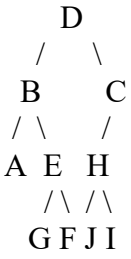
Juan Menduiña

Ejercitación Teórica N° 1:
Árboles Binarios, de Expresión y Generales.

Ejercicio 1.

Dado un árbol binario T cuyo recorrido postorden es A G F E B J I H C D y su recorrido inorden es A B G E F D J H I C. ¿Cuántos son los descendientes del nodo "C"?

(a) *2.* **(b)** *1.* **(c)** *3.* **(d)** *Ninguna de las anteriores.*



Ejercicio 2.

Definir árbol binario completo y árbol binario lleno. Ejemplificar. ¿Es verdad que todo árbol binario completo es lleno? ¿Y viceversa?

Árbol binario lleno: Dado un árbol binario T de altura h, se dice que T es lleno si cada nodo interno tiene grado 2 y todas las hojas están en el mismo nivel.

Árbol binario completo: Dado un árbol binario T de altura h, se dice que T es completo si es lleno de altura h-1 y el nivel h se completa de izquierda a derecha.

No, no es verdad que todo árbol binario completo es lleno, pero sí es cierto que todo árbol binario lleno es completo.

Ejercicio 3.

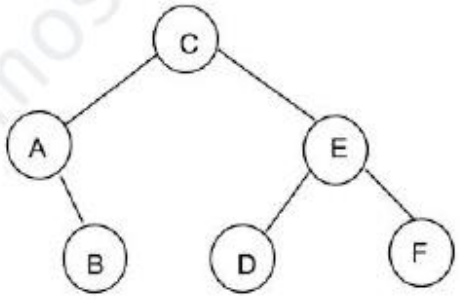
Suponer que, para un árbol binario T con N nodos (N > 1), el último nodo en postorden es el mismo que el último nodo en inorden, ¿qué se puede concluir?

(a) *El subárbol izquierdo de T es vacío.*
(b) *El subárbol derecho de T es vacío.*
(c) *Ningún nodo en el árbol tiene dos hijos.*
(d) *Hay a lo sumo 3 nodos en el árbol.*

Ejercicio 4.

Se han estudiado los distintos recorridos de un árbol binario. Abajo se muestra un código que combina dos de ellos. ¿Cuál es el resultado si se llama con la raíz del árbol de la figura?

```
public void traverse(ArbolBinario<T> a) {
      if (!a.esVacio()) {
            System.out.print(a.getDato());
            if (a.tieneHijoIzquierdo())
               traverse(a.getHijoIzquierdo());
            if (a.tieneHijoDerecho())
               traverse(a.getHijoDerecho());
            System.out.print(a.getDato());
      }
}
```



El resultado, si se llama con la raíz del árbol de la figura, es C A B B A E D D F F E C.

Ejercicio 5.

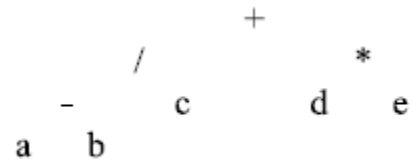
*Evaluar la siguiente expresión postfija y determinar cuál es el resultado: 6 5 * 7 3 - 4 8 + * +.*

(a) *78.* **(b)** *66.* **(c)** *34.* **(d)** *44.*

(6 * 5) + (7 - 3) * (4 + 8)= 30 + 4 * 12
(6 * 5) + (7 - 3) * (4 + 8)= 30 + 48
(6 * 5) + (7 - 3) * (4 + 8)= 78.

Ejercicio 6.

Elegir la expresión algebraica almacenada en el siguiente árbol:



(a) *((a - b / c) + d * e).*
(b) *(((a - b) / (c + d)) + d * e).*
(c) *((a - b / c) + (d * e)).*
(d) *(((a - b) / c) + (d * e)).*

Ejercicio 7.

¿Cuál es el número mínimo de nodos de un árbol binario completo de altura 4?

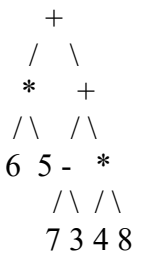
(a) *10.* **(b)** *15.* **(c)** *12.* **(d)** *31.* **(e)** *16.*

h= 4.

$2^h = 2^4$
$2^h = 16$.

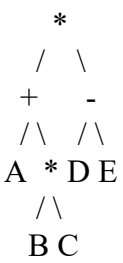
Ejercicio 8.

*Construir el árbol de expresión correspondiente a la siguiente expresión postfija: 6 5 * 7 3 - 4 8 * + +.*



Ejercicio 9.

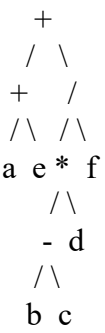
*Construir el árbol de expresión correspondiente a la siguiente expresión infija: (A + (B * C)) * (D - E).*



Ejercicio 10.

*Construir el árbol de expresión correspondiente a la siguiente expresión prefija: ++ a e / * - b c d f. ¿Cuál es la profundidad del nodo d?*

(a) *1.* **(b)** *2.* **(c)** *3.* **(d)** *4.*



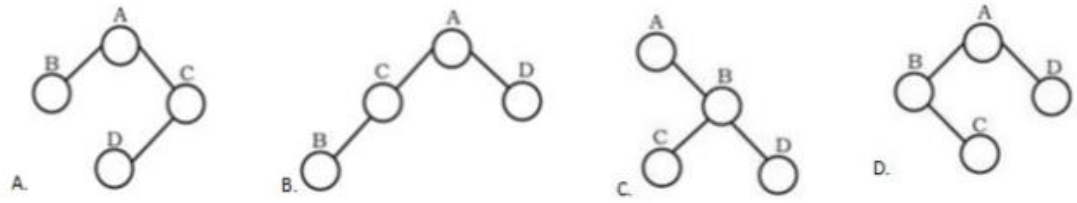
Ejercicio 11.

*Obtener la expresión prefija de la siguiente expresión postfija: A B C * D - E F / G / - *.*

Expresión prefija: * A - - * B C D / / E F G.

Ejercicio 12.

¿Cuál de los siguientes árboles binarios tiene un recorrido inorden BCAD y preorden ABCD?

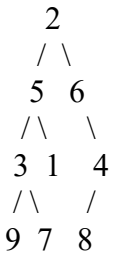


(a) inorden BADC - preorden ABCD.
(b) inorden BCAD - preorden ACBD.
(c) inorden ACBD - preorden ABCD.
(d) inorden BCAD - preorden ABCD.

De estos árboles binarios, el que tiene un recorrido inorden BCAD y preorden ABCD es el (d).

Ejercicio 13.

Reconstruir el árbol binario T cuyo recorrido preorden es 2 5 3 9 7 1 6 4 8 y su recorrido inorden es 9 3 7 5 1 2 6 8 4.



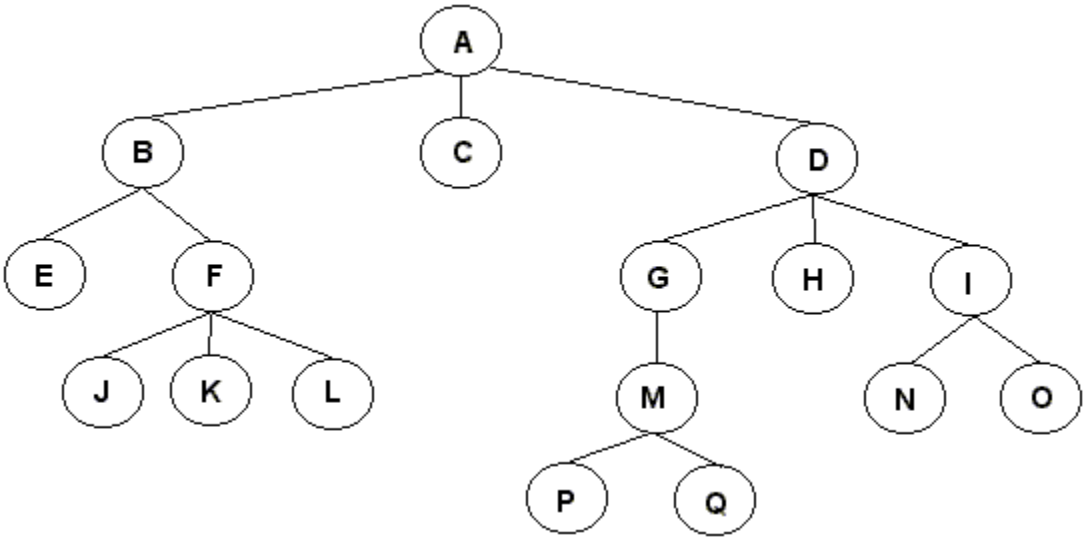
Ejercicio 14.

En un árbol binario lleno, si hay L hojas, entonces, el número total de nodos N es:

(a) *N= 2L.* **(b)** *N= L + 1.* **(c)** *N= L - 1.* **(d)** *N= 2L - 1.*

Ejercicio 15.

La siguiente figura muestra un árbol general:



(a) *Completar los blancos de las sentencias con la terminología vista en clase.*

(i) ***A*** *es la raíz del árbol.*
(ii) ***A*** *es padre de B, C y D.*
(iii) ***E*** *y* ***F*** *son hermanos, puesto que ambos son hijos de B.*
(iv) ***E***, ***J***, ***K***, ***L***, ***C***, ***P***, ***Q***, ***H***, ***N*** *y* ***O*** *son las hojas del árbol.*
(v) *El camino desde A a J es único, lo conforman los nodos* ***A***, ***B***, ***F***, ***J*** *y es de largo* ***3***.
(vi) ***M*** *es ancestro de P y, por lo tanto,* ***M*** *es descendiente de D.*
(vii) *L no es descendiente de C, puesto que no existe* ***un camino*** *desde C a L.*
(viii) *La profundidad/nivel de C es* ***1***, *de F es* ***2*** *y de* ***P*** *y* ***Q*** *es 4.*
(ix) *La altura de C es 0, de* ***F***, ***M*** *e* ***I*** *es 1 y de D es* ***3***.
(x) *La altura del árbol es 4 (largo del camino entre la* ***raíz*** *y* ***la hoja más lejana***).

(b) *Aplicar los recorridos en profundidad (preorden, inorden, postorden) y por niveles.*

Preorden: ABEFJKLCDGMPQHINO.
Inorden: EBJFKLACPMQGDHNIO.
Postorden: EJKLFBCPQMGHNOIDA.
Por niveles: ABCDEFGHIJKLMNOPQ.

Ejercicio 16.

¿Cuál es el número mínimo y máximo de nodos de un árbol general completo de altura h y grado k?

El número mínimo y máximo de nodos de un árbol general completo de altura h y grado k es $\frac{k^h+k-2}{k-1}$ y $\frac{k^{h+1}-1}{k-1}$, respectivamente.

Ejercicio 17.

El recorrido inorden en un árbol general visita:

(a) *Primero la mitad de los subárboles hijos, luego la raíz y luego los restantes subárboles hijos.*
(b) *Primero la raíz y luego los subárboles hijos.*
(c) *Primero los subárboles hijos y luego la raíz.*
(d) *Primero el subárbol hijo más izquierdo, luego la raíz y luego los restantes subárboles hijos.*

Ejercicio 18.

En un árbol general, la profundidad de un nodo n1 es:

(a) *La longitud del único camino que existe entre la raíz y el nodo n1.*
(b) *La longitud del camino más largo que existe entre el nodo n1 y una hoja.*
(c) *La cantidad de nodos hijos del nodo n1.*
(d) *Ninguna de las otras opciones.*

Ejercicio 19.

Un árbol general lleno de grado 4 tiene 21 nodos.

(a) *¿Cuál es la altura del árbol?*

La altura del árbol es 2.

(b) *Desarrollar el proceso realizado para obtener la respuesta anterior.*

k= 4.

$\frac{k^{h+1}-1}{k-1}$= 21
$\frac{4^{h+1}-1}{4-1}$= 21
$\frac{4^{h+1}-1}{3}$= 21
4^{h+1} - 1= 21 * 3
4^{h+1} - 1= 63
4^{h+1}= 63 + 1
4^{h+1}= 64
$log_4\ 4^{h+1} = log_4\ 64$
(h + 1) log_4 4= 3
(h + 1) * 1= 3
h + 1= 3
h= 3 - 1
h= 2.

Ejercicio 20.

¿Cuál es la cantidad mínima de nodos en un árbol general completo de grado 3 y altura 4?

(a) *40.* **(b)** *41.* **(c)** *121.* **(d)** *122.*

k= 3; h= 4.

$$\frac{k^h+k-2}{k-1} = \frac{3^4+3-2}{3-1}$$
$$\frac{k^h+k-2}{k-1} = \frac{81+3-2}{2}$$
$$\frac{k^h+k-2}{k-1} = \frac{82}{2}$$
$$\frac{k^h+k-2}{k-1} = 41.$$

Ejercicio 21.

Un árbol general lleno de grado 5 tiene 125 hojas.

(a) *¿Cuál es la cantidad de nodos internos del árbol?*

La cantidad de nodos internos del árbol es 31.

(b) *Desarrollar el proceso realizado para obtener la respuesta anterior.*

k= 5.

k^h= 125
5^h= 125
$log_5\ 5^h = log_5\ 125$
h log_5 5= 3
h * 1= 3
h= 3.

Nodos internos= $\frac{k^{h+1}-1}{k-1}$ - k^h
Nodos internos= $\frac{5^{3+1}-1}{5-1}$ - 5^3
Nodos internos= $\frac{5^4-1}{4}$ - 125
Nodos internos= $\frac{625-1}{4}$ - 125
Nodos internos= $\frac{624}{4}$ - 125
Nodos internos= 156 - 125
Nodos internos= 31.

Ejercicio 22.

¿Cuál es la cantidad de nodos en un árbol general completo de grado 4 y altura 3?

(a) *entre 16 y 21.* **(b)** *entre 22 y 85.* **(c)** *entre 22 y 64.* **(d)** *entre 16 y 64.*

k= 4; h= 3.

$$\frac{k^h+k-2}{k-1} = \frac{4^3+4-2}{4-1}$$
$$\frac{k^h+k-2}{k-1} = \frac{64+4-2}{3}$$
$$\frac{k^h+k-2}{k-1} = \frac{66}{3}$$
$$\frac{k^h+k-2}{k-1} = 22.$$

$$\frac{k^{h+1}-1}{k-1} = \frac{4^{3+1}-1}{4-1}$$
$$\frac{k^{h+1}-1}{k-1} = \frac{4^4-1}{3}$$
$$\frac{k^{h+1}-1}{k-1} = \frac{256-1}{3}$$
$$\frac{k^{h+1}-1}{k-1} = \frac{255}{3}$$
$$\frac{k^{h+1}-1}{k-1} = 85.$$

Juan Menduiña

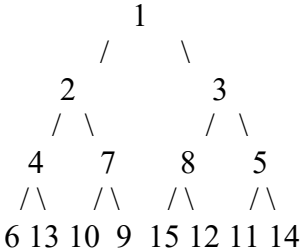
Ejercitación Teórica N° 2:
Cola de Prioridades - Heap.

Ejercicio 1.

A partir de una heap inicialmente vacía, insertar, de a uno, los siguientes valores: 6, 4, 15, 2, 10, 11, 8, 1, 13, 7, 9, 12, 5, 3, 14.

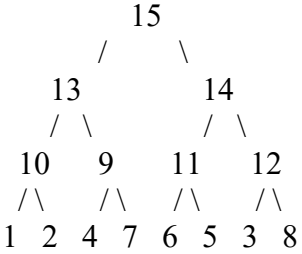
[6] - *Insert(6)*
[6, 4] - *Insert(4)*
[4, 6]
[4, 6, 15] - *Insert(15)*
[4, 6, 15, 2] - *Insert(2)*
[4, 2, 15, 6]
[2, 4, 15, 6]
[2, 4, 15, 6, 10] - *Insert(10)*
[2, 4, 15, 6, 10, 11] - *Insert(11)*
[2, 4, 11, 6, 10, 15]
[2, 4, 11, 6, 10, 15, 8] - *Insert(8)*
[2, 4, 8, 6, 10, 15, 11]
[2, 4, 8, 6, 10, 15, 11, 1] - *Insert(1)*
[2, 4, 8, 1, 10, 15, 11, 6]
[2, 1, 8, 4, 10, 15, 11, 6]
[1, 2, 8, 4, 10, 15, 11, 6]
[1, 2, 8, 4, 10, 15, 11, 6, 13] - *Insert(13)*
[1, 2, 8, 4, 10, 15, 11, 6, 13, 7] - *Insert(7)*
[1, 2, 8, 4, 7, 15, 11, 6, 13, 10]
[1, 2, 8, 4, 7, 15, 11, 6, 13, 10, 9] - *Insert(9)*
[1, 2, 8, 4, 7, 15, 11, 6, 13, 10, 9, 12] - *Insert(12)*
[1, 2, 8, 4, 7, 12, 11, 6, 13, 10, 9, 15]
[1, 2, 8, 4, 7, 12, 11, 6, 13, 10, 9, 15, 5] - *Insert(5)*
[1, 2, 8, 4, 7, 5, 11, 6, 13, 10, 9, 15, 12]
[1, 2, 5, 4, 7, 8, 11, 6, 13, 10, 9, 15, 12]
[1, 2, 5, 4, 7, 8, 11, 6, 13, 10, 9, 15, 12, 3] - *Insert(3)*
[1, 2, 5, 4, 7, 8, 3, 6, 13, 10, 9, 15, 12, 11]
[1, 2, 3, 4, 7, 8, 5, 6, 13, 10, 9, 15, 12, 11]
[1, 2, 3, 4, 7, 8, 5, 6, 13, 10, 9, 15, 12, 11, 14] - *Insert(14)*

Min-Heap: [1, 2, 3, 4, 7, 8, 5, 6, 13, 10, 9, 15, 12, 11, 14].



[6] - *Insert(6)*
[6, 4] - *Insert(4)*
[6, 4, 15] - *Insert(15)*
[15, 4, 6]
[15, 4, 6, 2] - *Insert(2)*
[15, 4, 6, 2, 10] - *Insert(10)*
[15, 10, 6, 2, 4]
[15, 10, 6, 2, 4, 11] - *Insert(11)*
[15, 10, 11, 2, 4, 6]
[15, 10, 11, 2, 4, 6, 8] - *Insert(8)*
[15, 10, 11, 2, 4, 6, 8, 1] - *Insert(1)*
[15, 10, 11, 2, 4, 6, 8, 1, 13] - *Insert(13)*
[15, 10, 11, 13, 4, 6, 8, 1, 2]
[15, 13, 11, 10, 4, 6, 8, 1, 2]
[15, 13, 11, 10, 4, 6, 8, 1, 2, 7] - *Insert(7)*
[15, 13, 11, 10, 7, 6, 8, 1, 2, 4]
[15, 13, 11, 10, 7, 6, 8, 1, 2, 4, 9] - *Insert(9)*
[15, 13, 11, 10, 9, 6, 8, 1, 2, 4, 7]
[15, 13, 11, 10, 9, 6, 8, 1, 2, 4, 7, 12] - *Insert(12)*
[15, 13, 11, 10, 9, 12, 8, 1, 2, 4, 7, 6]
[15, 13, 12, 10, 9, 11, 8, 1, 2, 4, 7, 6]
[15, 13, 12, 10, 9, 11, 8, 1, 2, 4, 7, 6, 5] - *Insert(5)*
[15, 13, 12, 10, 9, 11, 8, 1, 2, 4, 7, 6, 5, 3] - *Insert(3)*
[15, 13, 12, 10, 9, 11, 8, 1, 2, 4, 7, 6, 5, 3, 14] - *Insert(14)*
[15, 13, 12, 10, 9, 11, 14, 1, 2, 4, 7, 6, 5, 3, 8]
[15, 13, 14, 10, 9, 11, 12, 1, 2, 4, 7, 6, 5, 3, 8].

Max-Heap: [15, 13, 14, 10, 9, 11, 12, 1, 2, 4, 7, 6, 5, 3, 8].



6, 4, 15, 2, 10, 11, 8, 1, 13, 7, 9, 12, 5, 3, 14

La raíz está almacenada en la posición 1. Para un elemento que está en la posición i, se tiene:

- El hijo izquierdo está en la posición 2i.
- El hijo derecho está en la posición 2i + 1.
- El padre está en la posición $\left\lfloor \frac{i}{2} \right\rfloor$.

Ejercicio 2.

(a) *¿Cuántos elementos hay, al menos, en una heap de altura h?*

En una *heap* de altura h, hay, al menos, 2^h elementos.

(b) *¿Dónde se encuentra ubicado el elemento mínimo en una max-heap?*

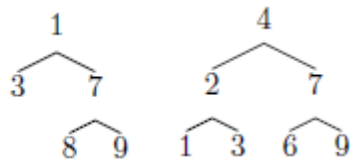
El elemento mínimo en una *max-heap* se encuentra ubicado en alguna de las hojas.

