Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2133 — Estructuras de Datos y Algoritmos
2020 - 2

# Interrogación 1

## Pregunta 1

a) Supongamos que al ejecutar Quicksort sobre un arreglo particular, la subrutina partition hace siempre el mayor número posible de intercambios; ¿cuánto tiempo toma Quicksort en este caso? ¿Qué fracción del mayor número posible de intercambios se harían en el mejor caso? Justifica

b) El algoritmo quicker-sort llama a la subrutina pq-partition, que utiliza dos pivotes $p$ y $q$ $(p < q)$ para particionar el arreglo en 5 partes: los elementos menores que $p$, el pivote $p$, los elementos entre $p$ y $q$, el pivote $q$, y los elementos mayores que $q$. Escribe el pseudocódigo de pq-partition. ¿Es quicker-sort más eficiente que quick-sort? Justifica.

### Solución Pregunta 1a)

La subrutina partition para arreglos que vimos en clases:

```
partition(A, i, f):
    x ← un indice aleatorio en [i, f]
    p ← A[x]
    A[x] ⇄ A[f]
    j ← i
    for k ∈ [i, f − 1]:
        if A[k] < p:
            A[j] ⇄ A[k]
            j ← j + 1
    A[j] ⇄ A[f]
    return j
```

Podemos ver que hay dos intercambios que siempre se hacen, que son poner el pivote al final del arreglo y luego moverlo a la posición que separa los menores de los mayores.

Los demás intercambios se hacen solo cuando encontramos una clave que es menor al pivote. Por lo tanto, en el caso de que se hagan la mayor cantidad de intercambios es cuando todas las claves del subarreglo son menores al pivote.

Esto significa que si el subarreglo tenía $\boldsymbol{m}$ elementos, se hacen $\boldsymbol{m+1}$ intercambios, y el subarreglo se separa en dos subarreglos disparejos, uno de largo $\mathbf{0}$ y el otro de largo $\boldsymbol{m-1}$.

Esto significa que el tiempo que toma Quicksort para un arreglo inicial de $\boldsymbol{n}$ elementos es:

$$\boldsymbol{T(n) = n + (n-1) + (n-2) + \cdots + 1}$$

$$\boldsymbol{T(n) \in \mathcal{O}(n^2)}$$

El cual es el peor caso de Quicksort.

Por otro lado, sabemos que el mejor caso de Quicksort se produce cuando partition separa el subarreglo en dos mitades de el mismo largo (o lo más parejo posible).

Esto significa que la mitad de los elementos del subarreglo deben ser mayores al pivote y otra mitad menores al pivote. Ya vimos que los intercambios se hacen sólo para los elementos menores al pivote, por lo que en este caso tendríamos **la mitad** de intercambios que en el caso anterior.

[**1pt**] Por identificar que implican los intercambios en partition

[**1pt**] Por identificar correctamente el peor caso

[**1pt**] Por identificar correctamente fracción en el mejor caso

[**Era posible interpretar esta pregunta como *cuál es la fracción de intercambios que se efectúan en Quicksort*, en lugar de cada llamada a partition. Esta interpretación es válida y se evalúa de manera correspondiente. No basta con el resultado final, debe tener los pasos intermedios correctos.**]

## Solución Pregunta 1b)

Ya que la definición dice que $p < q$, entonces es posible asumir que no hay datos repetidos. De todos modos, esto no afecta el análisis.

Lo más simple es implementar partition con listas ligadas como vimos en clases:

```
1:  procedure PQ-PARTITION(lista ligada L)
2:      p, q ← dos nodos de L tal que p < q. Quitar estos nodos de L.
3:      A, B, C ← listas vacias   ▷ Los elementos menores a p, entre p y q y mayores a q respectivamente
4:      for nodo x ∈ L do
5:          if x < p then
6:              agregar x al final de A
7:          else if x < q then
8:              agregar x al final de B
9:          else
10:             agregar x al final de C
11:         end if
12:     end for
13:     return A, p, B, q, C
14: end procedure
```

El paso en la linea 2 es fácil de hacer en $\mathcal{O}(1)$, basta con extraer los primeros dos elementos de $L$, e intercambiarlos si $p > q$.

