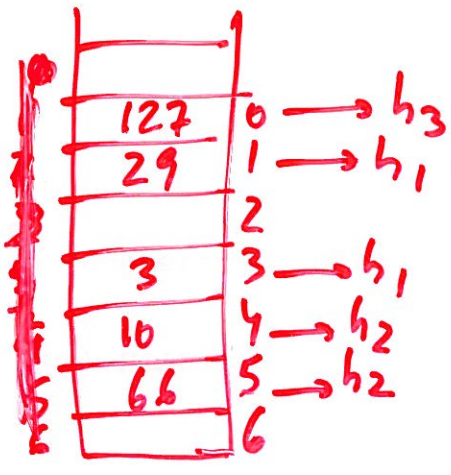
Colisiones

$h_2(10) = [h_1(10) + h_0(10)] \% 7 = (3+1) \% 7 = 4$

$h_2(66) = [h_1(66) + h_0(66)] \% 7 = (3+2) \% 7 = 5$

$h_2(127) = [h_1(127) + h_0(127)] \% 7 = (1+3) \% 7 = 4$

$h_3(127) = [h_2(127) + h_0(127)] \% 7 = (4+3) \% 7 = 0$



TEST

1. El TDA cola con prioridad no es más que un caso particular del TDA cola
2. El orden en que las hojas se listan en los recorridos pre, in y post en un árbol Binario es el mismo en los 3 casos
3. Un AVL puede reconstruirse de forma unívoca dado su recorrido en inorden
4. Es imposible que un árbol Binario sea AVL y APO a la vez
5. En un esquema de hashing doble puede ocurrir que para 2 claves $k_1 \neq k_2$ ocurra simultáneamente que $\begin{cases} h(k_1) = h(k_2) \\ h_0(k_1) = h_0(k_2) \end{cases}$
6. Un analista tiene que resolver un problema de tamaño 100 y para ello utiliza un algoritmo $O(n^2)$. Uno de sus colaboradores consigue obtener una solución que es $O(n)$. El analista debe olvidarse de su primera solución y usar la de su colaborador de mejor eficiencia en tiempo

7. Si la eficiencia de un algoritmo viene dada por la función $f(n) = 1 + 2 + \dots + n$, ese algoritmo es $O(\max\{1, 2, \dots, n\})$ es decir $O(n)$

8. Puede hacerse esta definición

stack<int>::iterator P;

9 Un ABO puede reconstruirse de forma unívoca dado su recorrido en postorden

10. En un esquema de hashing doble es correcto usar como función hash secundaria la función:

$h_0(x) = [(B-1) - (x \,\%\, B)] \,\%\, B$ B primo

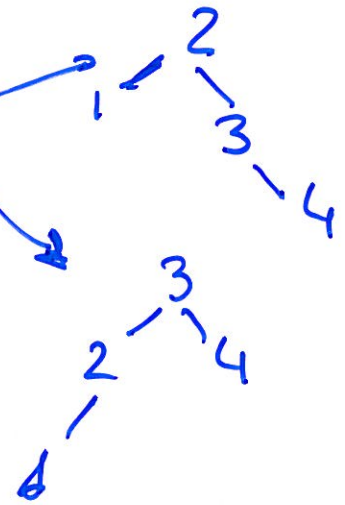
1. Falso

2. Verdadero. El orden en que los hijos se listan en los 3 recorridos es el mismo y en particular el orden de las hojas (siempre de izqda a drcha)

3. Falso.

Contraejemplo:

Inorden: 1, 2, 3, 4



4. Verdadero salvo casos degenerados (todas las claves iguales o árboles de profundidad 1). Las condiciones de definición de los 2 árboles dan lugar a una contradicción:

E(padre) < E(hijo drcha) } ABO

E(padre) < E(hijo izqda) }

↓ ↯

E(padre) > E(hijo izqda) } AVL

E(padre) < E(hijo drcha) }

Imposible

5 Falso. Si que puede ocurrir:

$h(k) = k \% 5$

$h_0(k) = 3 - k \% 3$

k	3	18
$h(k)$	3	3
$h_0(k)$	3	3

o

$h(k) = k \% 7$

$h_0(k) = 1 + k \% 5$

k	70	35
$h(k)$	0	0
$h_0(k)$	1	1

6 Falso. La eficiencia en espacio y las constantes pueden influir en ejemplos de tamaño pequeño

7. Falso. El algoritmo es $\Theta(n^2)$

8 Falso. Una pila no tiene iteradores

9. Verdadero. Por la condición geométrica de los ABO

10. No es una función correcta porque puede llegar a tomar el valor $\emptyset$.

4. Para cada función $f(n)$ y cada tiempo t de la tabla siguiente, determinar el mayor tamaño de un problema que puede ser resuelto en un tiempo t (suponiendo que el algoritmo para resolver el problema tarda $f(n)$ microsegundos, es decir, $f(n) \times 10^{-6}$ sg.)

$f(n)$	t				
	1 sg.	1 h.	1 semana	1 año	1000 años
$\log_2 n$	$\approx 10^{300000}$	$10^{10^{12}}$	$10^{1.82 \cdot 10^{10}}$	$10^{94.9 \cdot 10^{11}}$	$10^{9.5 \cdot 10^{15}}$
n	10^6	$3.6\,10^8$	$6048\,10^8$	$\approx 3{,}15 \times 10^{15}$	$3.15\,10^{15}$
$n \log_2 n$	62.74	$\approx 1{,}33 \times 10^8$	$1.77\,10^{10}$	$7.98 \cdot 10^{11}$	$6.41 \cdot 10^{14}$
n^3	$3.3 \cdot 10^5$	1533	8457	~~146645~~ 31594	$3.1594 \cdot 10^5$
2^n	19	31.7	39.1	44.8	54.8
$n!$	10	12	15	17	19