(c) *¿El siguiente arreglo es una max-heap: [23, 17, 14, 6, 13, 10, 1, 5, 7, 12]?*

No, este arreglo no es una *max-heap*, ya que el nodo con valor 6 es padre del nodo con valor 7, no cumpliendo con que cada nodo sea mayor o igual que sus hijos.

Ejercicio 3.

Dados los siguientes árboles, indicar si representan una heap. Justificar la respuesta.



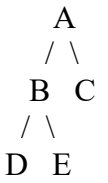
El primer árbol no es una *heap*, ya que no es un árbol binario completo.

El segundo árbol no es una *heap*, ya que, si bien es completo, cada nodo no es menor o igual (o mayor o igual) que sus hijos.

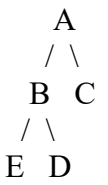
Ejercicio 4.

Dibujar todas las min-heaps posibles para este conjunto de claves: {A, B, C, D, E}.

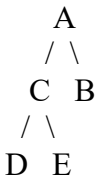
Opción 1:



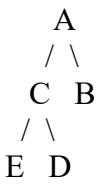
Opción 2:



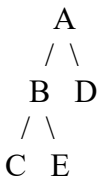
Opción 3:



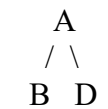
Opción 4:



Opción 5:



Opción 6:



Opción 7:

Opción 8:

Ejercicio 5.

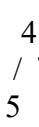
A partir de una min-heap inicialmente vacía, dibujar la evolución del estado de la heap al ejecutar las siguientes operaciones:

Insert(5), Insert(4), Insert(7), Insert(1), DeleteMin(), Insert(3), Insert(6), DeleteMin(), DeleteMin(), Insert(8), DeleteMin(), Insert(2), DeleteMin(), DeleteMin().

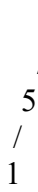
Insert(5):

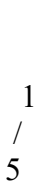
Insert(4):

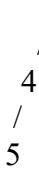
Insert(7):



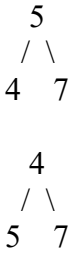
Insert(1):



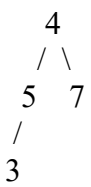


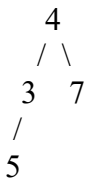


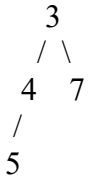
DeleteMin():



Insert(3):

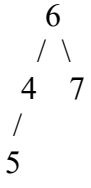


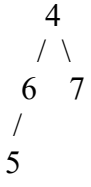


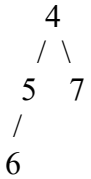


Insert(6):

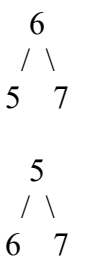
DeleteMin():



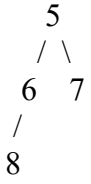




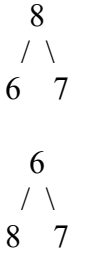
DeleteMin():



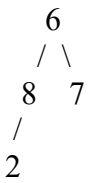
Insert(8):

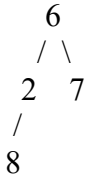


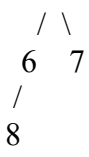
DeleteMin():



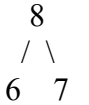
Insert(2):

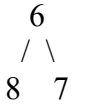






DeleteMin():





DeleteMin():

Ejercicio 6.

Aplicar el algoritmo BuildHeap para construir una min-heap en tiempo lineal, con los siguientes valores: {150, 80, 40, 10, 70, 110, 30, 120, 140, 60, 50, 130, 100, 20, 90}.

Se parte de:

[150, 80, 40, 10, 70, 110, 30, 120, 140, 60, 50, 130, 100, 20, 90].

Se empieza en la posición i= $\left\lfloor \frac{n}{2} \right\rfloor$= 7 del arreglo, el 30, y se filtra:

[150, 80, 40, 10, 70, 110, 20, 120, 140, 60, 50, 130, 100, 30, 90].

Se avanza a la posición anterior del arreglo, el 110, y se filtra:

[150, 80, 40, 10, 70, 100, 20, 120, 140, 60, 50, 130, 110, 30, 90].

Se avanza a la posición anterior del arreglo, el 70, y se filtra:

[150, 80, 40, 10, 50, 100, 20, 120, 140, 60, 70, 130, 110, 30, 90].

Se avanza a la posición anterior del arreglo, el 10, y se filtra:

[150, 80, 40, 10, 50, 100, 20, 120, 140, 60, 70, 130, 110, 30, 90].

Se avanza a la posición anterior del arreglo, el 40, y se filtra:

[150, 80, 20, 10, 50, 100, 40, 120, 140, 60, 70, 130, 110, 30, 90]
[150, 80, 20, 10, 50, 100, 30, 120, 140, 60, 70, 130, 110, 40, 90].

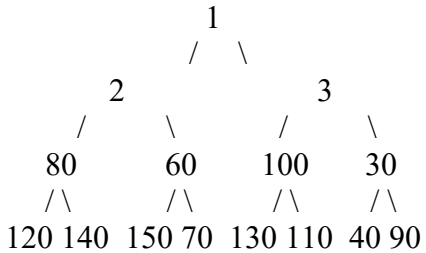
Se avanza a la posición anterior del arreglo, el 80, y se filtra:

[150, 10, 20, 80, 50, 100, 30, 120, 140, 60, 70, 130, 110, 40, 90].

Se avanza a la posición anterior del arreglo, el 150, y se filtra:

[10, 150, 20, 80, 50, 100, 30, 120, 140, 60, 70, 130, 110, 40, 90]
[10, 50, 20, 80, 150, 100, 30, 120, 140, 60, 70, 130, 110, 40, 90]
[10, 50, 20, 80, 60, 100, 30, 120, 140, 150, 70, 130, 110, 40, 90].

Min-Heap: [10, 50, 20, 80, 60, 100, 30, 120, 140, 150, 70, 130, 110, 40, 90].



Ejercicio 7.

Aplicar el algoritmo HeapSort para ordenar, descendentemente, los siguientes elementos: {15, 18, 40, 1, 7, 10, 33, 2, 140, 500, 11, 12, 13, 90}. Mostrar, paso a paso, la ejecución del algoritmo sobre los datos.

Se parte de:

Min-Heap: [1, 2, 10, 7, 11, 12, 33, 18, 140, 500, 15, 40, 13, 90].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 90:

[90, 2, 10, 7, 11, 12, 33, 18, 140, 500, 15, 40, 13 | 1]
[2, 90, 10, 7, 11, 12, 33, 18, 140, 500, 15, 40, 13 | 1]
[2, 7, 10, 90, 11, 12, 33, 18, 140, 500, 15, 40, 13 | 1]
[2, 7, 10, 18, 11, 12, 33, 90, 140, 500, 15, 40, 13 | 1].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 13:

[13, 7, 10, 18, 11, 12, 33, 90, 140, 500, 15, 40 | 2, 1]
[7, 13, 10, 18, 11, 12, 33, 90, 140, 500, 15, 40 | 2, 1]
[7, 11, 10, 18, 13, 12, 33, 90, 140, 500, 15, 40 | 2, 1].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 40:

[40, 11, 10, 18, 13, 12, 33, 90, 140, 500, 15 | 7, 2, 1]
[10, 11, 40, 18, 13, 12, 33, 90, 140, 500, 15 | 7, 2, 1]
[10, 11, 12, 18, 13, 40, 33, 90, 140, 500, 15 | 7, 2, 1].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 15:

[15, 11, 12, 18, 13, 40, 33, 90, 140, 500 | 10, 7, 2, 1]
[11, 15, 12, 18, 13, 40, 33, 90, 140, 500 | 10, 7, 2, 1]
[11, 13, 12, 18, 15, 40, 33, 90, 140, 500 | 10, 7, 2, 1].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 500:

[500, 13, 12, 18, 15, 40, 33, 90, 140 | 11, 10, 7, 2, 1]
[12, 13, 500, 18, 15, 40, 33, 90, 140 | 11, 10, 7, 2, 1]
[12, 13, 33, 18, 15, 40, 500, 90, 140 | 11, 10, 7, 2, 1].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 140:

[140, 13, 33, 18, 15, 40, 500, 90 | 12, 11, 10, 7, 2, 1]
[13, 140, 33, 18, 15, 40, 500, 90 | 12, 11, 10, 7, 2, 1]
[13, 15, 33, 18, 140, 40, 500, 90 | 12, 11, 10, 7, 2, 1].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 90:

[90, 15, 33, 18, 140, 40, 500 | 13, 12, 11, 10, 7, 2, 1]
[15, 90, 33, 18, 140, 40, 500 | 13, 12, 11, 10, 7, 2, 1]
[15, 18, 33, 90, 140, 40, 500 | 13, 12, 11, 10, 7, 2, 1].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 500:

[500, 18, 33, 90, 140, 40 | 15, 13, 12, 11, 10, 7, 2, 1]
[18, 500, 33, 90, 140, 40 | 15, 13, 12, 11, 10, 7, 2, 1]
[18, 90, 33, 500, 140, 40 | 15, 13, 12, 11, 10, 7, 2, 1].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 40:

[40, 90, 33, 500, 140 | 18, 15, 13, 12, 11, 10, 7, 2, 1]
[33, 90, 40, 500, 140 | 18, 15, 13, 12, 11, 10, 7, 2, 1].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 140:

[140, 90, 40, 500 | 33, 18, 15, 13, 12, 11, 10, 7, 2, 1]
[40, 90, 140, 500 | 33, 18, 15, 13, 12, 11, 10, 7, 2, 1].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 500:

[500, 90, 140 | 40, 33, 18, 15, 13, 12, 11, 10, 7, 2, 1]
[90, 500, 140 | 40, 33, 18, 15, 13, 12, 11, 10, 7, 2, 1].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 140:

[140, 500 | 90, 40, 33, 18, 15, 13, 12, 11, 10, 7, 2, 1].

Intercambiar el primero con el último, decrementar el tamaño:

[500 | 140, 90, 40, 33, 18, 15, 13, 12, 11, 10, 7, 2, 1].

Arreglo ordenado descendentemente:

[500, 140, 90, 40, 33, 18, 15, 13, 12, 11, 10, 7, 2, 1].

Ejercicio 8.

Construir una max-heap con los siguientes datos: {5, 8, 12, 9, 7, 10, 21, 6, 14, 4}.

(a) *Insertándolos de a uno.*

[5] - *Insert(5)*
[5, 8] - *Insert(8)*
[8, 5]
[8, 5, 12] - *Insert(12)*
[12, 5, 8]
[12, 5, 8, 9] - *Insert(9)*
[12, 9, 8, 5]
[12, 9, 8, 5, 7] - *Insert(7)*
[12, 9, 8, 5, 7, 10] - *Insert(10)*
[12, 9, 10, 5, 7, 8]
[12, 9, 10, 5, 7, 8, 21] - *Insert(21)*
[12, 9, 21, 5, 7, 8, 10]
[21, 9, 12, 5, 7, 8, 10]
[21, 9, 12, 5, 7, 8, 10, 6] - *Insert(6)*
[21, 9, 12, 6, 7, 8, 10, 5]
[21, 9, 12, 6, 7, 8, 10, 5, 14] - *Insert(14)*
[21, 9, 12, 14, 7, 8, 10, 5, 6]
[21, 14, 12, 9, 7, 8, 10, 5, 6]
[21, 14, 12, 9, 7, 8, 10, 5, 6, 4] - *Insert(4)*

Max-Heap: [21, 14, 12, 9, 7, 8, 10, 5, 6, 4].

(b) *Usando el algoritmo BuildHeap.*

Se parte de:

[5, 8, 12, 9, 7, 10, 21, 6, 14, 4].

Se empieza en la posición i= $\left\lfloor \frac{n}{2} \right\rfloor$= 5 del arreglo, el 7, y se filtra:

[5, 8, 12, 9, 7, 10, 21, 6, 14, 4].

Se avanza a la posición anterior del arreglo, el 9, y se filtra:

[5, 8, 12, 14, 7, 10, 21, 6, 9, 4].

Se avanza a la posición anterior del arreglo, el 12, y se filtra:

[5, 8, 21, 14, 7, 10, 12, 6, 9, 4].

Se avanza a la posición anterior del arreglo, el 8, y se filtra:

[5, 14, 21, 8, 7, 10, 12, 6, 9, 4]
[5, 14, 21, 9, 7, 10, 12, 6, 8, 4].

Se avanza a la posición anterior del arreglo, el 5, y se filtra:

[21, 14, 5, 9, 7, 10, 12, 6, 8, 4]
[21, 14, 12, 9, 7, 10, 5, 6, 8, 4].

Max-Heap: [21, 14, 12, 9, 7, 10, 5, 6, 8, 4].

Ejercicio 9.

Suponer que una heap que representa una cola de prioridades está almacenada en el arreglo A (se comienza de la posición A[1]). Si se inserta la clave 16, ¿en qué posición quedará?

i	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*
A[i]	*11*	*21*	*27*	*37*	*36*	*34*	*32*	*43*	*44*	*42*	*51*	*62*

(a) *A[2].* **(b)** *A[3].* **(c)** *A[6].* **(d)** *A[7].* **(e)** *A[12].*

[11, 21, 27, 37, 36, 34, 32, 43, 44, 42, 51, 62]
[11, 21, 27, 37, 36, 34, 32, 43, 44, 42, 51, 62, 16] - *Insert(16)*
[11, 21, 27, 37, 36, 16, 32, 43, 44, 42, 51, 62, 34]
[11, 21, 16, 37, 36, 27, 32, 43, 44, 42, 51, 62, 34].

Ejercicio 10.

Suponer que una heap que representa una cola de prioridades está almacenada en el arreglo A (se comienza de la posición A[1]). Si se aplica un delete-min, ¿en qué posición quedará la clave 62?

i	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*
A[i]	*11*	*21*	*27*	*37*	*36*	*34*	*32*	*43*	*44*	*42*	*51*	*62*

(a) *A[1].* **(b)** *A[2].* **(c)** *A[10].* **(d)** *A[11].* **(e)** *A[12].*

[11, 21, 27, 37, 36, 34, 32, 43, 44, 42, 51, 62]
[62, 21, 27, 37, 36, 34, 32, 43, 44, 42, 51] - *DeleteMin()*
[21, 62, 27, 37, 36, 34, 32, 43, 44, 42, 51]
[21, 36, 27, 37, 62, 34, 32, 43, 44, 42, 51]
[21, 36, 27, 37, 42, 34, 32, 43, 44, 62, 51].

Ejercicio 11.

(a) *Ordenar, en forma creciente, los datos del ejercicio anterior, usando el algoritmo HeapSort.*

Se parte de:

Max-Heap: [62, 51, 42, 44, 43, 32, 34, 37, 36, 27, 21, 11].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 11:

[11, 51, 42, 44, 43, 32, 34, 37, 36, 27, 21 | 62]
[51, 11, 42, 44, 43, 32, 34, 37, 36, 27, 21 | 62]
[51, 44, 42, 11, 43, 32, 34, 37, 36, 27, 21 | 62]
[51, 44, 42, 37, 43, 32, 34, 11, 36, 27, 21 | 62].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 21:

[21, 44, 42, 37, 43, 32, 34, 11, 36, 27 | 51, 62]
[44, 21, 42, 37, 43, 32, 34, 11, 36, 27 | 51, 62]
[44, 43, 42, 37, 21, 32, 34, 11, 36, 27 | 51, 62]
[44, 43, 42, 37, 27, 32, 34, 11, 36, 21 | 51, 62].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 21:

[21, 43, 42, 37, 27, 32, 34, 11, 36 | 44, 51, 62]
[43, 21, 42, 37, 27, 32, 34, 11, 36 | 44, 51, 62]
[43, 37, 42, 21, 27, 32, 34, 11, 36 | 44, 51, 62]
[43, 37, 42, 36, 27, 32, 34, 11, 21 | 44, 51, 62].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 21:

[21, 37, 42, 36, 27, 32, 34, 11 | 43, 44, 51, 62]
[42, 37, 21, 36, 27, 32, 34, 11 | 43, 44, 51, 62]
[42, 37, 34, 36, 27, 32, 21, 11 | 43, 44, 51, 62].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 11:

[11, 37, 34, 36, 27, 32, 21 | 42, 43, 44, 51, 62]
[37, 11, 34, 36, 27, 32, 21 | 42, 43, 44, 51, 62]
[37, 36, 34, 11, 27, 32, 21 | 42, 43, 44, 51, 62].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 21:

[21, 36, 34, 11, 27, 32 | 37, 42, 43, 44, 51, 62]
[36, 21, 34, 11, 27, 32 | 37, 42, 43, 44, 51, 62]
[36, 27, 34, 11, 21, 32 | 37, 42, 43, 44, 51, 62].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 32:

[32, 27, 34, 11, 21 | 36, 37, 42, 43, 44, 51, 62]
[34, 27, 32, 11, 21 | 36, 37, 42, 43, 44, 51, 62].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 21:

[21, 27, 32, 11 | 34, 36, 37, 42, 43, 44, 51, 62]
[32, 27, 21, 11 | 34, 36, 37, 42, 43, 44, 51, 62].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 11:

[11, 27, 21 | 32, 34, 36, 37, 42, 43, 44, 51, 62]
[27, 11, 21 | 32, 34, 36, 37, 42, 43, 44, 51, 62].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 21:

[21, 11 | 27, 32, 34, 36, 37, 42, 43, 44, 51, 62].

Intercambiar el primero con el último, decrementar el tamaño:

[11 | 21, 27, 32, 34, 36, 37, 42, 43, 44, 51, 62].

Arreglo ordenado ascendentemente:

[11, 21, 27, 32, 34, 36, 37, 42, 43, 44, 51, 62].

(b) *¿Cuáles serían los pasos a seguir si se quiere ordenar en forma decreciente?*

Los pasos a seguir si se quiere ordenar en forma decreciente sería partir de una *Min-Heap*, intercambiar el primer con el último elemento del arreglo, decrementar el tamaño de éste y filtrar correspondientemente en cada caso.

Ejercicio 12.

¿Cuáles de los siguientes arreglos representan una max-heap, min-heap o ninguna de las dos?

(a) *arreglo 1: 0 1 2 0 4 5 6 7 8 9.*

Ninguna de las dos.

(b) *arreglo 2: 9 8 7 6 5 4 3 2 1 0.*

Max-Heap.

(c) *arreglo 3: 5 5 5 6 6 6 6 7 7 1.*

Ninguna de las dos.

(d) *arreglo 4: 9 3 9 2 1 6 7 1 2 1.*

Max-Heap.

(e) *arreglo 5: 8 7 6 1 2 3 4 2 1 2.*

Ninguna de las dos.

Ejercicio 13.

Un arreglo de 7 enteros se ordena ascendentemente usando el algoritmo HeapSort. Luego de la fase inicial del algoritmo (la construcción de la heap), ¿cuál de los siguientes es un posible orden del arreglo?

(a) *85 78 45 51 53 47 49.*
(b) *85 49 78 45 47 51 53.*
(c) *85 78 49 45 47 51 53.*
(d) *45 85 78 53 51 49 47.*
(e) *85 51 78 53 49 47 45.*

Ejercicio 14.

En una Heap, ¿para un elemento que está en la posición i, su hijo derecho está en la posición ...?

(a) $\left\lfloor \frac{i}{2} \right\rfloor$.
(b) *2i.*
(c) *2i + 1.*
(d) *Ninguna de las anteriores.*

Ejercicio 15.

¿Siempre se puede decir que un árbol binario lleno es una Heap?

(a) *Sí.* **(b)** *No.*

Ejercicio 16.

La operación que agrega un elemento a la heap que tiene n elementos, en el peor caso, es de ...

(a) *O(n).* **(b)** *O(n log n).* **(c)** *O(log n).* **(d)** *Ninguna de las anteriores.*

Ejercicio 17.

Se construyó una Max-Heap con las siguientes claves: 13, 21, 87, 30, 25, 22, 18. ¿Cuál de las siguientes opciones corresponde al resultado de realizar la construcción insertando las claves una a una?

(a) *87, 30, 25, 22, 21, 18, 13.*
(b) *87, 30, 22, 21, 25, 13, 18.*
(c) *87, 30, 25, 13, 22, 18, 21.*
(d) *87, 30, 22, 13, 25, 21, 18.*

[13] - *Insert(13)*
[13, 21] - *Insert(21)*
[21, 13]
[21, 13, 87] - *Insert(87)*
[87, 13, 21]
[87, 13, 21, 30] - *Insert(30)*
[87, 30, 21, 13]
[87, 30, 21, 13, 25] - *Insert(25)*
[87, 30, 21, 13, 25, 22] - *Insert(22)*
[87, 30, 22, 13, 25, 21]
[87, 30, 22, 13, 25, 21, 18] - *Insert(18)*

Max-Heap: [87, 30, 22, 13, 25, 21, 18].

Ejercicio 18.

Se construyó una Max-Heap con las siguientes claves: 13, 21, 87, 30, 25, 22, 18. ¿Cuál de las siguientes opciones corresponde al resultado de realizar la construcción aplicando el algoritmo Build-Heap?