Recordemos que el peor caso de Quicksort se produce cuando partition separa una secuencia de $m$ datos en dos secuencias disparejas, de $0$ y $m-1$ datos cada una. En ese caso la complejidad para una secuencia inicial de $n$ datos es de:

$$T(n) = n + (n-1) + (n-2) + \cdots + 1$$

$$T(n) \in \mathcal{O}(n^2)$$

En este caso esto también puede suceder, solo que se separa en 3 secuencias disparejas, de $0$, $0$ y $m-2$ elementos cada una. En este caso la complejidad para una secuencia inicial de $n$ datos es de:

$$T(n) = n + (n-2) + (n-4) + \cdots + 1$$

$$T(n) \in \mathcal{O}(n^2)$$

Ambos algoritmos son iguales en el peor caso, así que no podemos afirmar que quickersort sea mejor que quicksort.

[**1pt**] Por el pseudocódigo

[**1pt**] Por calcular la complejidad de quicker-sort

[**1pt**] Por argumentar cual de los dos es mejor [varía según el argumento]

# Pregunta 2

a) Escribe un algoritmo $\mathcal{O}(n)$ para convertir un ABB de $n$ claves en un min-Heap. Demuestra que es correcto.

b) ¿Por qué encontrar el elemento de menor prioridad en un Heap de $n$ elementos requiere revisar solo $\frac{n}{2}$ elementos?

## Solución Pregunta 2a)

Recordemos que un ABB tiene un orden total de los datos, por lo que podemos obtener las claves ordenadas haciendo un recorrido *in-order*, el cual es $\mathcal{O}(n)$. En clases se discutió como hacer un recorrido *in-order* del árbol para imprimir las claves en orden, de la siguiente manera:

```
1: procedure InOrder(árbol T)
2:     if T es un árbol then
3:         InOrder(T.left)
4:         imprimir T.key
5:         InOrder(T.right)
6:     end if
7: end procedure
```

Necesitamos modificar esta función para que en lugar de imprimir las claves en orden, nos entregue una lista ligada o arreglo con las claves en orden. Lo más simple es hacerlo sobre una lista:

```
1: procedure InOrder(árbol T, lista ligada L)
2:     if T es un árbol then
3:         InOrder(T.left)
4:         agregar (T.key, T.value) al final de L
5:         InOrder(T.right)
6:     end if
7: end procedure
```

Tras llamar a InOrder($T$, $L$), tenemos en la lista $L$ todos los pares $(key, value)$ de $T$, ordenados de menor a mayor clave. Esta lista podemos trivialmente convertirla a un arreglo $A$ en $\mathcal{O}(n)$.

Notar que $A$ está ordenado de menor a mayor, por lo que cumple las propiedades de *min-Heap*. Esto es, para cada elemento en posición $\alpha$ se tiene que $A[\alpha] \leq A[2\alpha + 1]$ y $A[\alpha] \leq A[2\alpha + 2]$. Por lo tanto $A$ es el heap que buscamos (almacenado como un arreglo)

Finalmente, se tiene que el algoritmo es correcto puesto a que utiliza dos procesos finitos de $O(n)$ pasos y entrega el output deseado, ya que el array $A$ cumple las propiedades de *Heap*.

**[1pt]** Por poner los datos del árbol ordenadamente en una lista o arreglo

**[1pt]** Por argumentar que un arreglo ordenado cumple con la propiedad de Heap

**[1pt]** Por demostrar que el algoritmo propuesto es correcto

**Si el algoritmo propuesto no es $\mathcal{O}(n)$ entonces el puntaje máximo a obtener es 1pt**

Observaciones:

- En caso de errores pequeños en el algoritmo, descontar 0.5
- Si para convertir el árbol en una lista usan el algoritmo de las rotaciones visto en la ayudantía, también es válido.

## Solución Pregunta 2b)

El elemento de menor prioridad en un heap no puede tener hijos, ya que de tenerlos, sus prioridades deberían ser aun menores.

Por lo tanto, si el elemento de menor prioridad está en una de las hojas del heap, pero no sabemos cual.

Eso significa que para encontrarlo, sería necesario revisar cada hoja.

Por lo tanto, necesitamos saber cuantas hojas tiene.

Haciendo las inserciones por nivel (para que el heap sea lo más balanceado posible), tenemos que la cantidad de hojas en función de $\boldsymbol{n}$ es:

$\boldsymbol{h(1) = 1}$, donde la raíz es hoja.