(a) *87, 30, 25, 22, 21, 18, 13.*
(b) *87, 30, 22, 21, 25, 13, 18.*
(c) *87, 30, 25, 13, 22, 18, 21.*
(d) *87, 30, 22, 13, 25, 21, 18.*

Se parte de:

[13, 21, 87, 30, 25, 22, 18].

Se empieza en la posición i= $\left\lfloor \frac{n}{2} \right\rfloor$= 3 del arreglo, el 87, y se filtra:

[13, 21, 87, 30, 25, 22, 18].

Se avanza a la posición anterior del arreglo, el 21, y se filtra:

[13, 30, 87, 21, 25, 22, 18].

Se avanza a la posición anterior del arreglo, el 13, y se filtra:

[87, 30, 13, 21, 25, 22, 18]
[87, 30, 22, 21, 25, 13, 18].

Max-Heap: [87, 30, 22, 21, 25, 13, 18].

Ejercicio 19.

El algoritmo HeapSort consta de dos etapas:

(1) Se construye una heap y
(2) Se realizan los intercambios necesarios para dejar ordenados los datos.

Asumir que la heap ya está construida y es la siguiente: 58 38 53 23 28 40 35 18.

¿Cómo quedan los datos en el arreglo después de ejecutar sólo 2 pasos de la segunda etapa del HeapSort?

(a) *40 38 23 28 35 18 53 58.*
(b) *53 38 40 23 28 18 35 58.*
(c) *40 38 23 35 28 18 53 58.*
(d) *40 38 35 23 28 18 53 58.*

Se parte de:

Max-Heap: [58, 38, 53, 23, 28, 40, 35, 18].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 18:

[18, 38, 53, 23, 28, 40, 35 | 58]
[53, 38, 18, 23, 28, 40, 35 | 58]
[53, 38, 40, 23, 28, 18, 35 | 58].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 35:

[35, 38, 40, 23, 28, 18, 35 | 53, 58]
[40, 38, 35, 23, 28, 18 | 53, 58].

Ejercicio 20.

Dada la Min-Heap 3, 8, 5, 15, 10, 7, 19, 28, 16, 25, 12. ¿En qué posición está ubicado el hijo derecho de la clave 15?

(a) *7.*
(b) *8.*
(c) *9.*
(d) *10.*

2i + 1= 2 * 4 + 1
2i + 1= 8 + 1
2i + 1= 9.

Ejercicio 21.

Construir una min-heap con las siguientes claves: 15, 25, 23, 13, 18, 2, 19, 20, 17, insertándose una a una. Indicar en qué posiciones quedaron ubicadas las claves: 2, 18 y 25.

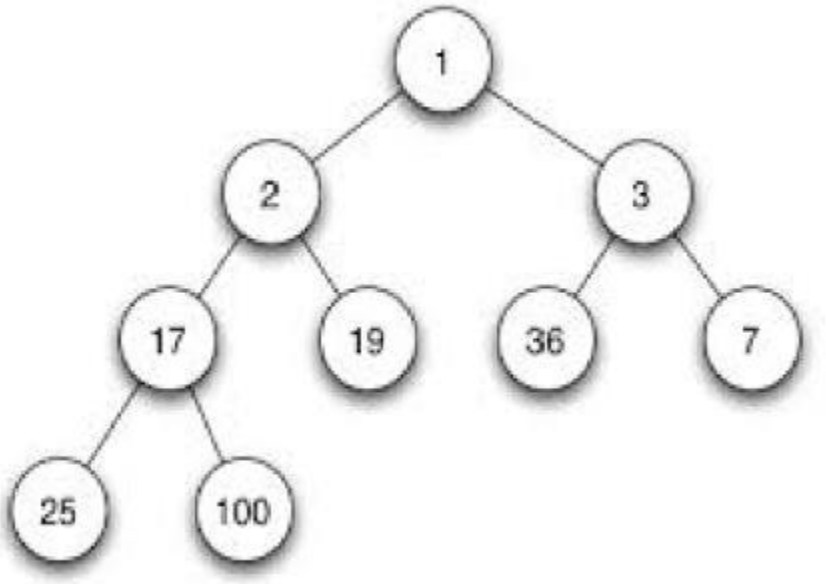
[15] - *Insert(15)*
[15, 25] - *Insert(25)*
[15, 25, 23] - *Insert(23)*
[15, 25, 23, 13] - *Insert(13)*
[13, 15, 23, 25]
[13, 15, 23, 25, 18] - *Insert(18)*
[13, 15, 23, 25, 18, 2] - *Insert(2)*
[13, 15, 2, 25, 18, 23]
[2, 15, 13, 25, 18, 23]
[2, 15, 13, 25, 18, 23, 19] - *Insert(19)*
[2, 15, 13, 25, 18, 23, 19, 20] - *Insert(20)*
[2, 15, 13, 20, 18, 23, 19, 25]
[2, 15, 13, 20, 18, 23, 19, 25, 17] - *Insert(17)*
[2, 15, 13, 17, 18, 23, 19, 25, 20].

Min-Heap: [2, 15, 13, 17, 18, 23, 19, 25, 20].

Las claves 2, 18 y 25 quedaron ubicadas en las posiciones 1, 5 y 8, respectivamente.

Ejercicio 22.

Luego de insertar la clave 15 en la siguiente min-heap, ¿cuántas de las claves que ya estaban en la heap han mantenido su lugar (es decir, ocupan en la min-heap resultante la misma posición que ocupaban antes de la inserción)?

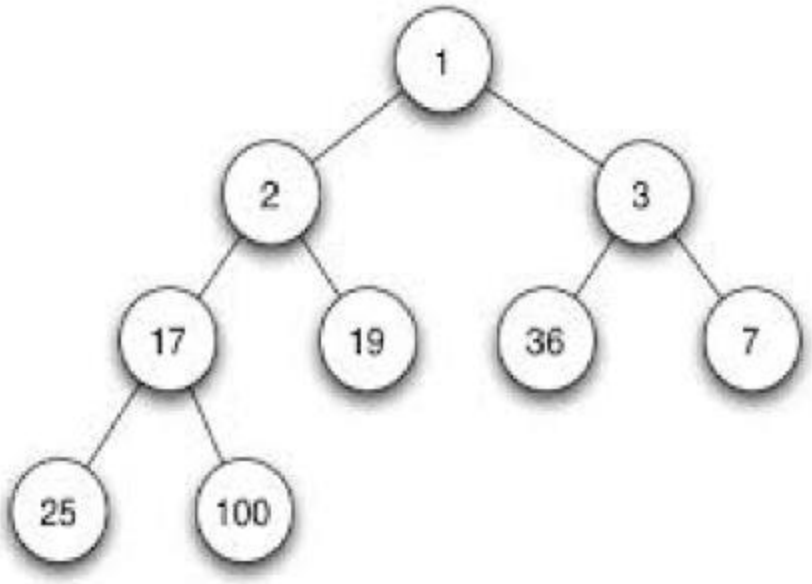


(a) *Ninguna.*
(b) *Seis.*
(c) *Ocho.*
(d) *Nueve.*

[1, 2, 3, 17, 19, 36, 7, 25, 100]
[1, 2, 3, 17, 19, 36, 7, 25, 100, 15] - *Insert(15)*
[1, 2, 3, 17, 15, 36, 7, 25, 100, 19].

Ejercicio 23.

Luego de una operación de borrado del mínimo en la siguiente min-heap, ¿cuántas claves han cambiado de lugar (es decir, ocupan en la min-heap resultante un lugar diferente al que ocupaban en la min-heap antes del borrado) ? (No contar la clave borrada, ya que no pertenece más a la heap).



(a) *Ninguna.*
(b) *Dos.*
(c) *Tres.*
(d) *Cuatro.*

[1, 2, 3, 17, 19, 36, 7, 25, 100]
[100, 2, 3, 17, 19, 36, 7, 25] - *DeleteMin()*
[2, 100, 3, 17, 19, 36, 7, 25]
[2, 17, 3, 100, 19, 36, 7, 25]
[2, 17, 3, 25, 19, 36, 7, 100].

Juan Menduiña

Ejercitación Teórica N° 3:
Análisis de Algoritmos.

Ejercicio 1.

Ordenar las siguientes funciones: $\sqrt{n}$*, n,* 3^n*,* n^2*, cte,* 2^n*,* log_2^2 *n,* log_3 *n,* log_2 *n, según su velocidad de crecimiento.*

cte $< log_3$ n $< log_2$ n $< log_2^2$ n $< \sqrt{n} <$ n $< n^2 < 2^n < 3^n$.

Ejercicio 2.

Expresar de qué orden es el siguiente fragmento de código:

```
for (int j = 4; j < n; j=j+2) {
      val = 0;
      for (int i = 0; i < j; ++i) {
            val = val + i * j;
            for (int k = 0; k < n; ++k){
                  val++;
            }
      }
}
```

(a) $O(n \log n)$. **(b)** $O(n^2)$. **(c)** $O(n^2 \log n)$. **(d)** $O(n^3)$.

Iteraciones del primer *for*:

j= 4.
j= 4 + 2.
j= 4 + 2 * 2.
…
j= 4 + 2 (k - 1).

4 + 2 (k - 1)= n - 1
2 (k - 1)= n - 1 - 4
2 (k - 1)= n - 5
k - 1= $\frac{n-5}{2}$
k= $\frac{n-5}{2}$ + 1
k= $\frac{n-3}{2}$.

Iteraciones del segundo *for*:

i= 0.
i= 1.
i= 2.
…
i= k - 1.

k - 1= j - 1
k= j - 1 + 1
k= j.

Iteraciones del tercer *for*:

k= 0.

k= 1.
k= 2.
…
k= k´ - 1.

k - 1= n - 1
k= n - 1 + 1
k= n.

Entonces:

T(n)= $\sum_{j=1}^{\frac{n-3}{2}} cte_1 \ (\sum_{i=1}^{j} cte_2 + \sum_{k=1}^{n} cte_3)$
T(n)= $\sum_{j=1}^{\frac{n-3}{2}} cte_1 \ (\sum_{i=1}^{j} cte_2 + n * cte_3)$
T(n)= $\sum_{j=1}^{\frac{n-3}{2}} cte_1 * j * (cte_2 + n * cte_3)$
T(n)= $cte_1 \ (cte_2 + \text{n } cte_3) \sum_{j=1}^{\frac{n-3}{2}} j$
T(n)= $cte_1 \ (cte_2 + \text{n } cte_3) \frac{\frac{n-3}{2}(\frac{n-3}{2}+1)}{2}$
T(n)= $cte_1 \ (cte_2 + \text{n } cte_3) \frac{\frac{n-3}{2}\frac{n-1}{2}}{2}$
T(n)= $cte_1 \ (cte_2 + \text{n } cte_3) \frac{\frac{n^2-n-3n+3}{4}}{2}$
T(n)= $cte_1 \ (cte_2 + \text{n } cte_3) \frac{\frac{n^2-4n+3}{4}}{2}$
T(n)= $cte_1 \ (cte_2 + \text{n } cte_3) \frac{n^2-4n+3}{8}$
T(n)= $cte_1 \ (cte_2 + \text{n } cte_3) (\frac{1}{8} n^2 - \frac{1}{2} \text{n} + \frac{3}{8})$
T(n)= $(\frac{1}{8} n^2 - \frac{1}{2} \text{n} + \frac{3}{8}) \ cte_1 cte_2 + (\frac{1}{8} n^3 - \frac{1}{2} n^2 + \frac{3}{8} \text{n}) \ cte_1 cte_3$.

Por lo tanto, el fragmento de código es de orden O(n^3).

Ejercicio 3.

Suponer que se dispone de un algoritmo A, que resuelve un problema de tamaño n, y su función de tiempo de ejecución es T(n)= n log(n). Este algoritmo se ejecuta en una computadora que procesa 10.000 operaciones por segundo. Determinar el tiempo que requerirá el algoritmo para resolver un problema de tamaño n= 1.024.

T(n)= n log_2 n.

T(1024)= 1024 log_2 1024
T(1024)= 1024 * 10
T(1024)= 10240.

10.000 operaciones por segundo.

Tiempo (en segundos)= $\frac{10240}{10000}$
Tiempo (en segundos)= 1,024.

Por lo tanto, el tiempo en segundos que requerirá el algoritmo para resolver un problema de tamaño n= 1.024 es 1,024.

Ejercicio 4.

¿Cuál es el resultado de la siguiente sumatoria? $\sum_{i=3}^{8} ni$.

(a) *(8-3+1) n.* **(b)** *(8-3+1) in.* **(c)** *33n.*
(d) *5n.* **(e)** *8i.* **(f)** *Ninguna de las otras opciones.*

$\sum_{i=3}^{8} ni$= n $\sum_{i=3}^{8} i$
$\sum_{i=3}^{8} ni$= n ($\sum_{i=1}^{8} i$ - $\sum_{i=1}^{2} i$)
$\sum_{i=3}^{8} ni$= n ($\sum_{i=1}^{8} i$ - $\sum_{i=1}^{2} i$)
$\sum_{i=3}^{8} ni$= n ($\frac{8*9}{2}$ - $\frac{2*3}{2}$)
$\sum_{i=3}^{8} ni$= n ($\frac{72}{2}$ - $\frac{6}{2}$)
$\sum_{i=3}^{8} ni$= n (36 - 3)
$\sum_{i=3}^{8} ni$= 33n.

Ejercicio 5.

¿Cuál de las siguientes sentencias es correcta, según la definición vista en clase?

(a) n^2 *es* $O(n^2)$.
(b) n^2 *es* $O(n^3)$.
(c) n^2 *es* $O(n^2 \log n)$.
(d) *Opciones a y b.*
(e) *Opciones a, b y c.*
(f) *Ninguna de las otras opciones.*

Ejercicio 6.

Dado el siguiente algoritmo:

```
void ejercicio5 (int n) {
   if (n ≥2) {
      2 * ejercicio5 (n/2);
      n = n/2;
      ejercicio5 (n/2);
   }
}
```

Indicar el T(n) para n ≥ 2:

(a) $T(n) = d + 3\ T(\frac{n}{2})$.
(b) $T(n) = d + 2\ T(\frac{n}{2}) + T(\frac{n}{4})$.
(c) $T(n) = d + T(\frac{n}{2}) + T(\frac{n}{4})$.
(d) $T(n) = d + T(\frac{n}{2}) + T(\frac{n}{2})$.
(e) $T(n) = d + T(\frac{n}{2}) + T(\frac{n}{2}) + T(\frac{n}{4})$.

Ejercicio 7.

Dada la recurrencia:

$$T(n) = \begin{cases} 1, para\ n \leq 1 \\ T(\frac{n}{3}) + c, para\ n > 1 \end{cases}$$

(a) *¿Cómo se reemplaza* $T(\frac{n}{3})$*, considerando* $\frac{n}{3} > 1$*?*

(i) $T(\frac{n}{3}) + c$.
(ii) *Ninguna de las otras opciones.*
(iii) $T(\frac{n}{3}) + 1$.
(iv) $T(\frac{\frac{n}{3}}{3}) + c$.
(v) $T(\frac{\frac{n}{3}}{3}) + 1$.

(b) *Desarrollar la función T(n).*

Paso 1:

T(n)= $T(\frac{n}{3})$ + c, si n > 1.

Paso 2:

T(n)= $T(\frac{\frac{n}{3}}{3})$ + c + c
T(n)= $T(\frac{n}{9})$ + 2c, si n > 2.

Paso 3:

T(n)= $T(\frac{\frac{n}{9}}{3})$ + c + 2c
T(n)= $T(\frac{n}{27})$ + 3c, si n > 3.

Paso i (Paso general):

T(n)= $T(\frac{n}{3^i})$ + ic, si n > i.

$\frac{n}{3^i}$= 1
1 * 3^i= n
3^i= n
i log_3 3= log_3 n
i * 1= log_3 n
i= log_3 n.

Entonces:

T(n)= T($\frac{n}{3^{log_3 n}}$) + log_3 n * c
T(n)= T($\frac{n}{n}$) + log_3 n * c
T(n)= T(1) + log_3 n * c
T(n)= 1 + log_3 n * c ≤ O(log_3 n).

Ejercicio 8.

Considerar el siguiente fragmento de código:

```
int count = 0; int n = a.length;
  for (int i = 0; i < n; i+=n/2)    {
                for (int j = 0; j < n; j++)  {
        a[j]++;
                }
  }
```

Este algoritmo se ejecuta en una computadora que procesa 100.000 operaciones por cada segundo. Determinar el tiempo aproximado que requerirá el algoritmo para resolver un problema de tamaño n= 1.000.

(a) *0,01 seg.* **(b)** *0,1 seg.* **(c)** *1 seg.* **(d)** *Ninguna de las opciones anteriores.*

T(n)= $cte_1 + \sum_{i=1}^{2} \sum_{j=1}^{n} cte_2$
T(n)= $cte_1 + \sum_{i=1}^{2} n * cte_2$
T(n)= cte_1 + 2n cte_2.

T(1000)≅ 1000.

100.000 operaciones por segundo.

Tiempo (en segundos)≅ $\frac{1000}{100000}$
Tiempo (en segundos)≅ 0,01.

Por lo tanto, el tiempo aproximado que requerirá el algoritmo para resolver un problema de tamaño n= 1.000 es 0,01 segundos.

Ejercicio 9.

Considerar la siguiente recurrencia:

T(1)= 4.
T(n)= 2 T($\frac{n}{2}$) + 5n + 1 (n ≥ 2).

¿Cuál es el valor de T(n) para n= 4?

(a) *51.* **(b)** *38.* **(c)** *59.* **(d)** *79.* **(e)** *Ninguna de las opciones anteriores.*

T(4)= 2 T($\frac{4}{2}$) + 5 * 4 + 1
T(4)= 2 T(2) + 20 + 1
T(4)= 2 T(2) + 21
T(4)= 2 [2 T($\frac{2}{2}$) + 5 * 2] + 21
T(4)= 2 [2 T(1) + 10 + 1] + 21
T(4)= 2 [2 T(1) + 11] + 21
T(4)= 2 (2 * 4 + 11) + 21
T(4)= 2 (8 + 11) + 21
T(4)= 2 * 19 + 21
T(4)= 38 + 21
T(4)= 59.

Ejercicio 10.

Expresar la función T(n) del siguiente segmento de código:

```
public static void ejercicio (int n) {
    int x = 0;
    int j = 1;
    while ( j <= n ) {
        for ( int i = n*n ; i >=1 ; i = i - 3 )
             x = x + 1 ;
        j = j * 2 ;
    }
}
```

(a) *T(n)*= $\frac{1}{3} n^2 + log_2 n$.
(b) *T(n)*= $n^2 + \frac{1}{3} log_2 n$.
(c) *T(n)*= $\frac{1}{3} log_2 n$.
(d) *T(n)*= $\frac{1}{3} n^2 \ log_2 n + log_2 n$.

Iteraciones del *while*:

j= 1.
j= 2.
j= 4.
…
j= 2^{k-1}.

2^{k-1}= n
$log_2 \ 2^{k-1} = log_2$ n
(k - 1) log_2 2= log_2 n
(k - 1) * 1= log_2 n
k - 1= log_2 n
k= log_2 n + 1.

Iteraciones del *for*:

i= n^2.
i= n^2 - 3.
i= n^2 - 3 * 2.
…
i= n^2 - 3 (k - 1).

n^2 - 3 (k - 1)= 1
3 (k - 1)= n^2 - 1

k - 1= $\frac{n^2-1}{3}$
k= $\frac{n^2-1}{3}$ + 1
k= $\frac{n^2+2}{3}$.

Entonces:

T(n)= $\sum_{j=1}^{log_2 n+1} \sum_{i=1}^{\frac{n^2+2}{3}} cte$
T(n)= $\sum_{j=1}^{log_2 n+1} \frac{n^2+2}{3} cte$
T(n)= (log_2 n + 1) $\frac{n^2+2}{3}$ cte
T(n)= (log_2 n + 1) ($\frac{1}{3} n^2 + \frac{2}{3}$) cte
T(n)= ($\frac{1}{3} n^2\ log_2$ n + $\frac{2}{3}\ log_2$ n + $\frac{1}{3} n^2 + \frac{2}{3}$) cte.

Ejercicio 11.

¿Cuál es el tiempo de ejecución del siguiente método?

```
void fun(int n, int arr[])
{    int i = 0, j = 0;
    for (; i < n; ++i)
        while (j < n && arr[i] < arr[j])
            j++;
}
```

Iteraciones del *for*:

i= 0.
i= 1.
i= 2.
…
i= k - 1.

k - 1= n - 1
k= n - 1 + 1
k= n.

Iteraciones del *while*:

j= 0.
j= 1.
j= 2.
…
j= k - 1.

k - 1= n - 1
k= n - 1 + 1
k= n.

Entonces:

T(n)= $cte_1 + \sum_{i=1}^{n} \sum_{j=1}^{n} cte_2$
T(n)= $cte_1 + \sum_{i=1}^{n} n * cte_2$
T(n)= cte_1 + nn cte_2
T(n)= $cte_1 + n^2\ cte_2$.

Por lo tanto, el tiempo de ejecución del método es T(n)= $cte_1 + n^2\ cte_2$.

Ejercicio 12.

¿Cuál es el valor que retorna el método fun1?

```
int fun1 (int n) {
   int i, j, k, p, q = 0;
   for (i = 1; i < n; ++i)    {
      p = 0;
      for (j = n; j > 1; j = j/2)
         ++p;
      for (k = 1; k < p; k = k*2)
         ++q;
   }
   return q;
}
```

Iteraciones del primer *for*:

i= 1.
i= 2.
i= 3.
…
i= k.

k= n - 1.

Iteraciones del segundo *for*:

j= n.
j= $\frac{n}{2}$.
j= $\frac{n}{4}$.
…
j= $\frac{n}{2^{k-1}}$.

$\frac{n}{2^{k-1}}$= 1 + 1
$\frac{n}{2^{k-1}}$= 2
$2 * 2^{k-1}$= n
2^k= n
$log_2\ 2^k = log_2$ n
k log_2 2= log_2 n
k * 1= log_2 n
k= log_2 n.

Iteraciones del tercer *for*:

k= 1.
k= 2.
k= 4.
…
k= $2^{k'-1}$.

2^{k-1}= p - 1
$log_2\ 2^{k-1} = log_2$ (p - 1)
(k - 1) log_2 2= log_2 (p - 1)
(k - 1) * 1= log_2 (p - 1)
k - 1= log_2 (p - 1)
k= log_2 (p - 1) + 1, si p > 1.

Entonces:

q= (n - 1) [log_2 (log_2 n - 1) + 1], si n > 2.

Por lo tanto, el valor que retorna *fun1* es q= (n - 1) [log_2 (log_2 n - 1) + 1], si n > 2.

Ejercicio 13.

¿Cuál es el tiempo de ejecución del siguiente código?

```
void fun(int n)
{
    for (int i = 0; i < n / 2; i++)
        for (int j = 1; j + n / 2 <= n; j++)
            for (int k = 1; k <= n; k = k * 2)
                System.out.print("AyED");
}

int main()
{
  int n=8;
  fun(3);
}
```

Iteraciones del primer *for*:

i= 0.
i= 1.
i= 2.
…
i= k - 1.

k - 1= $\frac{n}{2}$ - 1
k= $\frac{n}{2}$ - 1 + 1
k= $\frac{n}{2}$.

Iteraciones del segundo *for*:

j= 1.
j= 2.
j= 3.
…
j= k.

k + $\frac{n}{2}$= n
k= n - $\frac{n}{2}$
k= $\frac{n}{2}$.

Iteraciones del tercer *for*:

k= 1.
k= 2.
k= 4.
…
k= $2^{k'-1}$.

2^{k-1}= n
$log_2\ 2^{k-1}= log_2$ n
(k - 1) log_2 2= log_2 n
(k - 1) * 1= log_2 n
k - 1= log_2 n
k= log_2 n + 1.

Entonces:

T(n)= $\sum_{i=1}^{\frac{n}{2}} \sum_{j=1}^{\frac{n}{2}} \sum_{k=1}^{log_2 n+1} cte$
T(n)= $\sum_{i=1}^{\frac{n}{2}} \sum_{j=1}^{\frac{n}{2}} (log_2 n + 1)\ cte$
T(n)= $\sum_{i=1}^{\frac{n}{2}} \frac{n}{2} (log_2 n + 1)\ cte$
T(n)= $\frac{n}{2}\frac{n}{2}$ (log_2 n +1) cte
T(n)= $\frac{1}{4} n^2$ (log_2 n + 1) cte.

Por lo tanto, el tiempo de ejecución del código es T(n)= $\frac{1}{4} n^2$ (log_2 n + 1) cte.

Ejercicio 14.

¿Cuál es el tiempo de ejecución del siguiente código?

```
void fun(int a, int b)
{
    // Consider a and b both are positive integers
    while (a != b) {
        if (a > b)
            a = a - b;
        else
            b = b - a;
    }
}
```

Este algoritmo implementa el cálculo del máximo común divisor (MCD) utilizando el método de resta sucesiva. La complejidad temporal de este enfoque es O(max(a, b)), ya que, en cada iteración del bucle *while*, la variable más grande se reduce por la cantidad de la menor. En el peor de los casos, si *a* y *b* son muy desiguales, el número de iteraciones podría ser proporcional a la variable con mayor valor.

Ejercicio 15.

¿Cuál es el tiempo de ejecución del siguiente código?

```
void fun(int n)
{
  for(int i=0;i*i<n;i++)
    System.out.print("AyED");
}
```

Iteraciones del *for*:

i= 0.
i= 1.
i= 2.
…
i= k - 1.

(k - 1) (k - 1)= n - 1
$(k-1)^2$= n - 1
k - 1= $\sqrt{n-1}$
k= $\sqrt{n-1}$ + 1.

Entonces:

T(n)= $\sum_{i=1}^{\sqrt{n-1}+1} cte$
T(n)= ($\sqrt{n-1}$ + 1) cte.

Por lo tanto, el tiempo de ejecución del código es T(n)= ($\sqrt{n-1}$ + 1) cte.

Ejercicio 16.

¿Cuál es el tiempo de ejecución del siguiente código?

```
int fun(int n)
{
    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j < n; j += i)
        {
            // Some O(1) task
        }
    }
}
```

Nota: Tener en cuenta que ($\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}$) *se puede acotar con O(log n).*

Iteraciones del primer *for*:

i= 1.
i= 2.
i= 3.
…
i= k.

k= n.

Iteraciones del segundo *for*:

j= 1.
j= 1 + i
j= 1 + i * 2
…
j= 1 + i (k - 1).

1 + i (k - 1)= n - 1
i (k - 1)= n - 1 - 1
i (k - 1)= n - 2
k - 1= $\frac{n-2}{i}$
k= $\frac{n-2}{i}$ + 1.

Entonces:

T(n)= $\sum_{i=1}^{n} \sum_{j=1}^{\frac{n-2}{i}+1} cte$

T(n)= $\sum_{i=1}^{n}(\frac{n-2}{i}+1)\ cte$
T(n)= $\sum_{i=1}^{n}\frac{n-2}{i}\ cte + cte$
T(n)= $\sum_{i=1}^{n}\frac{n-2}{i}\ cte$ + $\sum_{i=1}^{n} cte$
T(n)= (n - 2) cte $\sum_{i=1}^{n}\frac{1}{i}$ + n cte
T(n)≅ cte (n - 2) log_2 n + n cte
T(n)≅ [(n - 2) log_2 n + n] cte.

Por lo tanto, el tiempo de ejecución del código es T(n)≅ [(n - 2) log_2 n + n] cte.

Juan Menduiña

Ejercitación Teórica N° 4:
Grafos.

Ejercicio 1.

Ejercicio 2.

Ejercicio 3.

Ejercicio 4.

Ejercicio 5.

Ejercicio 6.

Ejercicio 7.

Ejercicio 8.

Ejercicio 9.

Ejercicio 10.

Ejercicio 11.

Ejercicio 12.

Ejercicio 13.

Ejercicio 14.

Ejercicio 15.

Juan Menduiña

Trabajo Práctico N° 1:
Estructuras de Control y Estructuras de Datos Básicas en Java. Recursión.

Ejercicio 1.

Escribir tres métodos de clase (static) que reciban por parámetro dos números enteros (tipo int) a y b e impriman todos los números enteros comprendidos entre a; b (inclusive), uno por cada línea en la salida estándar. Para ello, dentro de una nueva clase, escribir un método por cada uno de los siguientes incisos:

- *Que realice lo pedido con un for.*
- *Que realice lo pedido con un while.*
- *Que realice lo pedido sin utilizar estructuras de control iterativas (for, while, do while).*

Por último, escribir, en el método de clase main, el llamado a cada uno de los métodos creados, con valores de ejemplo. En la computadora, ejecutar el programa y verificar que se cumple con lo pedido.

Ejercicio 2.

*Escribir un método de clase que, dado un número n, devuelva un nuevo arreglo de tamaño n con los n primeros múltiplos enteros de n mayores o iguales que 1. Ejemplo: f(5)= [5; 10; 15; 20; 25]; f(k)= {n*k, donde k: 1…k}. Agregar al programa la posibilidad de probar con distintos valores de n ingresándolos por teclado, mediante el uso de System.in. La clase Scanner permite leer, de forma sencilla, valores de entrada.*

Ejercicio 3.

Creación de instancias mediante el uso del operador new.

(a) *Crear una clase llamada Estudiante con los atributos especificados abajo y sus correspondientes métodos getters y setters (hacer uso de las facilidades que brinda eclipse).*

- *nombre*
- *apellido*
- *comision*
- *email*
- *direccion*

(b) *Crear una clase llamada Profesor con los atributos especificados abajo y sus correspondientes métodos getters y setters (hacer uso de las facilidades que brinda eclipse).*

- *nombre*
- *apellido*
- *email*
- *catedra*
- *facultad*

(c) *Agregar un método de instancia llamado tusDatos() en la clase Estudiante y en la clase Profesor, que retorne un String con los datos de los atributos de las mismas. Para acceder a los valores de los atributos, utilizar los getters previamente definidos.*

(d) *Escribir una clase llamada Test con el método main, el cual cree un arreglo con 2 objetos Estudiante, otro arreglo con 3 objetos Profesor, y, luego, recorrer ambos arreglos imprimiendo los valores obtenidos mediante el método tusDatos(). Recordar asignar los valores de los atributos de los objetos Estudiante y Profesor invocando los respectivos métodos setters.*

(e) *Agregar dos breakpoints, uno en la línea donde itera sobre los estudiantes y otro en la línea donde itera sobre los profesores.*

(f) *Ejecutar la clase Test en modo debug y avanzar paso a paso visualizando si el estudiante o el profesor recuperado es lo esperado.*

Ejercicio 4.

Pasaje de parámetros en Java.

(a) *Sin ejecutar el programa en la computadora, sólo analizándolo, indicar qué imprime el siguiente código.*

(b) *Ejecutar el ejercicio en la computadora y comparar el resultado con lo esperado en el inciso anterior.*

(c) *Insertar un breakpoint en las líneas donde se indica: y= tmp y ejecutar en modo debug. ¿Los valores que adoptan las variables x, y coinciden con los valores impresos por consola?*

Ejercicio 5.

Dado un arreglo de valores tipo entero, se desea calcular el valor máximo, mínimo y promedio en un único método. Escribir tres métodos de clase, donde respectivamente:

(a) *Devolver lo pedido por el mecanismo de retorno de un método en Java ("return").*

(b) *Devolver lo pedido interactuando con algún parámetro (el parámetro no puede ser de tipo arreglo).*

(c) *Devolver lo pedido sin usar parámetros ni la sentencia "return".*

Ejercicio 6.

Análisis de las estructuras de listas provistas por la API de Java.

(a) *¿En qué casos ArrayList ofrece un mejor rendimiento que LinkedList?*

- Frecuentes accesos aleatorios a los elementos: *ArrayList* usa un *array* dinámico internamente, por lo que acceder a un elemento por índice es O(1), mientras que, en *LinkedList*, es O(n) porque debe recorrer los nodos secuencialmente.
- Pocas inserciones/eliminaciones al inicio o medio de la lista: Si bien *LinkedList* es más eficiente en la eliminación/inserción en estos casos, si estas operaciones no son muy frecuentes, en general, *ArrayList* sigue siendo más rápido debido a su menor sobrecarga en memoria.
- Poco uso de memoria adicional: *ArrayList* almacena sólo los elementos, mientras que *LinkedList* usa referencias adicionales para enlazar nodos.
- Frecuentes operaciones de iteración: La iteración sobre un *ArrayList* es más rápida debido a la contigüidad de la memoria y la mejor localización en caché.

(b) *¿Cuándo LinkedList puede ser más eficiente que ArrayList?*

- Pocos accesos aleatorios a los elementos: Como *LinkedList* no tiene acceso directo a los elementos, sólo es útil si no se necesita acceder frecuentemente por índices.
- Frecuentes inserciones/eliminaciones al inicio o medio de la lista: *LinkedList* tiene O(1) en inserciones/eliminaciones en estos casos, mientras que *ArrayList* tiene O(n) debido al desplazamiento de elementos.
- Uso de *Iterator* para eliminaciones: Al eliminar elementos con un *Iterator*, *LinkedList* tiene un mejor rendimiento, ya que la eliminación es O(1), mientras que, en *ArrayList*, es O(n) por el desplazamiento.

(c) *¿Qué diferencia se encuentra en el uso de la memoria en ArrayList y LinkedList?*

- *ArrayList*: Usa menos memoria porque sólo almacena los elementos en un *array* contiguo, sin punteros adicionales.
- *LinkedList*: Usa más memoria porque cada nodo almacena el dato junto con dos referencias adicionales (*next* y *prev*), lo que aumenta el consumo de memoria significativamente.

(d) *¿En qué casos sería preferible usar un ArrayList o un LinkedList?*

Sería preferible usar un *ArrayList* cuando:

- Se requiere acceso rápido a los elementos por índice.

- Se realizan pocas inserciones/eliminaciones al inicio o medio de la lista.
- Se realizan recorridos de la lista frecuentemente.
- Se prioriza la eficiencia en el uso de memoria.

Sería preferible usar *LinkedList* cuando:

- No se requiere acceso rápido a los elementos por índice.
- Se realizan muchas inserciones/eliminaciones al inicio o medio de la lista.
- Se quiere usar *Iterator* para modificar la lista mientras se recorre.
- No se realizan recorridos de la lista frecuentemente.

En la mayoría de los casos, *ArrayList* es preferible debido a su menor consumo de memoria y mejor rendimiento general, excepto en escenarios con muchas modificaciones al inicio de la lista.

Ejercicio 7.

Uso de las estructuras de listas provistas por la API de Java. Para resolver este ejercicio, crear el paquete tp1.ejercicio7.

(a) *Escribir una clase llamada TestArrayList cuyo método main recibe una secuencia de números, los agrega a una lista de tipo ArrayList y, luego de haber agregado todos los números a la lista, imprime el contenido de la misma iterando sobre cada elemento.*

(b) *Si en lugar de usar un ArrayList en el inciso anterior se hubiera usado un LinkedList, ¿qué diferencia se encuentra respecto de la implementación? Justificar.*

Si en lugar de usar un *ArrayList* en el inciso anterior se hubiera usado un *LinkedList*, el acceso a los números sería muy ineficiente, ya que hay que recorrer la lista, no es directo.

(c) *¿Existen otras alternativas para recorrer los elementos de la lista del inciso (a)?*

Sí, existen otras alternativas para recorrer los elementos de la lista del inciso (a).

(d) *Escribir un método que realice las siguientes acciones:*

- *Crear una lista que contenga 3 estudiantes.*
- *Generar una nueva lista que sea una copia de la lista anterior.*
- *Imprimir el contenido de la lista original y el contenido de la nueva lista.*
- *Modificar algún dato de los estudiantes.*
- *Volver a imprimir el contenido de la lista original y el contenido de la nueva lista. ¿Qué conclusiones se obtiene a partir de lo realizado?*
- *¿Cuántas formas de copiar una lista existen? ¿Qué diferencias existen entre ellas?*

(e) *A la lista del inciso (d), agregar un nuevo estudiante. Antes de agregar, verificar que el estudiante no estaba incluído en la lista.*

(f) *Escribir un método que devuelva verdadero o falso si la secuencia almacenada en la lista es o no capicúa: public boolean esCapicua(ArrayList<Integer> lista).*

(g) *Considerar que se aplica la siguiente función de forma recursiva. A partir de un número n positivo se obtiene una sucesión que termina en 1:*

$$f(n) = \begin{cases} \frac{n}{2}, si\ n\ es\ par \\ 3n + 1, si\ n\ es\ impar \end{cases}.$$

Escribir un programa recursivo que, a partir de un número n, devuelva una lista con cada miembro de la sucesión.

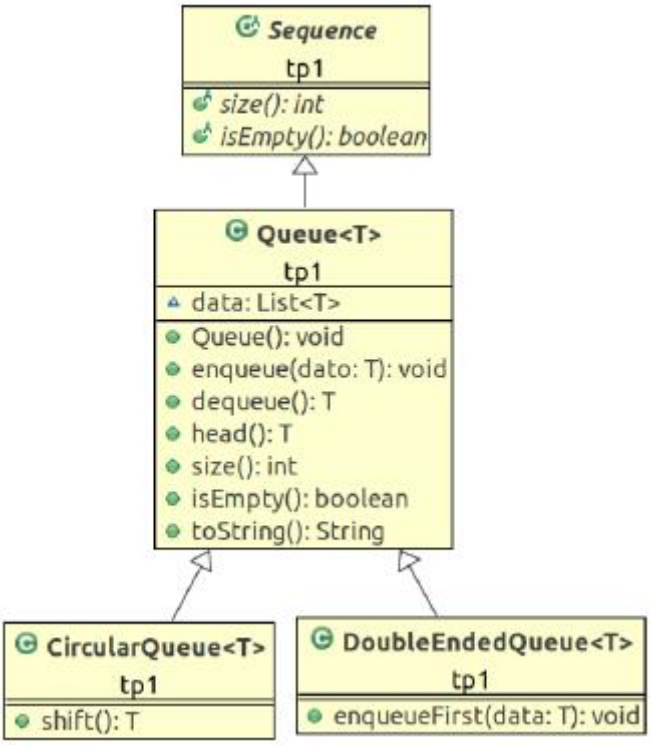
(h) *Implementar un método recursivo que invierta el orden de los elementos en un ArrayList: public void invertirArrayList(ArrayList<Integer> lista).*

(i) *Implementar un método recursivo que calcule la suma de los elementos en un LinkedList: public int sumarLinkedList(LinkedList<Integer> lista).*

(j) *Implementar el método "combinarOrdenado" que reciba 2 listas de números ordenados y devuelva una nueva lista también ordenada conteniendo los elementos de las 2 listas: public ArrayList<Integer> combinarOrdenado(ArrayList<Integer> lista1, ArrayList<Integer> lista2).*

Ejercicio 8.

El objetivo de este punto es ejercitar el uso de la API de listas de Java y aplicar conceptos de la programación orientada a objetos. Sean las siguientes especificaciones de cola, cola circular y cola con 2 extremos disponibles, vistas en la explicación teórica:



(a) *Implementar en JAVA la clase Queue de acuerdo con la especificación dada en el diagrama de clases. Definir esta clase dentro del paquete tp1.ejercicio8.*

- *Queue(): Constructor de la clase.*
- *enqueue(dato: T): Inserta el elemento al final de la cola.*
- *dequeue(): T Elimina el elemento del frente de la cola y lo retorna. Si la cola está vacía, se produce un error..*
- *head(): T Retorna el elemento del frente de la cola. Si la cola está vacía, se produce un error.*
- *isEmpty(): boolean Retorna verdadero si la cola no tiene elementos y falso en caso contrario.*
- *size(): int Retorna la cantidad de elementos de la cola.*
- *toString(): String Retorna los elementos de la cola en un String.*

(b) *Implementar en JAVA la clase CircularQueue de acuerdo con la especificación dada en el diagrama de clases. Definir esta clase dentro del paquete tp1.ejercicio8.*

- *shift(): T Permite rotar los elementos, haciéndolo circular. Retorna el elemento encolado.*

(c) *Implementar en JAVA la clase DoubleEndedQueue de acuerdo con la especificación dada en el diagrama de clases. Definir esta clase dentro del paquete tp1.ejercicio8.*

- *enqueueFirst(): void Permite encolar al inicio.*

Ejercicio 9.

Considerar un string de caracteres S, el cual comprende, únicamente, los caracteres: (,), [,], {, }. Se dice que S está balanceado si tiene alguna de las siguientes formas:

- *S = "" S es el string de longitud cero.*
- *S = "(T)".*
- *S = "[T]".*
- *S = "{T}".*
- *S = "TU".*

Donde ambos T y U son strings balanceados. Por ejemplo, "{()[()]}" está balanceado, pero "([)]" no lo está.

(a) *Indicar qué estructura de datos se utilizará para resolver este problema y cómo se utilizará.*

Se utilizará la estructura de datos *"Stack"*. Por cada signo de apertura, se hará un *PUSH* y, por cada signo de cierre, se hará un *POP*, verificando que sean del mismo tipo (el abrir y cerrar de signo). Si la pila queda vacía, significa que el *String* está balanceado. Además, se tendrán dos listas enlazadas, una con los caracteres de apertura y otra con los caracteres de cierre, para ir viendo si el caracter actual corresponde con alguno de esos seis caracteres y, dependiendo de eso, se realizará la operación correspondiente (*push* o *pop*).

(b) *Implementar una clase llamada tp1.ejercicio9.TestBalanceo, cuyo objetivo es determinar si un String dado está balanceado. El String a verificar es un parámetro de entrada (no es un dato predefinido).*

Ejercicio 10.