$\boldsymbol{h(2) = 1}$, donde la nueva hoja cuelga de una hoja con $\boldsymbol{n = 1}$

$\boldsymbol{h(3) = 2}$, donde la nueva hoja es ahora hermana de la hoja que apareció en $\boldsymbol{n = 2}$

$\boldsymbol{h(4) = 2}$, donde la nueva hoja cuelga de una hoja con $\boldsymbol{n = 3}$

$\boldsymbol{h(5) = 3}$, donde la nueva hoja es ahora hermana de la hoja que apareció en $\boldsymbol{n = 4}$

Esto sigue así:

$\boldsymbol{h(6) = 3}$

$\boldsymbol{h(7) = 4}$

$\boldsymbol{h(8) = 4}$

$\boldsymbol{h(9) = 5}$

Por lo que $\boldsymbol{h = \lceil \frac{n}{2} \rceil}$.

[**1.25pt**] Por identificar que la clave de menor prioridad es una hoja

[**1.25pt**] Por contar la cantidad de hojas en función de $\boldsymbol{n}$

[**0.5pt**] Por la justificación

# Pregunta 3

a) Sea $\boldsymbol{T}$ un ABB de altura $\boldsymbol{h}$. Escribe un algoritmo que posicione un elemento arbitrario de $\boldsymbol{T}$ como raíz de $\boldsymbol{T}$ en $\boldsymbol{\mathcal{O}(h)}$ pasos.

b) Sean $\boldsymbol{T_1}$ y $\boldsymbol{T_2}$ dos ABB de $\boldsymbol{n}$ y $\boldsymbol{m}$ nodos respectivamente. Explica, de manera clara y precisa, cómo realizar un merge entre ambos árboles en $\boldsymbol{\mathcal{O}(n+m)}$ pasos para dejarlos como un solo ABB balanceado $\boldsymbol{T}$ con los nodos de ambos árboles.

## Solución Pregunta 3a)

La solución puede o no considerar el proceso de búsqueda del nodo solicitado (de ahora en adelante, $\boldsymbol{y}$). Este paso tiene complejidad $\boldsymbol{\mathcal{O}(h)}$, por lo que sigue dentro de la cota especificada.

Tras este proceso, es necesario hacer rotaciones de tal forma que $\boldsymbol{y}$ quede en la raíz del árbol. Esto se puede realizar utilizando las rotaciones AVL vistas en clases. A continuación se propone un pseudocódigo.

```
1:  procedure VolverRaiz(nodo y)
2:      while y tiene padre do
3:          x ← y.padre
4:          if y es hijo izquierdo de su padre then
5:              rotar hacia la derecha en torno a x-y        ▷ esto modifica el padre de y
6:          else
7:              rotar hacia la izquierda en torno a x-y      ▷ esto modifica el padre de y
8:          end if
9:      end while
10: end procedure
```

Vimos en clases que cada rotación es $\boldsymbol{\mathcal{O}(1)}$ y en cada una de estas $\boldsymbol{y}$ sube un nivel. En el peor caso $\boldsymbol{y}$ es una hoja por lo tanto es necesario hacerlo subir $\boldsymbol{h}$ niveles $\boldsymbol{\implies}$ se hacen $\boldsymbol{\mathcal{O}(h)}$ rotaciones.

[**1pt**] Por identificar que una rotación permite hacer subir el nodo $\boldsymbol{y}$ en un nivel.

[**1pt**] Por escribir el algoritmo correctamente.

[**1pt**] Por justificar la complejidad, indicando que cada rotación es $\boldsymbol{\mathcal{O}(1)}$.

**Si el algoritmo propuesto no es $\boldsymbol{\mathcal{O}(h)}$ entonces el puntaje máximo a obtener es 1pt**

## Solución Pregunta 3b)

La pregunta no solicita un algoritmo, por lo que basta con describir el proceso:

1. Se itera por sobre los nodos de ambos árboles de manera ordenada, copiando los nodos a un array. Este paso consiste en un proceso recursivo que visita siempre el nodo izquierdo antes que el derecho (recorrido *in-order* o algoritmo visto en la ayudantía, ver solución pregunta 2a). Haciendo esto tenemos dos arreglos ordenados, con los elementos de $\boldsymbol{T_1}$ y $\boldsymbol{T_2}$ respectivamente.
2. Combinamos estos dos arreglos usando la subrutina ***merge*** de MergeSort en $\boldsymbol{\mathcal{O}(n+m)}$. Con esto obtenemos un arreglo ordenado $\boldsymbol{A}$ con los $\boldsymbol{n+m}$ elementos de ambos árboles.
3. Finalmente, se convierte el array ordenado $\boldsymbol{A}$ en un ABB balanceado poniendo la mediana como raiz e insertando los valores menores y mayores a esta en las ramas izquierdas y derechas respectivamente. Notar que encontrar la mediana en un array ordenado es $\boldsymbol{O(1)}$ por lo que proceso se realiza en $\boldsymbol{O(m+n)}$ pasos.