Considerar el siguiente problema: Se quiere modelar la cola de atención en un banco. A medida que la gente llega al banco, toma un ticket para ser atendido, sin embargo, de acuerdo a la LEY 14.564 de la Provincia de Buenos Aires, se establece la obligatoriedad de otorgar prioridad de atención a mujeres embarazadas, a personas con necesidades especiales o movilidad reducida y a personas mayores de setenta (70) años. De acuerdo a las estructuras de datos vistas en esta práctica, ¿qué estructura de datos se sugeriría para el modelado de la cola del banco?

Para el modelado de la cola del banco, la estructura de datos que se sugeriría es "DoubleEndedQueue", es decir, una cola de doble extremo, ya que esta estructura permite agregar tanto al principio como al final de la cola, con lo cual las personas con orden de prioridad serán puestas al principio de la cola, mientras que las personas que no tengan este orden serán puestas al final de la cola.

Ejercicio 11.

Considerar el siguiente problema: Se quiere modelar el transporte público de la ciudad de La Plata, lo cual involucra las líneas de colectivos y sus respectivas paradas. Cada línea de colectivos, tiene asignado un conjunto de paradas donde se detiene de manera repetida durante un mismo día. De acuerdo a las estructuras de datos vistas en esta práctica, ¿qué estructura de datos se sugeriría para el modelado de las paradas de una línea de colectivos?

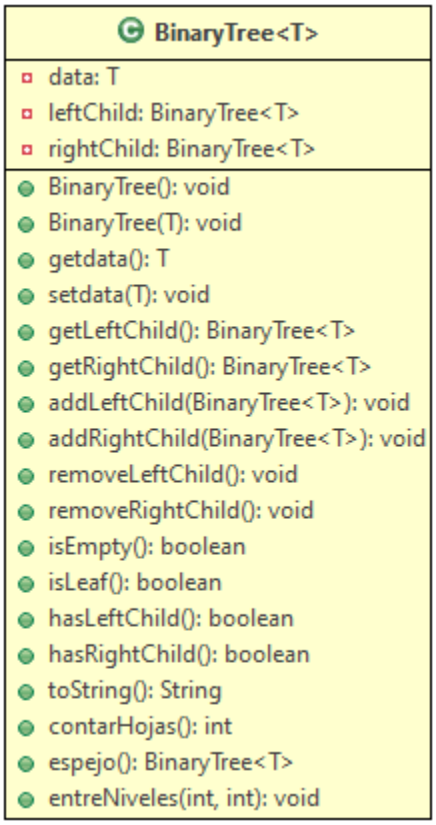
Para el modelado de las paradas de una línea de colectivos, la estructura de datos que se sugeriría es “CircularQueue”, es decir, una cola circular, ya que esta estructura permite que, cuando un colectivo llega a la última parada, automáticamente, vuelve a la primera, sin necesidad de reiniciar la estructura.

Juan Menduiña

Trabajo Práctico N° 2:
Árboles Binarios.

Ejercicio 1.

Considerar la siguiente especificación de la clase Java BinaryTree (con la representación hijo izquierdo e hijo derecho):



- *El constructor BinaryTree(T data) inicializa un árbol con el dato pasado como parámetro y ambos hijos nulos.*
- *Los métodos getLeftChild(): BinaryTree<T> y getRightChild(): BinaryTree<T> retornan los hijos izquierdo y derecho, respectivamente, del árbol. Si no tiene el hijo, tira error.*
- *El método addLeftChild(BinaryTree<T> child) y addRightChild(BinaryTree<T> child) agrega un hijo como hijo izquierdo o derecho del árbol.*
- *El método removeLeftChild() y removeRightChild() eliminan el hijo correspondiente.*
- *El método isEmpty() indica si el árbol está vacío y el método isLeaf() indica si no tiene hijos.*
- *El método hasLeftChild() y hasRightChild() devuelve un booleano indicando si tiene dicho hijo el árbol receptor del mensaje.*

Analizar la implementación en JAVA de la clase BinaryTree brindada por la cátedra.

Ejercicio 2.

Agregar, a la clase BinaryTree, los siguientes métodos:

(a) *contarHojas(): int Devuelve la cantidad de árbol/subárbol hojas del árbol receptor.*

(b) *espejo(): BinaryTree<T> Devuelve el árbol binario espejo del árbol receptor.*

(c) *entreNiveles(int n, m) Imprime el recorrido por niveles de los elementos del árbol receptor entre los niveles n y m (ambos inclusive). (0 ≤ n < m ≤ altura del árbol).*

Ejercicio 3.

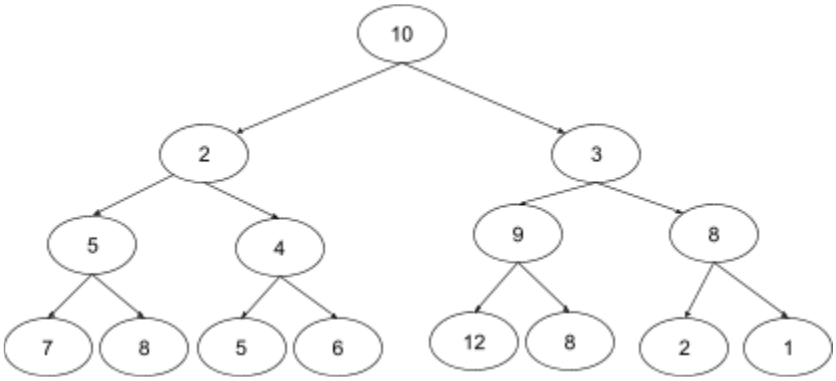
Definir una clase Java denominada ContadorArbol cuya función principal es proveer métodos de validación sobre árboles binarios de enteros. Para ello, la clase tiene como variable de instancia un BinaryTree<Integer>. Implementar, en dicha clase, un método denominado numerosPares() que devuelve, en una estructura adecuada (sin ningún criterio de orden), todos los elementos pares del árbol (divisibles por 2).

(a) *Implementar el método realizando un recorrido InOrden.*

(b) *Implementar el método realizando un recorrido PostOrden.*

Ejercicio 4.

Una red binaria es una red que posee una topología de árbol binario lleno. Por ejemplo:



Los nodos que conforman una red binaria llena tienen la particularidad de que todos ellos conocen cuál es su retardo de reenvío. El retardo de reenvío se define como el período comprendido entre que un nodo recibe un mensaje y lo reenvía a sus dos hijos.

La tarea es calcular el mayor retardo posible, en el camino que realiza un mensaje desde la raíz hasta llegar a las hojas en una red binaria llena. En el ejemplo, se debería retornar 10 + 3 + 9 + 12= 34 (si hay más de un máximo, retornar el último valor hallado).

NOTA: Asumir que cada nodo tiene el dato de retardo de reenvío expresado en cantidad de segundos.

(a) *Indicar qué estrategia (recorrido en profundidad o por niveles) se utilizará para resolver el problema.*

Para resolver el problema, se utilizará un recorrido en profundidad.

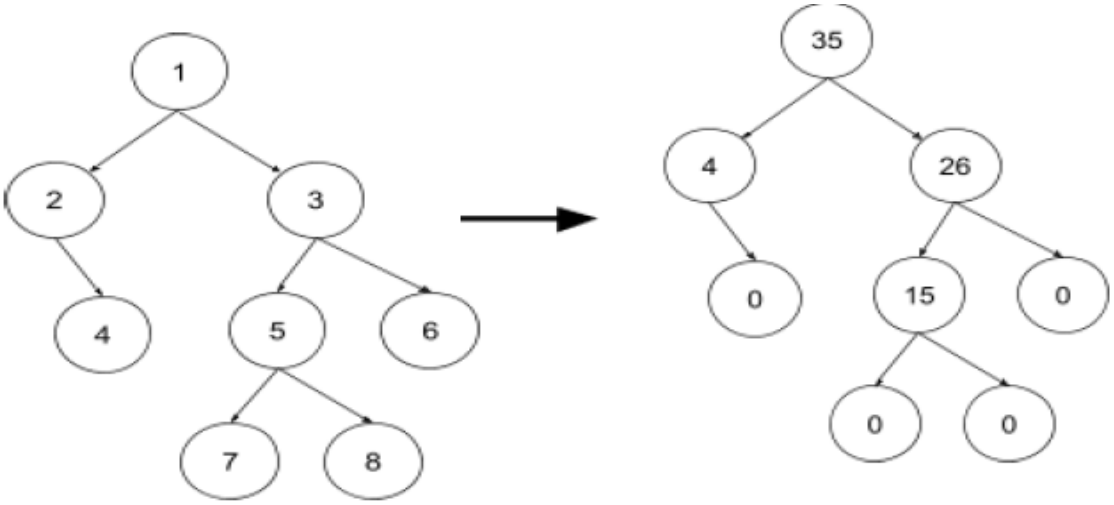
(b) *Crear una clase Java llamada RedBinariaLlena donde se implementará lo solicitado en el método retardoReenvio(): int.*

Ejercicio 5.

Implementar una clase Java llamada ProfundidadDeArbolBinario que tiene, como variable de instancia, un árbol binario de números enteros y un método de instancia sumaElementosProfundidad(int p): int, el cuál devuelve la suma de todos los nodos del árbol que se encuentren a la profundidad pasada como argumento.

Ejercicio 6.

Crear una clase Java llamada Transformacion que tenga como variable de instancia un árbol binario de números enteros y un método de instancia suma(): BinaryTree<Integer>, el cual devuelve el árbol en el que se reemplazó el valor de cada nodo por la suma de todos los elementos presentes en su subárbol izquierdo y derecho. Asumir que los valores de los subárboles vacíos son ceros. Por ejemplo:



¿La solución recorre una única vez cada subárbol? En el caso que no, ¿se puede mejorar para que sí lo haga?

Ejercicio 7.

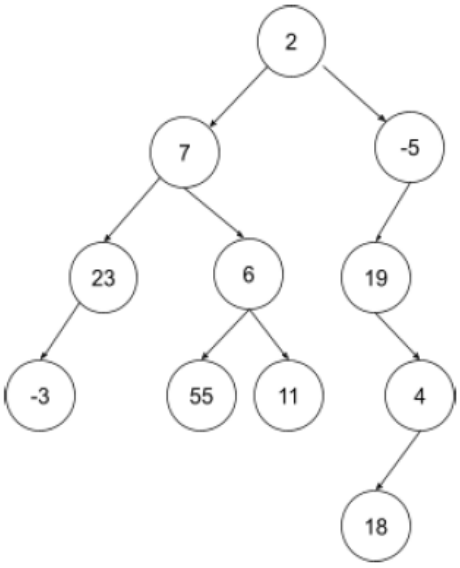
Escribir una clase ParcialArboles que contenga UNA ÚNICA variable de instancia de tipo BinaryTree de valores enteros NO repetidos y el método público con la siguiente firma:

public boolean isLeftTree(int num).

El método devuelve true si el subárbol cuya raíz es "num" tiene, en su subárbol izquierdo, una cantidad mayor estricta de árboles con un único hijo que en su subárbol derecho. Y false en caso contrario. Consideraciones:

- *Si "num" no se encuentra en el árbol, devuelve false.*
- *Si el árbol con raíz "num" no cuenta con una de sus ramas, considerar que, en esa rama, hay -1 árboles con único hijo.*

Por ejemplo, con un árbol como se muestra en la siguiente imagen:



Si num = 7 devuelve **true** ya que en su rama izquierda hay 1 árbol con un único hijo (el árbol con raíz 23) y en la rama derecha hay 0. (1 > 0) → true

Si num = 2 devuelve **false**, ya que en su rama izquierda hay 1 árbol con único hijo (árbol con raíz 23) y en la rama derecha hay 3 (árboles con raíces -5, 19 y 4). (1 > 3) → false)

Si num = -5 devuelve **true**, ya que en su rama izquierda hay 2 árboles con único hijo (árboles con raíces 19 y 4) y al no tener rama derecha, tiene -1 árboles con un único hijo. (2 > -1) → true

Si num = 19 debería devolver **false**, ya que al no tener rama izquierda tiene -1 árboles con un único hijo y en su rama derecha hay 1 árbol con único hijo. (-1 > 1) → false

Si num = -3 debería devolver **false**, ya que al no tener rama izquierda tiene -1 árboles con un único hijo y lo mismo sucede con su rama derecha. (-1 > -1) → false

Ejercicio 8.

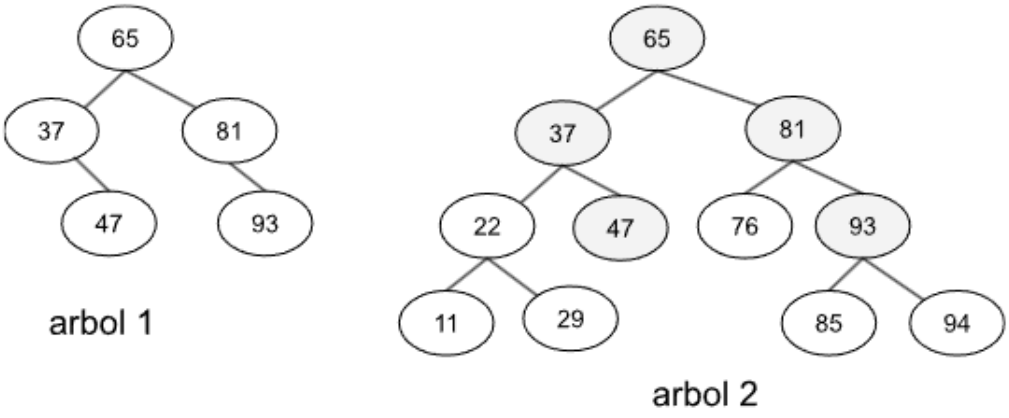
Escribir, en una clase ParcialArboles, el método público con la siguiente firma:

public boolean esPrefijo(BinaryTree<Integer> arbol1, BinaryTree<Integer> arbol2).

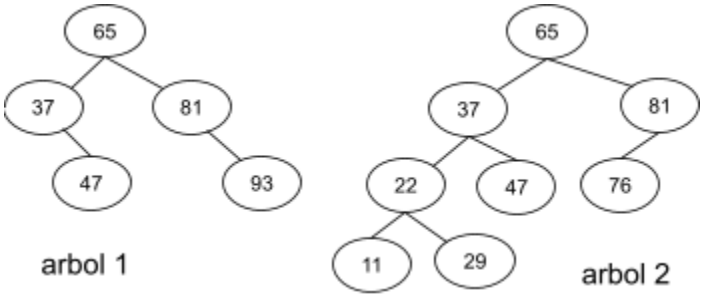
El método devuelve true si arbol1 es prefijo de arbol2, false en caso contrario.

Se dice que un árbol binario arbol1 es prefijo de otro árbol binario arbol2 cuando arbol1 coincide con la parte inicial del árbol arbol2 tanto en el contenido de los elementos como en su estructura.

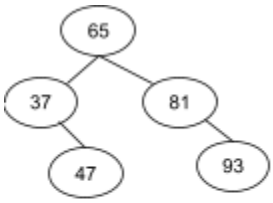
Por ejemplo, en la siguiente imagen, arbol1 ES prefijo de arbol2.



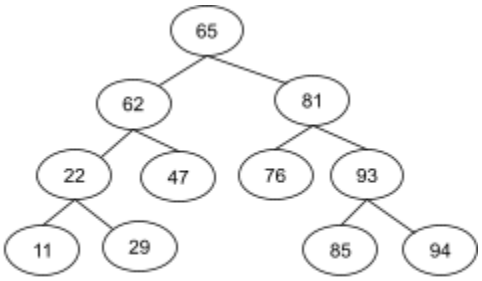
En esta otra, arbol1 NO es prefijo de arbol2 (el subárbol con raíz 93 no está en el árbol2).



En la siguiente, no coincide el contenido. El subárbol con raíz 37 figura con raíz 62, entonces, arbol1 NO es prefijo de arbol2.



arbol 1



arbol 2

Ejercicio 9.

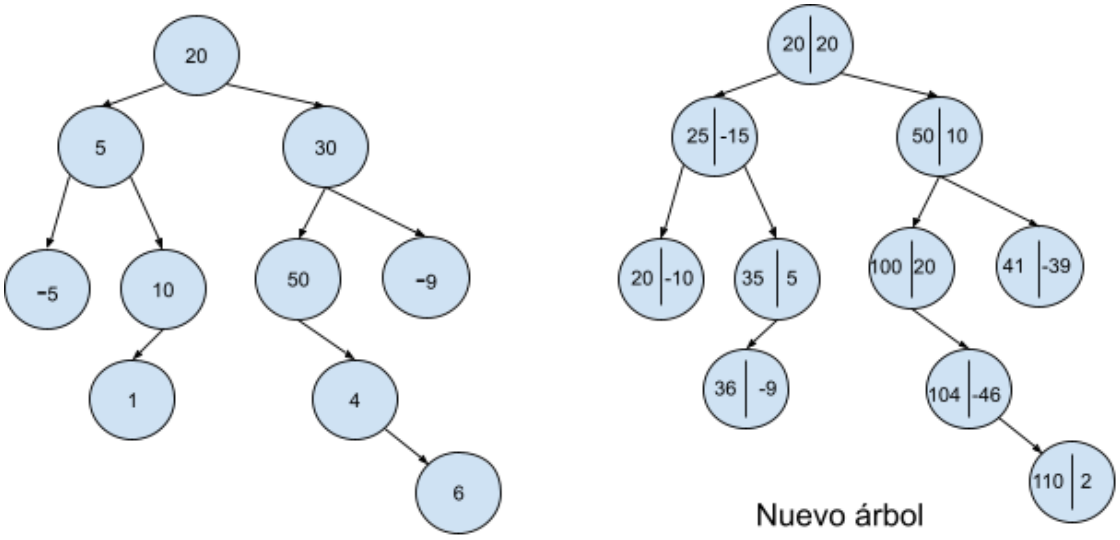
Escribir, en una clase ParcialArboles, el método público con la siguiente firma:

public BinaryTree<?> sumAndDif(BinaryTree<Integer> arbol).

El método recibe un árbol binario de enteros y devuelve un nuevo árbol que contiene, en cada nodo, dos tipos de información:

- *La suma de los números a lo largo del camino desde la raíz hasta el nodo actual.*
- *La diferencia entre el número almacenado en el nodo original y el número almacenado en el nodo padre.*

Ejemplo:



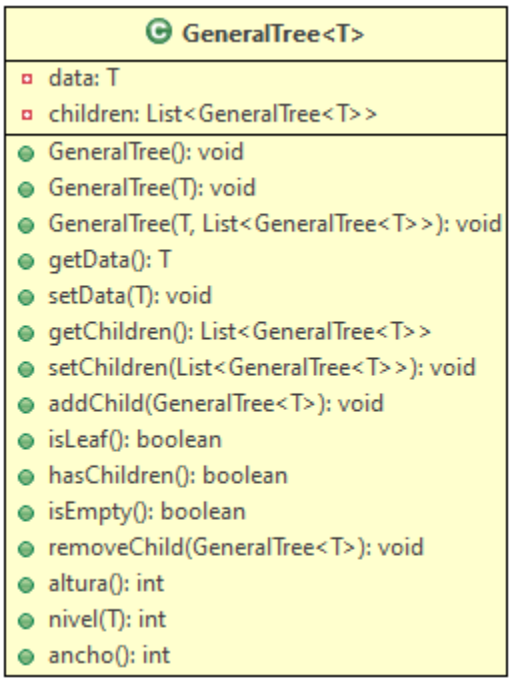
NOTA: En el nodo raíz, considerar que el valor del nodo padre es 0.

Juan Menduiña

Trabajo Práctico N° 3:
Árboles Generales.

Ejercicio 1.

Considerar la siguiente especificación de la clase GeneralTree (con la representación de Lista de Hijos).



- *El constructor GeneralTree(T data) inicializa un árbol que tiene como raíz un nodo y este nodo tiene el dato pasado como parámetro y una lista vacía.*
- *El constructor GeneralTree(T data, List<GeneralTree <T>> children) inicializa un árbol que tiene como raíz a un nodo y este nodo tiene el dato pasado como parámetro y como hijos children.*
- *El método getData(): T retorna el dato almacenado en la raíz del árbol.*
- *El método getChildren(): List<GeneralTree <T>> retorna la lista de hijos de la raíz del árbol.*
- *El método addChild(GeneralTree <T> child) agrega un hijo al final de la lista de hijos del árbol.*
- *El método hasChildren() devuelve verdadero si la lista de hijos del árbol no es null y tampoco es vacía.*
- *El método isEmpty() devuelve verdadero si el dato del árbol es null y, además, no tiene hijos.*
- *El método removeChild(GeneralTree <T> child) elimina del árbol el hijo pasado como parámetro.*
- *Los métodos altura(), nivel(T) y ancho() se resolverán en el Ejercicio 3.*

Analizar la implementación en JAVA de la clase GeneralTree brindada por la cátedra.

Ejercicio 2.