La solución consta de tres pasos $\boldsymbol{O(m+n)}$, por lo que la rutina completa es $\boldsymbol{O(m+n)}$.

[**1pt**] Por explicar como obtener el arreglo ordenado con todos los datos

[**1pt**] Por explicar como crear un árbol balanceado a partir de un arreglo ordenado

[**1pt**] Por justificar la complejidad del proceso

**Si el proceso propuesto no es $\boldsymbol{\mathcal{O}(n+m)}$ entonces el puntaje máximo a obtener es 1pt**

# Pregunta 4

a) Cualquier árbol AVL es también un árbol Rojo-Negro; solo hace falta pintar adecuadamente los nodos del AVL de color rojo o negro. Explica cómo decidir de qué color pintar cada nodo de un árbol AVL para que el árbol resultante sea un árbol Rojo-Negro (válido).

b) Calcula la profundidad a la que se encuentra la hoja menos profunda en un AVL de altura $\boldsymbol{h}$ lo más desbalanceado posible. ¿Cómo se relaciona con la profundidad a la que se encuentra la hoja menos profunda en un árbol Rojo-Negro de altura $\boldsymbol{h}$, también lo más desbalanceado posible? ¿Para qué valores de $\boldsymbol{h}$ estas dos profundidades son iguales?

## Solución Pregunta 4a)

Para poder pintar un árbol AVL de rojo negro, primero debemos recordar las siguientes propiedades:

- En un árbol AVL, las alturas de dos sub-árboles hermanos difieren en 0 o 1.
- En un árbol Rojo-Negro, la ruta de la raíz a cada hoja debe contener la misma cantidad de nodos negros. En particular, dos sub-árboles hermanos deben tener la misma cantidad de nodos negros camino a cada hoja.

Sabiendo esto, debemos preguntarnos como relacionar la altura de los hermanos con la cantidad de nodos negros camino a sus hojas.

Para un sub-árbol de altura $\boldsymbol{h}$, denotaremos $\boldsymbol{n}$ como la cantidad de nodos negros camino a cada hoja. Veamos como se relacionan $\boldsymbol{h}$ y $\boldsymbol{n}$.

El mínimo valor de $\boldsymbol{n}$ para un $\boldsymbol{h}$ dado está dictado por su rama más larga, la cual tiene altura $\boldsymbol{h}$. Recordemos que cada nodo que no es negro debe ser rojo, y que por definición de árbol rojo-negro, en una rama no puede haber dos nodos rojos seguidos. Por lo que si el árbol tuviera menos nodos negros, obligaríamos a que la rama más larga tenga dos nodos rojos seguidos, lo que viola la propiedad de árbol rojo negro.

Si $\boldsymbol{h}$ es par, tenemos que la cantidad de nodos negros debe ser igual a la cantidad de nodos rojos.

$$\begin{aligned} \boldsymbol{h} &= \boldsymbol{2n} \\ \boldsymbol{min(n)} &= \frac{\boldsymbol{h}}{\boldsymbol{2}} \end{aligned}$$

Si $\boldsymbol{h}$ es impar, entonces tenemos el caso en que **la raíz es roja**[1] y la última hoja es roja, y luego intercalamos nodos rojos con negros. Por lo tanto:

$$\begin{aligned} \boldsymbol{h} &= \boldsymbol{2n+1} \\ \boldsymbol{2n} &= \boldsymbol{h-1} \\ \boldsymbol{min(n)} &= \frac{\boldsymbol{h}}{\boldsymbol{2}} - \frac{\boldsymbol{1}}{\boldsymbol{2}} \end{aligned}$$

[1] Recordar que esto es un sub-árbol, por lo que la raiz no necesariamente es negra

Ahora, el máximo de $\boldsymbol{n}$ para un $\boldsymbol{h}$ dado está dictado por su rama más corta posible, ya que si fuera mayor no sería posible cumplir la propiedad de árbol rojo negro en caso de existir una rama así en el árbol.

En la pregunta 4b calculamos la altura de la rama más corta posible:

Para $\boldsymbol{h}$ par,

$$\boldsymbol{max(n) = \frac{h}{2}}$$

Y para $\boldsymbol{h}$ impar, sabemos que **la raíz debe ser negra**, por lo que

$$\boldsymbol{max(n) = \frac{h}{2} + \frac{1}{2}}$$

Por lo tanto, tenemos el $\boldsymbol{n}$ que debe tener un árbol rojo negro para permitir todos los casos posibles de estructura interna, para $\boldsymbol{h}$ par,

$$\boldsymbol{n = \frac{h}{2}}$$

Y para $\boldsymbol{h}$ impar,

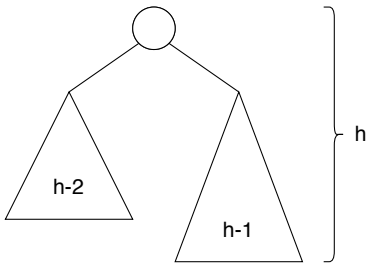
$$\boldsymbol{n = \frac{h}{2} \pm \frac{1}{2}}$$

Recordemos que un árbol AVL de altura $\boldsymbol{h}$ puede tener dos casos:

1. Ambos hijos son de igual altura, $\boldsymbol{h-1}$
2. Ambos hijos son de distinta altura, $\boldsymbol{h-1}$ y $\boldsymbol{h-2}$

Ahora debemos combinar esos casos con la paridad de las alturas.

## Caso 1: Distinta altura, $h-1$ par




Ambos hermanos deben tener igual $\boldsymbol{n}$.

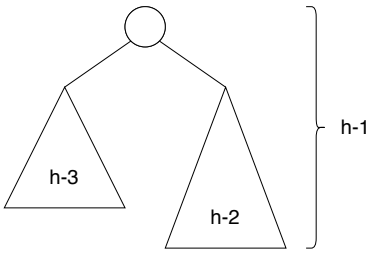
El $\boldsymbol{n}$ del hermano mayor, como $\boldsymbol{h-1}$ es par, es $\frac{\boldsymbol{h-1}}{\boldsymbol{2}}$

El $\boldsymbol{n}$ del hermano menor, como $\boldsymbol{h-2}$ es impar, es $\frac{\boldsymbol{h-2}}{\boldsymbol{2}} \pm \frac{\boldsymbol{1}}{\boldsymbol{2}}$

Ya que ambos deben ser iguales, entonces el hermano menor debe ser $\frac{\boldsymbol{h-2}}{\boldsymbol{2}} + \frac{\boldsymbol{1}}{\boldsymbol{2}}$, el cual es el caso en que la raíz de ese árbol es negra.

**Por lo tanto, la raíz del hermano menor debe ser negra.**

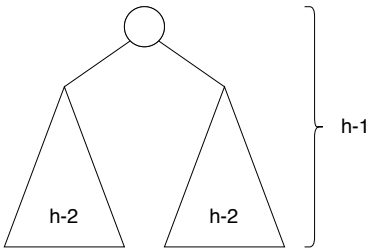
¿Qué hay del hermano mayor? Veamos sus hijos, los cuales pueden ser uno de los dos casos de un AVL. En primer lugar, cuando sus hijos son de distinta altura




Estos hermanos a su vez deben tener el mismo $\boldsymbol{n}$, el cual está dictado por el hermano de altura par $\boldsymbol{h-3}$. Este $\boldsymbol{n}$ es entonces $\frac{\boldsymbol{h-3}}{\boldsymbol{2}}$, el cual es exactamente el $\boldsymbol{n}$ de su padre, $\boldsymbol{-1}$.

Por lo tanto en este caso la raíz de este sub-árbol debe ser negra.

Cuando los hijos de este sub-arbol son de la misma altura, tenemos:




En este caso, ambos árboles son de altura impar, por lo que su $\boldsymbol{n}$ podría o no ser igual al del padre.

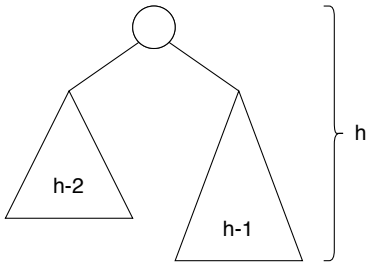
Cabe destacar que no existe una única manera de pintar un AVL a Rojo-Negro. En este caso, vamos a preferir el caso en que el $\boldsymbol{n}$ es lo menor posible[2], por lo que el $\boldsymbol{n}$ de estos hijos sería $\frac{\boldsymbol{n-2}}{\boldsymbol{2}} - \frac{\boldsymbol{1}}{\boldsymbol{2}}$. Este $\boldsymbol{n}$ es entonces $\frac{\boldsymbol{h-3}}{\boldsymbol{2}}$, el cual es exactamente el $\boldsymbol{n}$ de su padre, $\boldsymbol{-1}$.