(a) *Implementar, en la clase RecorridosAG, los siguientes métodos:*

- *public List<Integer> numerosImparesMayoresQuePreOrden(GeneralTree<Integer> a, Integer n) Método que retorna una lista con los elementos impares del árbol "a" que sean mayores al valor "n" pasados como parámetros, recorrido en preorden.*
- *public List<Integer> numerosImparesMayoresQueInOrden(GeneralTree<Integer> a, Integer n) Método que retorna una lista con los elementos impares del árbol "a" que sean mayores al valor "n" pasados como parámetros, recorrido en inorden.*
- *public List<Integer> numerosImparesMayoresQuePostOrden(GeneralTree<Integer> a, Integer n) Método que retorna una lista con los elementos impares del árbol "a" que sean mayores al valor "n" pasados como parámetros, recorrido en postorden.*
- *public List<Integer> numerosImparesMayoresQuePorNiveles(GeneralTree<Integer> a, Integer n) Método que retorna una lista con los elementos impares del árbol "a" que sean mayores al valor "n" pasados como parámetros, recorrido por niveles.*

(b) *Si, ahora, se tuviera que implementar estos métodos en la clase GeneralTree<T>, ¿qué modificaciones se harían tanto en la firma como en la implementación de los mismos?*

Ejercicio 3.

Implementar, en la clase GeneralTree, los siguientes métodos:

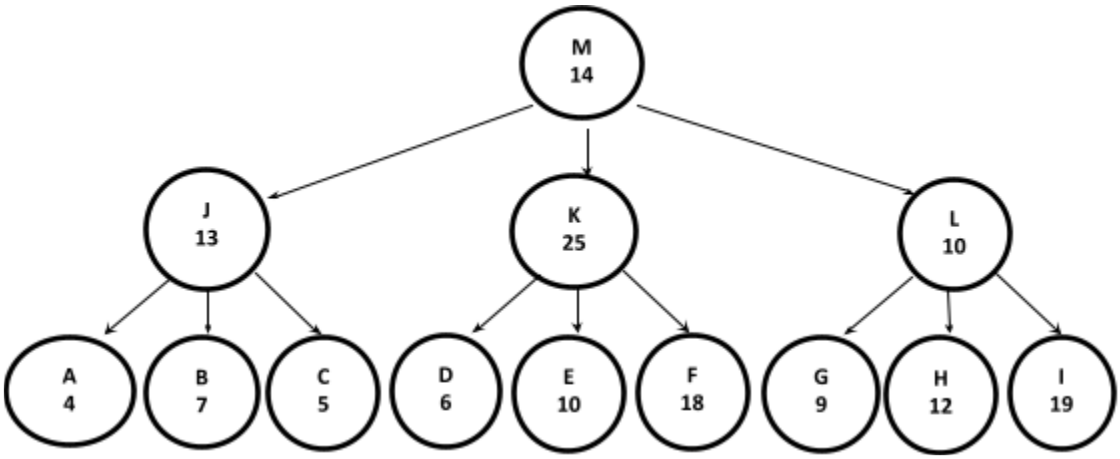
(a) *public int altura(): Devuelve la altura del árbol, es decir, la longitud del camino más largo desde el nodo raíz hasta una hoja.*

(b) *public int nivel(T dato): Devuelve la profundidad o nivel del dato en el árbol. El nivel de un nodo es la longitud del único camino de la raíz al nodo.*

(c) *public int ancho(): Devuelve la amplitud (ancho) de un árbol, que se define como la cantidad de nodos que se encuentran en el nivel que posee la mayor cantidad de nodos.*

Ejercicio 4.

El esquema de comunicación de una empresa está organizado en una estructura jerárquica, en donde cada nodo envía el mensaje a sus descendientes. Cada nodo posee el tiempo que tarda en transmitir el mensaje.



Se debe devolver el mayor promedio entre todos los valores promedios de los niveles. Para el ejemplo presentado, el promedio del nivel 0 es 14, el del nivel 1 es 16 y el del nivel 2 es 10. Por lo tanto, debe devolver 16.

(a) *Indicar y justificar qué tipo de recorrido se utilizará para resolver el problema.*

El tipo de recorrido que se utilizará para resolver el problema será por niveles.

(b) *Implementar, en una clase AnalizadorArbol, el método con la siguiente firma:*

public double devolverMaximoPromedio(GeneralTree<AreaEmpresa> arbol),

donde AreaEmpresa es una clase que representa a un área de la empresa mencionada y que contiene la identificación de la misma representada con un String y una tardanza de transmisión de mensajes interna representada con int.

Ejercicio 5.

Se dice que un nodo n es ancestro de un nodo m si existe un camino desde n a m. Implementar un método en la clase GeneralTree con la siguiente firma:

public boolean esAncestro(T a, T b): Devuelve true si el valor "a" es ancestro del valor "b".

Ejercicio 6.

Sea una red de agua potable, la cual comienza en un caño maestro y la misma se va dividiendo, sucesivamente, hasta llegar a cada una de las casas. Por el caño maestro, ingresan "x" cantidad de litros y, en la medida que el caño se divide de acuerdo con las bifurcaciones que pueda tener, el caudal se divide en partes iguales en cada una de ellas. Es decir, si un caño maestro recibe 1000 litros y tiene, por ejemplo, 4 bifurcaciones, se divide en 4 partes iguales, donde cada división tendrá un caudal de 250 litros. Luego, si una de esas divisiones se vuelve a dividir, por ejemplo, en 5 partes, cada una tendrá un caudal de 50 litros y así sucesivamente hasta llegar a un lugar sin bifurcaciones.

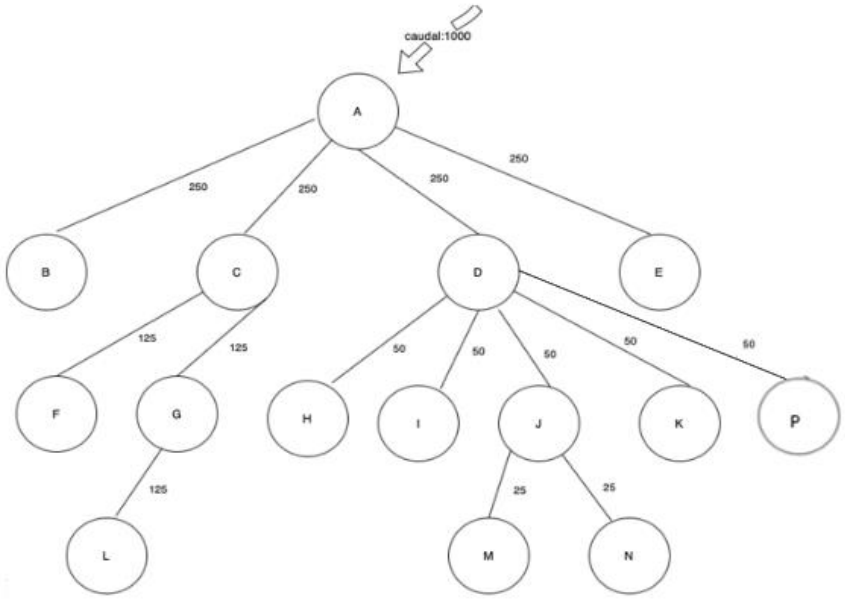
Se debe implementar una clase RedDeAguaPotable que contenga el método con la siguiente firma:

public double minimoCaudal(double caudal),

que calcule el caudal de cada nodo y determine cuál es el caudal mínimo que recibe una casa.

Asumir que la estructura de caños de la red está representada por una variable de instancia de la clase RedAguaPotable y que es un GeneralTree<Character>.

Extendiendo el ejemplo en el siguiente gráfico, al llamar al método minimoCaudal con un valor de 1000.0, debería retornar 25.0.

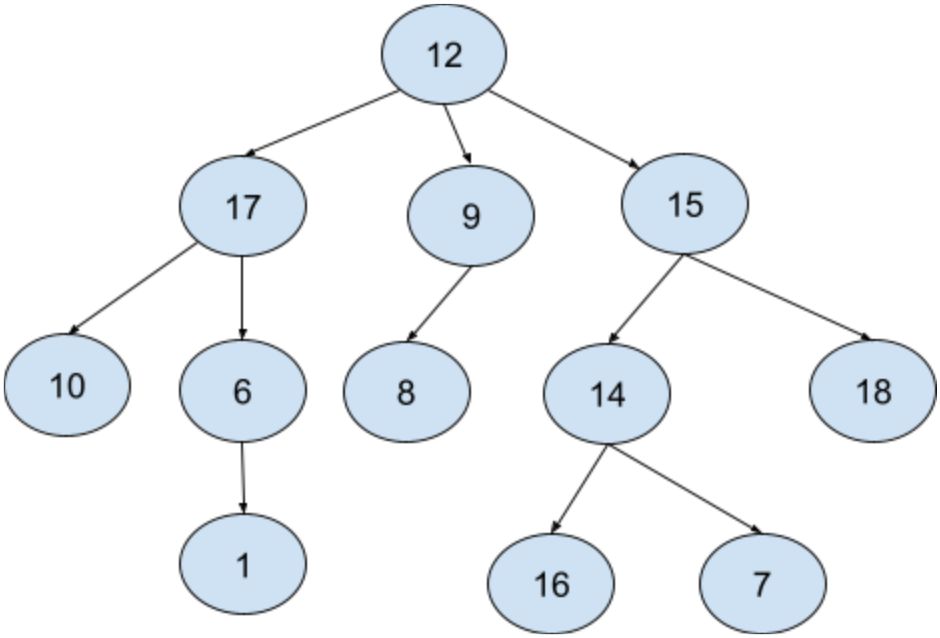


Ejercicio 7.

Dada una clase Caminos que contiene una variable de instancia de tipo GeneralTree de números enteros, implementar un método que retorne el camino a la hoja más lejana. En el caso de haber más de un camino máximo, retornar el primero que se encuentre. El método debe tener la siguiente firma:

public List<Integer> caminoAHojaMasLejana().

Por ejemplo, para el siguiente árbol, la lista a retornar sería: 12, 17, 6, 1 de longitud 3 (los caminos 12, 15, 14, 16 y 12, 15, 14, 7 son también máximos, pero se pide el primero).



Ejercicio 8.

Retomando el ejercicio abeto navideño visto en teoría, crear una clase Navidad que cuenta con una variable de instancia GeneralTree que representa al abeto (ya creado) e implementar el método con la firma: public String esAbetoNavidenio().

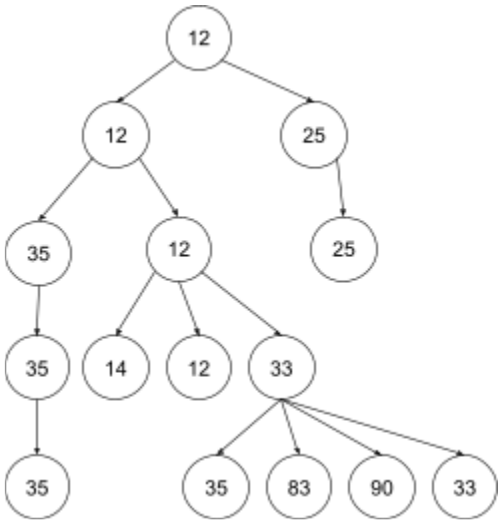
Ejercicio 9.

Implementar, en la clase ParcialArboles, el método:

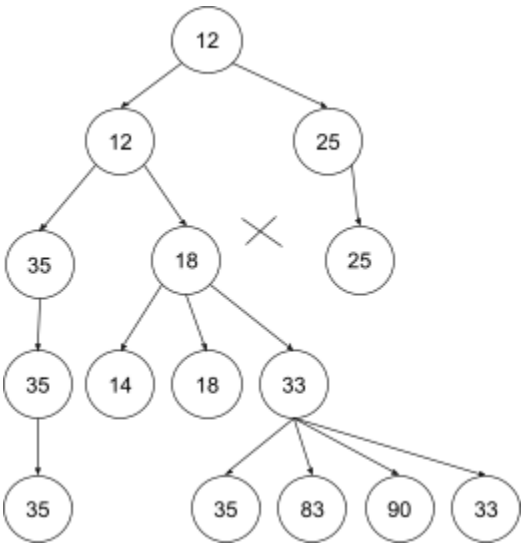
public static boolean esDeSeleccion(GeneralTree<Integer> arbol),

que devuelve true si el árbol recibido por parámetro es de selección, falso sino lo es.

Un árbol general es de selección si cada nodo tiene, en su raíz, el valor del menor de sus hijos. Por ejemplo, para el siguiente árbol, se debería retornar true.



Para este otro árbol, se debería retornar false (el árbol con raíz 18 tiene un hijo con valor mínimo 14).



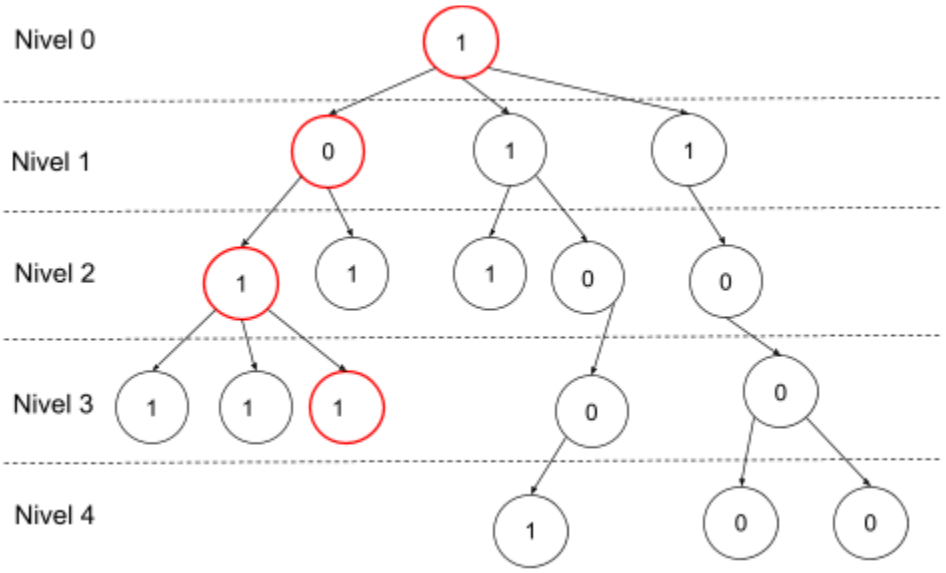
Ejercicio 10.

Implementar la clase ParcialArboles y el método:

public static List<Integer> resolver(GeneralTree<Integer> arbol),

que recibe un árbol general de valores enteros, que sólo pueden ser 0 o 1, y devuelve una lista con los valores que componen el "camino filtrado de valor máximo". Se llama "filtrado" porque sólo se agregan al camino los valores iguales a 1 (los 0 no se agregan), mientras que es "de valor máximo" porque se obtiene de realizar el siguiente cálculo: es la suma de los valores de los nodos multiplicados por su nivel. De haber más de uno, devolver el primero que se encuentre.

*Por ejemplo, para el árbol general que aparece en el gráfico, el resultado de la invocación al método resolver debería devolver una lista con los valores 1, 1, 1, y NO 1, 0, 1, 1, dado que se filtró el valor 0. Con esa configuración, se obtiene el mayor valor según el cálculo: 1*0 + 0*1 + 1*2 + 1*3 (el camino 1*0 + 1*1 + 0*2 + 0*3 + 1*4 también da 5, pero no es el primero).*



NOTA: No se puede generar la lista resultado con 0/1 y, en un segundo recorrido, eliminar los elementos con valor 0.

Ejercicio 11.

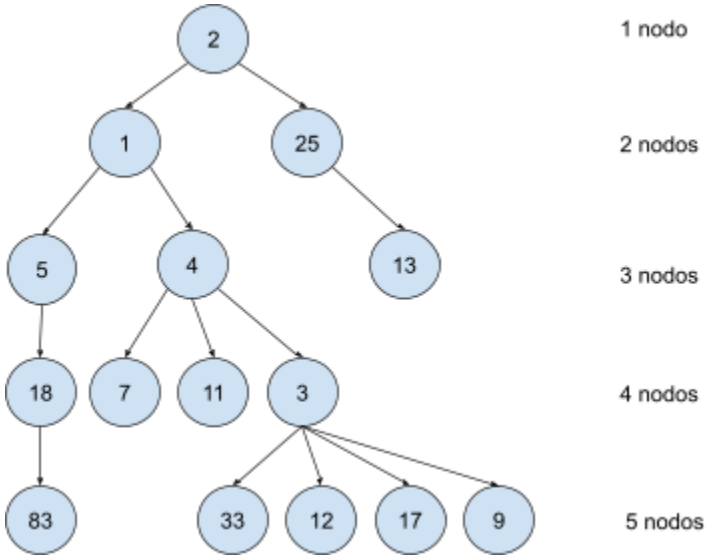
Implementar, en la clase ParcialArboles, el método:

public static boolean resolver(GeneralTree<Integer> arbol),

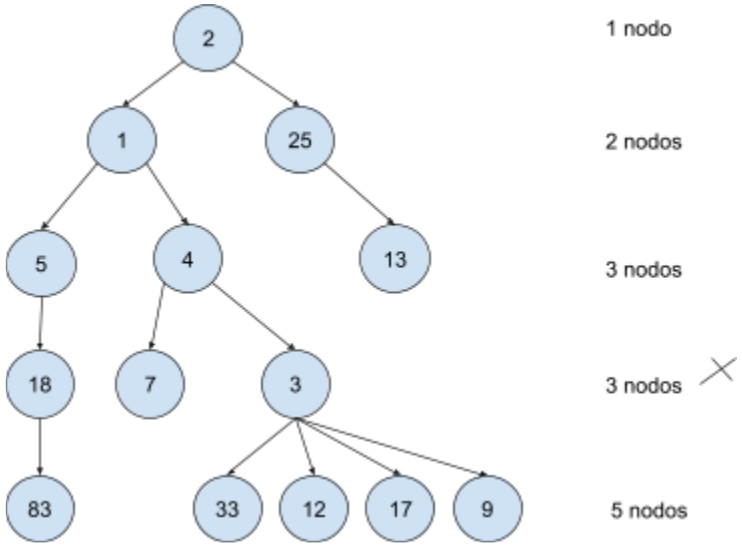
que devuelve true si el árbol es creciente, falso sino lo es.

Un árbol general es creciente si, para cada nivel del árbol, la cantidad de nodos que hay en ese nivel es, exactamente, igual a la cantidad de nodos del nivel anterior + 1.

Por ejemplo, para el siguiente árbol, se debería retornar true.



Para este otro árbol, se debería retornar false (ya que, en el nivel 3, debería haber 4 nodos y no 3).



Juan Menduiña

Trabajo Práctico N° 4:
Tiempos de Ejecución.

Ejercicio 1.

Debido a un error en la actualización de sus sistemas, el banco AyED perdió la información del estado de todas sus cuentas. Afortunadamente, logran recuperar un backup del día anterior y, utilizando las transacciones registradas en las últimas 24hrs, podrán reconstruir los saldos. Hay poco tiempo que perder, el sistema bancario debe volver a operar lo antes posible.

Las transacciones se encuentran agrupadas en consultas, una consulta cuenta con un valor y un rango de cuentas consecutivas a las que hay que aplicar este cambio, por ejemplo la consulta (333..688= 120) implica sumar $120 a todas las cuentas entre la número 333 y la número 688 (inclusive). Entonces, la recuperación de los datos consiste en aplicar todas las consultas sobre el estado de las cuentas recuperadas en el backup del día anterior.

El equipo de desarrollo se pone manos a la obra y llega a una solución rápidamente (Algoritmo procesarMovimientos). Toman cada consulta y recorren el rango de cuentas aplicando el valor correspondiente, como muestra el siguiente algoritmo.

```
Consultas.comenzar()
While(!consultas.fin()){
        Consulta = consultas.proximo();
        for(i = consulta.desde; i < consulta.hasta; i++){
                cuenta[i] = cuenta[i] + consulta.valor;
        }
}
```

Escriben la solución en pocos minutos y ponen en marcha el proceso de recuperación. Enseguida se dan cuenta que el proceso va a tardar muchas horas en finalizar, son muchas cuentas y muchos movimientos, la solución aunque simple es ineficiente. Luego de discutir varias ideas llegan a una solución (Algoritmo procesarMovimientosOptimizado) que logra procesar toda la información en pocos segundos. Ambos algoritmos se encuentran en el archivo Ejercicio 1 - rsq_tn_ayed.zip del material adicional.

(a) *Para que se pueda experimentar el tiempo que demora cada uno de los dos algoritmos en forma empírica, se debe ejecutar cada uno de ellos, con distintas cantidades de elementos y completar la tabla. Luego, hacer la gráfica para comparar los tiempos de ambos algoritmos. Tener en cuenta que el algoritmo posee dos constantes CANTIDAD_CUENTAS y CANTIDAD_CONSULTAS, sin embargo, por simplicidad, ambas toman el mismo valor. Sólo se necesita modificar CANTIDAD_CUENTAS.*

N° Cuentas (y consultas)	procesarMovimientos	procesarMovimientosOptimizado
1.000	0,034	0,001
10.000	0,036	0,003
25.000	0,034	0,006
50.000	0,033	0,011
100.000	0,046	0,019

(b) *¿Por qué procesarMovimientos es tan ineficiente? Tener en cuenta que pueden existir millones de movimientos diarios que abarquen gran parte de las cuentas bancarias.*

El método tiene una complejidad temporal de O(N*M), donde N es la cantidad de consultas y M es el promedio del tamaño de los rangos de cuentas afectadas por cada consulta. En el caso que haya muchas consultas o los rangos sean grandes, este algoritmo sería muy ineficiente.