Por lo tanto en este caso la raíz de este sub-árbol debe ser negra.

**Por lo tanto, la raíz del hermano mayor debe ser negra, independiente de como sean sus hijos**

[2]Para así minimizar la altura del 2-4 correspondiente

### Caso 2: Distinta altura, $h-1$ impar




Ambos hermanos deben tener igual $\boldsymbol{n}$.

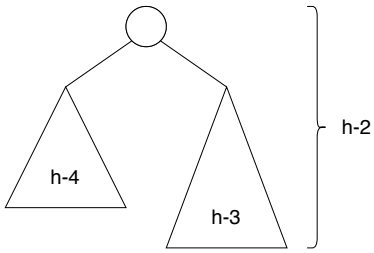
El $\boldsymbol{n}$ del hermano mayor, como $\boldsymbol{h-1}$ es impar, es $\frac{\boldsymbol{h-1}}{\boldsymbol{2}} \pm \frac{\boldsymbol{1}}{\boldsymbol{2}}$

El $\boldsymbol{n}$ del hermano menor, como $\boldsymbol{h-2}$ es par, es $\frac{\boldsymbol{h-2}}{\boldsymbol{2}}$

Ya que ambos deben ser iguales, entonces el hermano mayor debe ser $\frac{\boldsymbol{h-2}}{\boldsymbol{2}} - \frac{\boldsymbol{1}}{\boldsymbol{2}}$, el cual es el caso en que la raíz de ese árbol es roja.

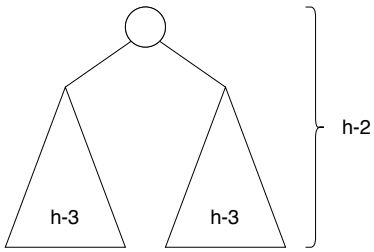
**Por lo tanto, la raíz del hermano mayor debe ser roja**

Para el hermano menor es necesario hacer el mismo análisis de sus hijos que hicimos para el caso 1.




El $\boldsymbol{n}$ de estos hermanos está determinado por el hermano par, $\boldsymbol{h-4}$, el cual sería $\frac{\boldsymbol{h-4}}{\boldsymbol{2}}$, el cual es exactamente 1 menos que el $\boldsymbol{n}$ de su padre.

En este caso entonces pintaríamos la raíz del hermano menor de negro.

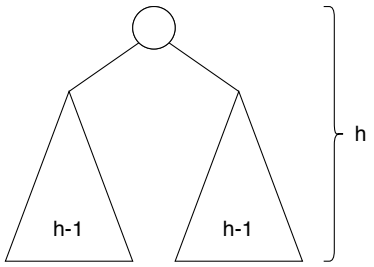



El $\boldsymbol{n}$ de estos hermanos es $\frac{\boldsymbol{h-3}}{\boldsymbol{2}} \pm \frac{\boldsymbol{1}}{\boldsymbol{2}}$, ya que $\boldsymbol{h-3}$ es impar. Nuevamente, elegimos el mínimo $\boldsymbol{n}$ posible y nos quedamos con $\frac{\boldsymbol{h-3}}{\boldsymbol{2}} - \frac{\boldsymbol{1}}{\boldsymbol{2}}$. Esto es exactamente 1 menos que el $\boldsymbol{n}$ de su padre.

En este caso entonces también pintaríamos la raíz del hermano menor de negro.

**Por lo tanto, la raíz del hermano menor debe ser negra, independiente de como sean sus hijos**

## Caso 3: Misma altura, $h - 1$ par




Ambos hermanos deben tener igual $\boldsymbol{n}$.

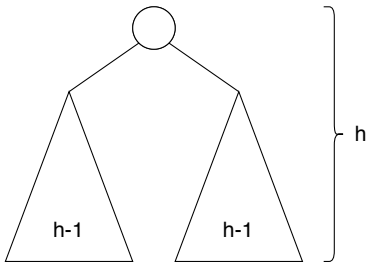
El $\boldsymbol{n}$ de ambos hermanos, como $\boldsymbol{h-1}$ es par, es $\frac{\boldsymbol{h-1}}{\boldsymbol{2}}$

Para determinar de que color pintar sus raíces, miramos a sus hijos.

Este exactamente el mismo análisis que para el hermano mayor del caso 1.

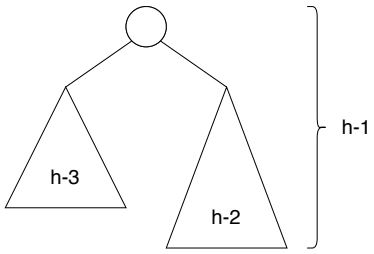
**Por lo tanto, la raíz de ambos hermanos debe ser negra, independiente de como sean sus hijos**

## Caso 4: Misma altura, $h - 1$ impar




El $\boldsymbol{n}$ de ambos hermanos, como $\boldsymbol{h-1}$ es par, es $\frac{\boldsymbol{h-1}}{\boldsymbol{2}} \pm \frac{\boldsymbol{1}}{\boldsymbol{2}}$

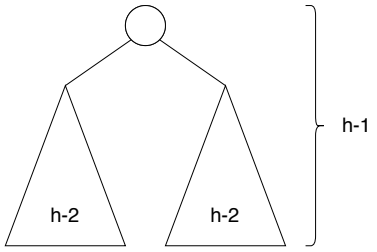
Veamos que pasa con sus hijos.



El $\boldsymbol{n}$ de estos hermanos está determinado por el hermano de altura par, $\boldsymbol{h-2}$, por lo que sería $\frac{\boldsymbol{h-2}}{\boldsymbol{2}}$

Esto podría ser menor o igual que el $\boldsymbol{n}$ de su padre. Nuevamente, vamos a preferir que todo $\boldsymbol{n}$ sea lo menor posible, por lo que ambos serían iguales.

Por lo tanto, en este caso las raíces de ambos hermanos en el primer árbol serían rojas.




Tenemos el mismo caso anterior: el $\boldsymbol{n}$ es $\frac{\boldsymbol{h-2}}{\boldsymbol{2}}$.

Siguiendo la misma lógica, llegamos entonces a la misma conclusión

**Las raíces de ambos hermanos deben ser rojas, independiente de como sean sus hijos**

[**1.5pt**] Por explicar ambos casos en que ambos hijos son de la misma altura.

[**1.5pt**] Por explicar ambos casos en que ambos hijos son de distinta altura

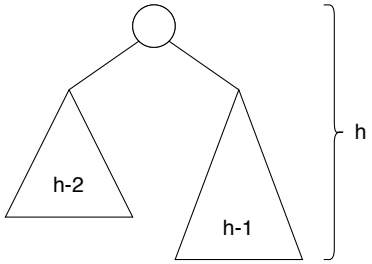
[**No es necesario que la explicación tenga este nivel de detalle**]

## Solución Pregunta 4b)

### Árbol AVL

La propiedad fundamental de un AVL nos dice que, para dos árboles hermanos, sus alturas no difieren en más de 1.

Definimos en clases el AVL lo más desbalanceado posible como el árbol en el que **cada** par de árboles hermanos difiere en 1 de altura. En particular, un árbol de altura $\boldsymbol{h}$ se ve como sigue:




La hoja menos profunda se encuentra en el sub-arbol de menor altura.

Podemos expresar la profundidad $\boldsymbol{p}$ a la que se encuentra la hoja menos profunda en funcion de $\boldsymbol{h}$ como sigue:

$$\boldsymbol{p(h) = 1 + p(h-2)}$$

Ya que la hoja menos profunda está en el sub-árbol de menor altura, y este sub-arbol está un nivel más abajo que el árbol actual. Si expandimos esto nos damos cuenta que:

$$\begin{aligned} \boldsymbol{p(h)} &= \boldsymbol{1 + p(h-2)} \\ &= \boldsymbol{2 + p(h-4)} \\ &= \boldsymbol{3 + p(h-6)} \\ &\boldsymbol{\cdots} \\ &= \boldsymbol{k + p(h-2k)} \end{aligned}$$

Donde basta con ver un AVL de altura 1 y altura 2 para ver que $\boldsymbol{p(1) = 1}$ y $\boldsymbol{p(2) = 2}$.