(c) *¿En qué se diferencia procesarMovimientosOptimizado? Observar las operaciones que se realizan para cada consulta. Aunque los dos algoritmos se encuentran explicados en los comentarios, no es necesario entender su funcionamiento para contestar las preguntas.*

En lo que se diferencia *procesarMovimientosOptimizado* es en que hace un único recorrido tanto en el arreglo de consultas como el de cuentas. Esto es así gracias a la ayuda de un arreglo auxiliar en donde se registran los cambios que deben aplicarse a las cuentas sumando los valores correspondientes (en el anterior algoritmo, los cambios se realizaban, directamente, dentro del bucle anidado).

Ejercicio 2.

La clase BuscadorEnArrayOrdenado del material adicional (Ejercicio 2 - Tiempo.zip) resuelve el problema de buscar un elemento dentro de un array ordenado. El mismo problema lo resuelve de dos maneras diferentes: búsqueda lineal y búsqueda dicotómica.

Se define la variable cantidadElementos, la cual se va modificando para determinar una escala (por ejemplo, de a 100.000 o 1.000.000, dependiendo de la capacidad de cada equipo). Realizar una tabla con el tiempo que tardan en ejecutarse ambos algoritmos, para los distintos valores de la variable.

Por ejemplo:

N	Lineal	Dicotómica
100.000	0,002	0,000
200.000	0,005	0,000
300.000	0,006	0,000
400.000	0,007	0,000
500.000	0,008	0,000
600.000	0,011	0,000

Ejercicio 3.

En la documentación de la clase ArrayList que se encuentra en el siguiente link https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html, se encuentran las siguientes afirmaciones:

- *"The size, isEmpty, get, set, iterator, and listIterator operations run in constant time."*
- *"All of the other operations run in linear time (roughly speaking)."*

Explicar por qué se cree que algunas operaciones se ejecutan en tiempo constante y otras en tiempo lineal.

Las operaciones como *size*, *isEmpty*, *get*, *set*, *Iterator* y *listIterator* se ejecutan en tiempo constante porque acceden a posiciones de memoria o propiedades simples. En cambio, el resto de las operaciones se ejecutan en tiempo lineal porque requieren mover o procesar varios elementos.

Ejercicio 4.

Determinar si las siguientes sentencias son verdaderas o falsas, justificando la respuesta utilizando notación Big-Oh.

(a) 3^n *es de O(*2^n*).*

¿Existen constantes c $>$ 0 y $n_0 > 0$ tales que f (n) $\leq$ c g (n) para todo n $\geq n_0$?

$3^n \leq c2^n$
$\frac{3^n}{2^n} \leq c$
$(\frac{3}{2})^n \leq c.$

La sentencia es FALSA, ya que no existen c $>$ 0 y $n_0 > 0$ tales que f (n) $\leq$ c g (n) para todo n $\geq n_0$.

(b) *n +* log_2 *(n) es de O(n).*

¿Existen constantes c $>$ 0 y $n_0 > 0$ tales que f (n) $\leq$ c g (n) para todo n $\geq n_0$?

n + log_2 (n) $\leq$ cn
log_2 (n) $\leq$ cn - n
log_2 (n) $\leq$ n (c - 1)
log_2 (n) $\leq$ n (2 - 1)
log_2 (n) $\leq$ n * 1
log_2 (n) $\leq$ n
n $\geq$ 1.

La sentencia es VERDADERA, ya que existen c= 2 y n_0= 1 tales que f (n) $\leq$ c g (n) para todo n $\geq n_0$.

(c) $n^{\frac{1}{2}} + 10^{20}$ *es de O(*$n^{\frac{1}{2}}$*).*

¿Existen constantes c $>$ 0 y $n_0 > 0$ tales que f (n) $\leq$ c g (n) para todo n $\geq n_0$?

$n^{\frac{1}{2}} + 10^{20} \leq cn^{\frac{1}{2}}$
$10^{20} \leq cn^{\frac{1}{2}} - n^{\frac{1}{2}}$
$10^{20} \leq n^{\frac{1}{2}}$ (c - 1)
$10^{20} \leq n^{\frac{1}{2}}$ (2 - 1)
$10^{20} \leq n^{\frac{1}{2}}$ * 1
$10^{20} \leq n^{\frac{1}{2}}$

$n \geq (10^{20})^2$
$n \geq 10^{40}$.

La sentencia es VERDADERA, ya que existen c= 2 y $n_0 = 10^{40}$ tales que f (n) ≤ c g (n) para todo $n \geq n_0$.

(d) $\begin{cases} 3n + 17, n < 100 \\ 317, n \geq 100 \end{cases}$ *tiene orden lineal.*

La sentencia es VERDADERA, ya que el orden de mayor grado es O(n).

(e) *Mostrar que p(n)= $3n^5 + 8n^4 + 2n + 1$ es O(n^5).*

¿Existen constantes $c > 0$ y $n_0 > 0$ tales que p (n) ≤ c g (n) para todo $n \geq n_0$?

$3n^5 + 8n^4 + 2n + 1 \leq cn^5$.

Primer término:

$3n^5 \leq cn^5$
$3n^5 \leq 3n^5$.

$c_1 = 3$; $n_1 = 1$.

Segundo término:

$8n^4 \leq cn^5$
$8n^4 \leq 8n^5$.

$c_2 = 8$; $n_2 = 1$.

Tercer término:

$2n \leq cn^5$
$2n \leq 2n^5$.

$c_3 = 2$; $n_3 = 1$.

Cuarto término:

$1 \leq cn^5$
$1 \leq 1n^5$
$1 \leq n^5$.

$c_4 = 1$; $n_4 = 1$.

Entonces:

$3n^5 + 8n^4 + 2n + 1 \leq c_1 n^5 + c_2 n^5 + c_3 n^5 + c_4 n^5$
$3n^5 + 8n^4 + 2n + 1 \leq (c_1 + c_2 + c_3 + c_4)\, n^5$
$3n^5 + 8n^4 + 2n + 1 \leq (3 + 8 + 2 + 1)\, n^5$
$3n^5 + 8n^4 + 2n + 1 \leq 14n^5$, con c= 14 para todo n $\geq n_0$, con n_0= 1.

Por lo tanto, p (n) es O(n^5), ya que existen constantes c= 14 y n_0= 1 tales que p (n) ≤ c g (n) para todo n ≥ n_0.

(f) *Si p(n) es un polinomio de grado k, entonces, p(n) es O(n^k).*

¿Existen constantes c > 0 y $n_0 > 0$ tales que p (n) ≤ c g (n) para todo n ≥ n_0?

La sentencia es VERDADERA, ya que el término de mayor grado k domina a medida que n ⟶ ∞, siendo los términos de orden inferior (n^{k-1}, n^{k-2}, …) despreciables frente a n^k.

Ejercicio 5.

Se necesita generar una permutación random de los n primeros números enteros. Por ejemplo, [4, 3, 1, 0, 2] es una permutación legal, pero [0, 4, 1, 2, 4] no lo es, porque un número está duplicado (el 4) y otro no está (el 3). Se presentan tres algoritmos para solucionar este problema. Se asume la existencia de un generador de números random, ran_int (i, j), el cual genera, en tiempo constante, enteros entre i y j inclusive con igual probabilidad (esto significa que puede retornar el mismo valor más de una vez). También se supone el mensaje swap() que intercambia dos datos entre sí.

```
public class EjercicioPermutaciones {
	private static Random rand = new Random();

	public static int[] randomUno(int n) {
		int i, x = 0, k;
		int[] a = new int[n];
		for (i = 0; i < n; i++) {
			boolean seguirBuscando = true;
			while (seguirBuscando) {
				x = ran_int(0, n - 1);
				seguirBuscando = false;
				for (k = 0; k < i && !seguirBuscando; k++)
					if (x == a[k])
						seguirBuscando = true;
			}
			a[i] = x;
		}
		return a;
	}

	public static int[] randomDos(int n) {
		int i, x;
		int[] a = new int[n];
		boolean[] used = new boolean[n];
		for (i = 0; i < n; i++) used[i] = false;
		for (i = 0; i < n; i++) {
			x = ran_int(0, n - 1);
			while (used[x]) x = ran_int(0, n - 1);
			a[i] = x;
			used[x] = true;
		}
		return a;
	}
```

```
	public static int[] randomTres(int n) {
		int i;
		int[] a = new int[n];
		for (i = 0; i < n; i++) a[i] = i;
		for (i = 1; i < n; i++) swap(a, i, ran_int(0, i - 1));
		return a;
	}


	private static void swap(int[] a, int i, int j) {
		int aux;
		aux = a[i]; a[i] = a[j]; a[j] = aux;
	}

	/** Genera en tiempo constante, enteros entre i y j con igual probabilidad.
	 */
	private static int ran_int(int a, int b) {
		if (b < a || a < 0 || b < 0) throw new IllegalArgumentException("Parametros
invalidos");
		return a + (rand.nextInt(b - a + 1));
	}

	public static void main(String[] args) {
		System.out.println(Arrays.toString(randomUno(1000)));
		System.out.println(Arrays.toString(randomDos(1000)));
		System.out.println(Arrays.toString(randomTres(1000)));
	}
}
```

(a) *Analizar si todos los algoritmos terminan o alguno puede quedar en loop infinito.*

Los primeros dos algoritmos podrían quedar en un *loop* infinito sólo cuando el generador de números aleatorios retorna siempre una secuencia de números repetidos. Ambos algoritmos están diseñados para que terminen una vez hallada una permutación legal y, en caso contrario, sigan iterando hasta hallarla.

El último de los algoritmos, en caso de que el generador de números aleatorios retorne siempre una secuencia de números repetidos, la posición intercambiada sería la misma, pero no generaría un *loop* infinito.

(b) *Describir, con palabras, la cantidad de operaciones que realizan.*

- *randomUno*: Para cada valor, compara con los anteriores hasta encontrar uno no repetido (con posibles repeticiones).
- *randomDos*: Usa *used[]* para evitar repeticiones y sólo realiza reintentos si hay colisiones.
- *randomTres*: Llena el arreglo y hace un intercambio aleatorio por posición.

Ejercicio 6.

(a) *Se supone que se tiene un algoritmo de O(log^2 n) y se dispone de 1 hora de uso de CPU. En esa hora, la CPU puede ejecutar el algoritmo con una entrada de tamaño n= 1.024 como máximo. ¿Cuál sería el mayor tamaño de entrada que podría ejecutar el algoritmo si se dispone de 4 horas de CPU?*

En 1 hora:

$(log_2\ n)^2 = (log_2\ 1024)^2$
$(log_2\ n)^2 = 10^2$
$(log_2\ n)^2 = 100$.

En 4 horas:

$(log_2\ n)^2 = 4 * 100$
$(log_2\ n)^2 = 400$
$(log_2\ n)^2 = 20^2$
log_2 n= 20
n= 2^{20}.

Por lo tanto, el mayor tamaño de entrada que podría ejecutar el algoritmo si se dispone de 4 horas de CPU es 2^{20}.

(b) *Considerando que un algoritmo requiere T(n) operaciones para resolver un problema y la computadora procesa 10.000 operaciones por segundo. Si T(n)= n^2, determinar el tiempo en segundos requerido por el algoritmo para resolver un problema de tamaño n= 2.000.*

T(n)= n^2.

T(2000)= 2000^2
T(2000)= 4000000.

10.000 operaciones por segundo.

Tiempo (en segundos)= $\frac{4000000}{10000}$
Tiempo (en segundos)= 400.

Por lo tanto, el tiempo en segundos requerido por el algoritmo para resolver un problema de tamaño n= 2.000 es 400.

Ejercicio 7.

Para cada uno de los siguientes fragmentos de código, calcular, intuitivamente, el orden del tiempo de ejecución.

`for(int i = 0; i< n; i++)` `  sum++;`	`for(int i = 0; i< n; i+=2)` `  sum++;`

`for(int i = 0; i< n; i++)`
`  for(int j = 0; j< n; j++)`
`  sum++;`
	`for(int i = 0; i < n + 100; ++i) {`
	`  for(int j = 0; j < i * n ; ++j){`
	`  sum = sum + j;`
	`  }`
	`  for(int k = 0; k < n + n + n; ++k){`
	`  c[k] = c[k] + sum;`
	`  }`
	`}`

`for(int i = 0; i< n; i++)`
`  for(int j = 0; j< n; j++)`
`  sum++;`
`for(int i = 0; i< n; i++)`
`  sum++;`
	`int i,j;`
	`int x = 1;`
	`for (i = 0; i <= n²; i=i+2)`
	`  for (j = n; j >= 1; j-= n/4)`
	`  x++;`

(a)

T(n)= $\sum_{i=1}^{n} cte$
T(n)= n cte.

Por lo tanto, para este fragmento de código, el orden de tiempo de ejecución es O(n).

(b)

T(n)= $\sum_{i=1}^{\frac{n}{2}} cte$
T(n)= $\frac{1}{2}$ n cte.

Por lo tanto, para este fragmento de código, el orden de tiempo de ejecución es O(n).

(c)

T(n)= $\sum_{i=1}^{n} \sum_{j=1}^{n} cte$
T(n)= $\sum_{i=1}^{n} n * cte$
T(n)= nn cte
T(n)= n^2 cte.

Por lo tanto, para este fragmento de código, el orden de tiempo de ejecución es O(n^2).

(d)

T(n)= $\sum_{i=1}^{n+100}(\sum_{j=1}^{in} cte_1 + \sum_{k=1}^{3n} cte_2)$
T(n)= $\sum_{i=1}^{n+100}(in * cte_1 + 3n * cte_2)$
T(n)= $\sum_{i=1}^{n+100} in * cte_1 + \sum_{i=1}^{n+100} 3n * cte_2$
T(n)= n cte_1 $\sum_{i=1}^{n+100} i$ + (n + 100) 3n cte_2
T(n)= n cte_1 $\frac{(n+100)(n+100+1)}{2}$ + (n + 100) 3n cte_2
T(n)= n cte_1 $\frac{(n+100)(n+101)}{2}$ + (n + 100) 3n cte_2
T(n)= $\frac{(n+100)(n+101)n}{2}$ cte_1 + ($3n^2$ + 300n) cte_2
T(n)= $\frac{n^3+101n^2+100n^2+10100n}{2}$ cte_1 + ($3n^2$ + 300n) cte_2
T(n)= $\frac{n^3+201n^2+10100n}{2}$ cte_1 + ($3n^2$ + 300n) cte_2
T(n)= ($\frac{1}{2}n^3 + \frac{201}{2}n^2$ + 5050n) cte_1 + ($3n^2$ + 300n) cte_2.

Por lo tanto, para este fragmento de código, el orden de tiempo de ejecución es O(n^3).

(e)

T(n)= $\sum_{i=1}^{n}\sum_{j=1}^{n} cte_1 + \sum_{i=1}^{n} cte_2$
T(n)= $\sum_{i=1}^{n} n * cte_1$ + n cte_2
T(n)= nn cte_1 + n cte_2
T(n)= n^2 cte_1 + n cte_2.

Por lo tanto, para este fragmento de código, el orden de tiempo de ejecución es O(n^2).

(f)

T(n)= $cte_1 + \sum_{i=1}^{\frac{n^2}{2}}\sum_{j=1}^{4} cte_2$
T(n)= $cte_1 + \sum_{i=1}^{\frac{n^2}{2}} 4cte_2$
T(n)= $cte_1 + \frac{n^2}{2} * 4cte_2$
T(n)= $cte_1 + 2n^2$ cte_2.

Por lo tanto, para este fragmento de código, el orden de tiempo de ejecución es O(n^2).

Ejercicio 8.

Para cada uno de los algoritmos presentados, calcular el T(n).

(a) *Expresar, en función de n, el tiempo de ejecución.*

(b) *Establecer el orden de dicha función usando notación Big-Oh.*

```
1. int c = 1;
   while ( c < n ) {
      algo_de_O(1);
      c = 2 * c;
   }
```

```
2. int c = n;
   while ( c > 1 ) {
     algo_de_O(1);
     c = c / 2;
   }
```

```
3. public static void calcular(int n) {
        int i, j, r = 0;
        for ( i = 1; i < n; i= i+2)
           for (j = 1; j <= i; j++ )
                 r = r + 1;
        return r;
   }

  }
```

(1)

Iteraciones del *while*:

c= 1.
c= 2.
c= 4.
…
c= 2^{k-1}.

2^{k-1}= n - 1
$log_2\ 2^{k-1}$= log_2 (n - 1)
(k - 1) log_2 2= log_2 (n - 1)
(k - 1) * 1= log_2 (n - 1)
k - 1= log_2 (n - 1)
k= log_2 (n - 1) + 1.

Entonces:

T(n)= $cte_1 + \sum_{c=1}^{log_2(n-1)+1} cte_2$
T(n)= cte_1 + [log_2 (n - 1) + 1] $cte_2 \leq$ O(log_2 n).

¿Existen constantes c $>$ 0 y $n_0 > 0$ tales que f (n) $\leq$ c g (n) para todo n $\geq n_0$?

cte_1 + [log_2 (n - 1) + 1] $cte_2 \leq$ c log_2 n.

Primer término:

$cte_1 \leq$ c log_2 n
$cte_1 \leq cte_1\ log_2$ n.

c_1= cte_1; n_0= 2.

Segundo término:

cte_2 [log_2 (n - 1) + 1] $\leq$ c log_2 n
cte_2 [log_2 (n - 1) + 1] $\leq cte_2\ log_2$ n.

c_2= cte_2; n_0= 2.

Entonces:

cte_1 + [log_2 (n - 1) + 1] $cte_2 \leq c_1\ log_2$ n + $c_2\ log_2$ n
cte_1 + [log_2 (n - 1) + 1] $cte_2 \leq (c_1 + c_2)\ log_2$ n
cte_1 + [log_2 (n - 1) + 1] $cte_2 \leq (cte_1 + cte_2)\ log_2$ n, con c= cte_1 + cte_2 para todo n $\geq$ n_0, con n_0= 2.

Por lo tanto, T(n) es O(log_2 n), ya que existen constantes c= cte_1 + cte_2 y n_0= 2 tales que f (n) $\leq$ c g (n) para todo n $\geq$ n_0.

(2)

Iteraciones del *while*:

c= n.
c= $\frac{n}{2}$.
c= $\frac{n}{4}$.
…
c= $\frac{n}{2^{k-1}}$.

$\frac{n}{2^{k-1}}$= 1 + 1
$\frac{n}{2^{k-1}}$= 2
2 * 2^{k-1}= n
2^k= n
$log_2\ 2^k = log_2$ n
k log_2 2= log_2 n
k * 1= log_2 n
k= log_2 n.

Entonces:

T(n)= $cte_1 + \sum_{c=1}^{log_2 n} cte_2$
T(n)= $cte_1 + log_2$ n * $cte_2 \leq$ O(log_2 n).

¿Existen constantes c > 0 y $n_0 > 0$ tales que f (n) ≤ c g (n) para todo n ≥ n_0?

Primer término:

$cte_1 \leq$ c log_2 n
$cte_1 \leq cte_1\ log_2$ n.

c_1= cte_1; n_0= 2.

Segundo término:

$cte_2\ log_2$ n ≤ c log_2 n
$cte_2\ log_2$ n ≤ $cte_2\ log_2$ n.

c_2= cte_2; n_0= 1.

Entonces:

$cte_1 + log_2$ n * $cte_2 \leq c_1\ log_2$ n + $c_2\ log_2$ n
$cte_1 + log_2$ n * $cte_2 \leq (c_1 + c_2)\ log_2$ n
$cte_1 + log_2$ n * $cte_2 \leq (cte_1 + cte_2)\ log_2$ n, con c= $cte_1 + cte_2$ para todo n ≥ n_0, con n_0= 2.

Por lo tanto, T(n) es O(log_2 n), ya que existen constantes c= $cte_1 + cte_2$ y n_0= 2 tales que f (n) ≤ c g (n) para todo n ≥ n_0.

(3)

Iteraciones del primer *for*:

i= 1.
i= 3.
i= 5.
…
i= 2k - 1.

2k - 1= n - 1
2k= n - 1 + 1
2k= n
k= $\frac{n}{2}$.

Iteraciones del segundo *for*:

j= 1.
j= 2.
j= 3.
…
j= k.

k= i.

Entonces:

T(n)= $cte_1 + \sum_{i=1}^{\frac{n}{2}} \sum_{j=1}^{i} cte_2$
T(n)= $cte_1 + \sum_{i=1}^{\frac{n}{2}} i * cte_2$
T(n)= $cte_1 + cte_2 \sum_{i=1}^{\frac{n}{2}} i$
T(n)= $cte_1 + cte_2 \frac{\frac{n}{2}(\frac{n}{2}+1)}{2}$
T(n)= $cte_1 + \frac{\frac{n^2}{4}+\frac{n}{2}}{2} cte_2$
T(n)= $cte_1 + (\frac{1}{8} n^2 + \frac{1}{4}$ n) $cte_2 \leq$ O(n^2).