Por lo tanto, en el caso que $\boldsymbol{h}$ sea impar, la recursión termina con $\boldsymbol{p(1)}$, y por lo tanto

$$\begin{aligned} \boldsymbol{h - 2k} &= \boldsymbol{1} \\ \boldsymbol{2k} &= \boldsymbol{h - 1} \\ \boldsymbol{k} &= \frac{\boldsymbol{h}}{\boldsymbol{2}} - \frac{\boldsymbol{1}}{\boldsymbol{2}} \end{aligned}$$

Reemplazando en $\boldsymbol{p(h)}$:

$$\begin{aligned} \boldsymbol{p(h)} &= \boldsymbol{k + p(h - 2k)} \\ &= \frac{\boldsymbol{h}}{\boldsymbol{2}} - \frac{\boldsymbol{1}}{\boldsymbol{2}} + \boldsymbol{p(1)} \\ &= \frac{\boldsymbol{h}}{\boldsymbol{2}} - \frac{\boldsymbol{1}}{\boldsymbol{2}} + \boldsymbol{1} \\ \boldsymbol{p(h)} &= \frac{\boldsymbol{h}}{\boldsymbol{2}} + \frac{\boldsymbol{1}}{\boldsymbol{2}} \end{aligned}$$

Para el caso de $\boldsymbol{h}$ par la recursión termina con $\boldsymbol{p(2)}$, por lo que:

$$\begin{aligned} \boldsymbol{h - 2k} &= \boldsymbol{2} \\ \boldsymbol{2k} &= \boldsymbol{h - 2} \\ \boldsymbol{k} &= \frac{\boldsymbol{h}}{\boldsymbol{2}} - \boldsymbol{1} \end{aligned}$$

Reemplazando en $\boldsymbol{p(h)}$:

$$\begin{aligned} \boldsymbol{p(h)} &= \boldsymbol{k + p(h - 2k)} \\ &= \frac{\boldsymbol{h}}{\boldsymbol{2}} - \boldsymbol{1} + \boldsymbol{p(1)} \\ &= \frac{\boldsymbol{h}}{\boldsymbol{2}} - \boldsymbol{1} + \boldsymbol{2} \\ \boldsymbol{p(h)} &= \frac{\boldsymbol{h}}{\boldsymbol{2}} + \boldsymbol{1} \end{aligned}$$

**Árbol Rojo-Negro**

La propiedad de balance de un árbol rojo-negro está dictada por que la cantidad de nodos negros camino a una hoja debe ser la misma para todas las hojas.

En este caso, un árbol Rojo-Negro lo más desbalanceado posible es un árbol que tiene hojas cuya ruta no tiene nodos rojos, mientras que además tiene hojas cuya ruta tiene la máxima cantidad posible de nodos rojos.

Sea $\boldsymbol{n}$ la cantidad de nodos negros para cualquier rama, y $\boldsymbol{r}$ la cantidad de nodos rojos de la rama más larga. Por definición, la hoja menos profunda está a profundidad $\boldsymbol{n}$.

$$\boldsymbol{h = n + r}$$

Sabemos que un nodo rojo no puede tener hijos rojos, por lo que $\boldsymbol{r \leq n}$.

El máximo valor de $\boldsymbol{r}$ es $\boldsymbol{n}$, el cual solo puede suceder cuando $\boldsymbol{h}$ es par.

Por lo tanto,

$$\boldsymbol{h = 2n}$$
$$\boldsymbol{n(h) = \frac{h}{2}}$$

En el caso de $\boldsymbol{h}$ impar el máximo valor que puede tomar $\boldsymbol{r}$ es $\boldsymbol{r = n - 1}$.

Por lo tanto, $\boldsymbol{h = 2n - 1}$

$$\boldsymbol{h = 2n - 1}$$
$$\boldsymbol{2n = h + 1}$$
$$\boldsymbol{n(h) = \frac{h}{2} + \frac{1}{2}}$$

Podemos ver que para $\boldsymbol{h}$ par, $\boldsymbol{p(h) > n(h)}$, y que para $\boldsymbol{h}$ impar, $\boldsymbol{p(h) = n(h)}$

Por lo tanto, $\boldsymbol{p(h) \geq n(h)}$, es decir, la profundidad de la hoja menos profunda en un AVL es mayor o igual a la profundidad de hoja menos profunda en un árbol Rojo-Negro.

---

[**1.25pt**] Por el cálculo de la hoja menos profunda en el árbol AVL.

(Sólo [**1pt**] si calcula uno solo de los dos casos, solo [**0.5**] si no calcula)

[**1.25pt**] Por el cálculo de la hoja menos profunda en el árbol Rojo-Negro

(Sólo [**1pt**] si calcula uno solo de los dos casos, solo [**0.85**] si no calcula. No hay descuento por no calcular si cita la solución ayudantía.)

[**0.5pt**] Por la comparación entre ambos.