¿Existen constantes c $>$ 0 y $n_0 > 0$ tales que f (n) ≤ c g (n) para todo n ≥ n_0?

Primer término:

$cte_1 \leq cn^2$
$cte_1 \leq cte_1 n^2$.

c_1= cte_1; n_0= 1.

Segundo término:

cte_2 ($\frac{1}{8} n^2 + \frac{1}{4}$ n) ≤ cn^2
cte_2 ($\frac{1}{8} n^2 + \frac{1}{4}$ n) ≤ $cte_2 n^2$.

c_2= cte_2; n_0= 1.

Entonces:

$cte_1 + (\frac{1}{8} n^2 + \frac{1}{4}$ n) $cte_2 \leq c_1 n^2 + c_2 n^2$
$cte_1 + (\frac{1}{8} n^2 + \frac{1}{4}$ n) $cte_2 \leq (c_1 + c_2)\, n^2$
$cte_1 + (\frac{1}{8} n^2 + \frac{1}{4}$ n) $cte_2 \leq (cte_1 + cte_2)\, n^2$, con c= $cte_1 + cte_2$ para todo n ≥ n_0, con n_0= 1.

Por lo tanto, T(n) es O(n^2), ya que existen constantes c= $cte_1 + cte_2$ y n_0= 1 tales que f (n) ≤ c g (n) para todo n ≥ n_0.

Ejercicio 9.

(a) *Expresar la función del tiempo de ejecución de cada uno de los siguientes algoritmos, resolverla y calcular el orden.*

```
static public int rec2(int n){
	if (n <= 1)
		return 1;
	else
		return (2 * rec2(n-1));
}

static public int rec1(int n){
	if (n <= 1)
		return 1;
	else
		return (rec1(n-1) + rec1(n-1));
}

static public int rec3(int n){
	if  ( n == 0 )
		return  0;
	else {
		if ( n == 1 )
			return 1;
		else
			return (rec3(n-2) * rec3(n-2));
	}
}
static public int potencia_iter(int x, int n){
	int potencia;
	if  (n == 0)
		potencia = 1;
	else {
		if  (n == 1)
			potencia  =  x;
		else{
			potencia  = x;
			for (int  i  = 2 ; i <=  n ; i++) {
				potencia *= x ;
			}
		}
	}
	return  potencia;
}
```

```
static public int potencia_rec( int x, int n){
        if( n == 0 )
                return 1;
        else{
                if( n == 1)
                        return x;
                else{
                        if ( (n  %  2 ) == 0)
                                return  potencia_rec (x * x, n / 2 );
                        else
                                return potencia_rec (x * x, n / 2) * x;
                }
        }
}
```

Algoritmo *rec2*:

T(n)= $\begin{cases} cte_1, n \leq 1 \\ T(n-1) + cte_2, n > 1 \end{cases}$.

Paso 1:

T(n)= T(n - 1) + cte_2, si n > 1.

Paso 2:

T(n)= T(n - 1 - 1) + cte_2 + cte_2
T(n)= T(n - 2) + $2cte_2$, si n > 2.

Paso 3:

T(n)= T(n - 2 - 1) + cte_2 + $2cte_2$
T(n)= T(n - 3) + $3cte_2$, si n > 3.

Paso i (Paso general):

T(n)= T(n - i) + icte_2, si n > i.

n - i= 1
i= n - 1.

Entonces:

T(n)= T(n - (n - 1)) + (n - 1) cte_2
T(n)= T(n - n + 1) + (n - 1) cte_2
T(n)= T(1) + (n - 1) cte_2
T(n)= cte_1 + (n - 1) cte_2 ≤ O(n).

¿Existen constantes c > 0 y $n_0 > 0$ tales que f (n) ≤ c g (n) para todo n ≥ n_0?

Primer término:

$cte_1 \leq$ cn
$cte_1 \leq cte_1$n.

$c_1 = cte_1$; $n_0 = 1$.

Segundo término:

cte_2 (n - 1) ≤ cn
cte_2 (n - 1) ≤ cte_2n.

$c_2 = cte_2$; $n_0 = 1$.

Entonces:

cte_1 + (n - 1) $cte_2 \leq c_1$n + c_2n
cte_1 + (n - 1) $cte_2 \leq (c_1 + c_2)$ n
cte_1 + (n - 1) $cte_2 \leq (cte_1 + cte_2)$ n, con c= $cte_1 + cte_2$ para todo n ≥ n_0, con $n_0 = 1$.

Por lo tanto, T(n) es O(n), ya que existen constantes c= $cte_1 + cte_2$ y $n_0 = 1$ tales que f (n) ≤ c g (n) para todo n ≥ n_0.

Por lo tanto, la función de tiempo de ejecución es T(n)= cte_1 + (n - 1) cte_2 y el orden es O(n).

Algoritmo *rec1*:

$$T(n) = \begin{cases} cte_1, n \leq 1 \\ 2\,T(n-1) + cte_2, n > 1 \end{cases}.$$

Paso 1:

T(n)= 2 T(n - 1) + cte_2, si n > 1.

Paso 2:

T(n)= 2 [2 T(n - 1 - 1) + cte_2] + cte_2
T(n)= 2 [2 T(n - 2) + cte_2] + cte_2
T(n)= 4 T(n - 2) + 2cte_2 + cte_2
T(n)= 4 T(n - 2) + 3cte_2, si n > 2.

Paso 3:

T(n)= 4 [2 T(n - 2 - 1) + cte_2] + 3cte_2
T(n)= 4 [2 T(n - 3) + cte_2] + 3cte_2
T(n)= 8 T(n - 3) + 4cte_2 + 3cte_2
T(n)= 8 T(n - 3) + 7cte_2, si n > 3.

Paso i (Paso general):

T(n)= 2^i T(n - i) + (2^i - 1) cte_2, si n > i.

n - i= 1
i= n - 1.

Entonces:

T(n)= 2^{n-1} T(n - (n - 1)) + (2^{n-1} - 1) cte_2
T(n)= 2^{n-1} T(n - n + 1) + (2^{n-1} - 1) cte_2
T(n)= 2^{n-1} T(1) + (2^{n-1} - 1) cte_2
T(n)= 2^{n-1} cte_1 + (2^{n-1} - 1) cte_2 ≤ O(2^n).

¿Existen constantes c > 0 y n_0 > 0 tales que f (n) ≤ c g (n) para todo n ≥ n_0?

Primer término:

cte_1 2^{n-1} ≤ c 2^n
cte_1 2^{n-1} ≤ cte_1 2^n.

c_1= cte_1; n_0= 1.

Segundo término:

cte_2 (2^{n-1} - 1) ≤ c 2^n
cte_2 (2^{n-1} - 1) ≤ cte_2 2^n.

c_2= cte_2; n_0= 1.

Entonces:

2^{n-1} cte_1 + (2^{n-1} - 1) cte_2 ≤ c_1 2^n + c_2 2^n
2^{n-1} cte_1 + (2^{n-1} - 1) cte_2 ≤ (c_1 + c_2) 2^n
2^{n-1} cte_1 + (2^{n-1} - 1) cte_2 ≤ (cte_1 + cte_2) 2^n, con c= cte_1 + cte_2 para todo n ≥ n_0, con n_0= 1.

Por lo tanto, T(n) es O(2^n), ya que existen constantes c= cte_1 + cte_2 y n_0= 1 tales que f (n) ≤ c g (n) para todo n ≥ n_0.

Por lo tanto, la función de tiempo de ejecución es T(n)= 2^{n-1} cte_1 + (2^{n-1} - 1) cte_2 y el orden es O(2^n).

Algoritmo *rec3*:

T(n)= $\begin{cases} cte_1, n \leq 1 \\ 2\,T(n-2) + cte_2, n > 1 \end{cases}$.

Paso 1:

T(n)= 2 T(n - 2) + cte_2, si n > 1.

Paso 2:

T(n)= 2 [2 T(n - 2 - 2) + cte_2] + cte_2
T(n)= 2 [2 T(n - 4) + cte_2] + cte_2
T(n)= 4 T(n - 4) + 2cte_2 + cte_2
T(n)= 4 T(n - 4) + 3cte_2, si n > 2.

Paso 3:

T(n)= 4 [2 T(n - 4 - 2) + cte_2] + 3cte_2
T(n)= 4 [2 T(n - 6) + cte_2] + 3cte_2
T(n)= 8 T(n - 6) + 4cte_2 + 3cte_2
T(n)= 8 T(n - 6) + 7cte_2, si n > 3.

Paso i (Paso general):

T(n)= 2^i T(n - 2i) + (2^i - 1) cte_2, si n > i.

n - 2i= 1
2i= n - 1
i= $\frac{n-1}{2}$.

Entonces:

T(n)= $2^{\frac{n-1}{2}}$ T(n - 2 $\frac{n-1}{2}$) + ($2^{\frac{n-1}{2}}$ - 1) cte_2
T(n)= $2^{\frac{n-1}{2}}$ T(n - (n - 1)) + ($2^{\frac{n-1}{2}}$ - 1) cte_2
T(n)= $2^{\frac{n-1}{2}}$ T(n - n + 1) + ($2^{\frac{n-1}{2}}$ - 1) cte_2
T(n)= $2^{\frac{n-1}{2}}$ T(1) + ($2^{\frac{n-1}{2}}$ - 1) cte_2
T(n)= $2^{\frac{n-1}{2}}$ cte_1 + ($2^{\frac{n-1}{2}}$ - 1) cte_2 ≤ O($\sqrt{2^n}$).

¿Existen constantes c > 0 y n_0 > 0 tales que f (n) ≤ c g (n) para todo n ≥ n_0?

Primer término:

cte_1 $2^{\frac{n-1}{2}}$ ≤ c $\sqrt{2^n}$
cte_1 $2^{\frac{n-1}{2}}$ ≤ cte_1 $\sqrt{2^n}$.

c_1= cte_1; n_0= 1.

Segundo término:

cte_2 ($2^{\frac{n-1}{2}}$ - 1) ≤ c $\sqrt{2^n}$
cte_2 ($2^{\frac{n-1}{2}}$ - 1) ≤ cte_2 $\sqrt{2^n}$.

c_2= cte_2; n_0= 1.

Entonces:

$2^{\frac{n-1}{2}}\ cte_1 + (2^{\frac{n-1}{2}} - 1)\ cte_2 \leq c_1\ 2^{\frac{n-1}{2}} + c_2\ 2^{\frac{n-1}{2}}$
$2^{\frac{n-1}{2}}\ cte_1 + (2^{n-1} - 1)\ cte_2 \leq (c_1 + c_2)\ 2^{\frac{n-1}{2}}$
$2^{\frac{n-1}{2}}\ cte_1 + (2^{\frac{n-1}{2}} - 1)\ cte_2 \leq (cte_1 + cte_2)\ 2^{\frac{n-1}{2}}$, con c= $cte_1 + cte_2$ para todo n $\geq n_0$, con n_0= 1.

Por lo tanto, T(n) es O($\sqrt{2^n}$), ya que existen constantes c= $cte_1 + cte_2$ y n_0= 1 tales que f (n) ≤ c g (n) para todo n ≥ n_0.

Por lo tanto, la función de tiempo de ejecución es T(n)= $2^{\frac{n-1}{2}}\ cte_1 + (2^{\frac{n-1}{2}} - 1)\ cte_2$ y el orden es O($\sqrt{2^n}$).

Algoritmo *potencia_iter*:

T(n)= $cte_1 + \sum_{i=2}^{n} cte_3$
T(n)= cte_1 + (n - 1) cte_2 ≤ O(n).

¿Existen constantes c > 0 y $n_0 > 0$ tales que f (n) ≤ c g (n) para todo n ≥ n_0?

Primer término:

$cte_1 \leq$ cn
$cte_1 \leq cte_1$n.

c_1= cte_1; n_0= 1.

Segundo término:

cte_2 (n - 1) ≤ cn
cte_2 (n - 1) ≤ cte_2n.

c_2= cte_2; n_0= 1.

Entonces:

cte_1 + (n - 1) $cte_2 \leq c_1$n + c_2n
cte_1 + (n - 1) $cte_2 \leq (c_1 + c_2)$ n
cte_1 + (n - 1) $cte_2 \leq (cte_1 + cte_2)$ n, con c= $cte_1 + cte_2$ para todo n ≥ n_0, con n_0= 1.

Por lo tanto, T(n) es O(n), ya que existen constantes c= $cte_1 + cte_2$ y n_0= 1 tales que f (n) ≤ c g (n) para todo n ≥ n_0.

Por lo tanto, la función de tiempo de ejecución es T(n)= cte_1 + (n - 1) cte_2 y el orden es O(n).

Algoritmo *potencia_rec*:

$T(n)= \begin{cases} cte_1, n \leq 1 \\ T(\frac{n}{2}) + cte_2, n > 1 \end{cases}$.

Paso 1:

T(n)= T($\frac{n}{2}$) + cte_2, si n > 1.

Paso 2:

T(n)= T($\frac{\frac{n}{2}}{2}$) + cte_2 + cte_2
T(n)= T($\frac{n}{4}$) + 2cte_2, si n > 2.

Paso 3:

T(n)= T($\frac{\frac{n}{2}}{4}$) + cte_2 + 2cte_2
T(n)= T($\frac{n}{8}$) + 3cte_2, si n > 3.

Paso i (Paso general):

T(n)= T($\frac{n}{2^i}$) + icte_2, si n > i.

$\frac{n}{2^i}$= 1
1 * 2^i= n
2^i= n
i log_2 2= log_2 n
i * 1= log_2 n
i= log_2 n.

Entonces:

T(n)= T($\frac{n}{2^i}$) + icte_2
T(n)= T($\frac{n}{2^{log_2 n}}$) + log_2 n * cte_2
T(n)= T($\frac{n}{n}$) + log_2 n * cte_2
T(n)= T(1) + log_2 n * cte_2
T(n)= cte_1 + log_2 n * cte_2 ≤ O(log_2 n).

¿Existen constantes c > 0 y n_0 > 0 tales que f (n) ≤ c g (n) para todo n ≥ n_0?

Primer término:

cte_1 ≤ c log_2 n
cte_1 ≤ cte_1 log_2 n.

c_1= cte_1; n_0= 2.

Segundo término:

$cte_2\ log_2$ n ≤ c log_2 n
$cte_2\ log_2$ n ≤ $cte_2\ log_2$ n.

c_2= cte_2; n_0= 1.

Entonces:

cte_1 + log_2 n * cte_2 ≤ $c_1\ log_2$ n + $c_2\ log_2$ n
cte_1 + log_2 n * cte_2 ≤ (c_1 + c_2) log_2 n
cte_1 + log_2 n * cte_2 ≤ (cte_1 + cte_2) log_2 n, con c= cte_1 + cte_2 para todo n ≥ n_0, con n_0= 1.

Por lo tanto, T(n) es O(log_2 n), ya que existen constantes c= cte_1 + cte_2 y n_0= 2 tales que f (n) ≤ c g (n) para todo n ≥ n_0.

Por lo tanto, la función de tiempo de ejecución es T(n)= cte_1 + log_2 n * cte_2 y el orden es O(log_2 n).

(b) *Comparar el tiempo de ejecución del método "rec2" con el del método "rec1".*

El tiempo de ejecución del método "*rec2*" es menor al del método "*rec1*".

(c) *Implementar un algoritmo más eficiente que el del método "rec3" (es decir, que el T(n) sea menor).*

```
static public int rec3Mejorado(int n) {
        return n%2
}
```

Este algoritmo es más eficiente que el del método "*rec3*", ya que el orden de tiempo de ejecución es O(1).

Ejercicio 10.

(a) *Resolver las siguientes recurrencias.*

(b) *Calcular el O(n). Justificar usando la definición de Big-Oh.*

1.

$$T(n) = \begin{cases} 2, n = 1 \\ T(n-1) + n, n \geq 2 \end{cases}$$

2.

$$T(n) = \begin{cases} 1 & n = 1 \\ 2T\left(\frac{n}{4}\right) + \sqrt{n}, & n \geq 2 \end{cases}$$

2.

$$T(n) = \begin{cases} 2, n = 1 \\ T(n-1) + \frac{n}{2}, n \geq 2 \end{cases}$$

3.

$$T(n) = \begin{cases} 1, n = 1 \\ 4T\left(\frac{n}{2}\right) + n^2, n \geq 2 \end{cases}$$

(1) $T(n) = \begin{cases} 2, n = 1 \\ T(n-1) + n, n \geq 2 \end{cases}$.

Paso 1:

T(n)= T(n - 1) + n, si n ≥ 2.

Paso 2:

T(n)= T(n - 1 - 1) + (n - 1) + n
T(n)= T(n - 2) + (n - 1) + n, si n ≥ 3.

Paso 3:

T(n)= T(n - 2 - 1) + (n - 2) + (n - 1) + n
T(n)= T(n - 3) + (n - 2) + (n - 1) + n, si n ≥ 4.

Paso i (Paso general):

T(n)= T(n - i) + (n - i + 1) + (n - i + 2) + … + n
T(n)= T(n - i) + $\sum_{i=2}^{n} i$
T(n)= T(n - i) + $\sum_{i=1}^{n} i$ - 1
T(n)= T(n - i) + $\frac{n(n+1)}{2}$ - 1
T(n)= T(n - i) + $\frac{n^2+n}{2}$ - 1
T(n)= T(n - i) + $\frac{1}{2}n^2 + \frac{1}{2}$n - 1, si n ≥ i + 1.

n - i= 1
i= n - 1.

Entonces:

T(n)= T(n - (n - 1)) + $\frac{1}{2} n^2 + \frac{1}{2}$ n - 1
T(n)= T(n - n + 1) + $\frac{1}{2} n^2 + \frac{1}{2}$ n - 1
T(n)= T(1) + $\frac{1}{2} n^2 + \frac{1}{2}$ n - 1
T(n)= 2 + $\frac{1}{2} n^2 + \frac{1}{2}$ n - 1
T(n)= $\frac{1}{2} n^2 + \frac{1}{2}$ n + 1 ≤ O(n^2).

¿Existen constantes c > 0 y $n_0 > 0$ tales que f (n) ≤ c g (n) para todo n ≥ n_0?

Primer término:

$\frac{1}{2} n^2 \leq cn^2$
$\frac{1}{2} n^2 \leq \frac{1}{2} n^2$.

$c_1 = \frac{1}{2}$; $n_0 = 1$.

Segundo término:

$\frac{1}{2}$ n ≤ cn^2
$\frac{1}{2}$ n ≤ $\frac{1}{2} n^2$.

$c_2 = \frac{1}{2}$; $n_0 = 1$.

Tercer término:

1 ≤ cn^2
1 ≤ 1n^2
1 ≤ n^2.

$c_3 = 1$; $n_0 = 1$.

Entonces:

$\frac{1}{2} n^2 + \frac{1}{2}$ n + 1 ≤ $c_1 n^2 + c_2 n^2 + c_3 n^2$
$\frac{1}{2} n^2 + \frac{1}{2}$ n + 1 ≤ $(c_1 + c_2 + c_3)\, n^2$
$\frac{1}{2} n^2 + \frac{1}{2}$ n + 1 ≤ $(\frac{1}{2} + \frac{1}{2} + 1)\, n^2$
$\frac{1}{2} n^2 + \frac{1}{2}$ n + 1 ≤ $2n^2$, con c= 2 para todo n ≥ n_0, con $n_0 = 1$.

Por lo tanto, T(n) es O(n^2), ya que existen constantes c= 2 y $n_0 = 1$ tales que f (n) ≤ c g (n) para todo n ≥ n_0.

(2) $T(n) = \begin{cases} 2, n = 1 \\ T(n-1) + \frac{n}{2}, n \geq 2 \end{cases}$.

Paso 1:

T(n)= T(n - 1) + $\frac{n}{2}$, si n ≥ 2.

Paso 2:

T(n)= T(n - 1 - 1) + $\frac{n-1}{2} + \frac{n}{2}$
T(n)= T(n - 2) + $\frac{n-1}{2} + \frac{n}{2}$, si n ≥ 3.

Paso 3:

T(n)= T(n - 2 - 1) + $\frac{n-2}{2} + \frac{n-1}{2} + \frac{n}{2}$
T(n)= T(n - 3) + $\frac{n-2}{2} + \frac{n-1}{2} + \frac{n}{2}$, si n ≥ 4.

Paso i (Paso general):

T(n)= T(n - i) + $\frac{n-i+1}{2} + \frac{n-i+2}{2} + \ldots + \frac{n}{2}$
T(n)= T(n - i) + $\frac{1}{2}$ [(n - i + 1) + (n - i + 2) + … + n]
T(n)= T(n - i) + $\frac{1}{2} \sum_{i=2}^{n} i$
T(n)= T(n - i) + $\frac{1}{2}$ ($\sum_{i=1}^{n} i$ - 1)
T(n)= T(n - i) + $\frac{1}{2}$ [$\frac{n(n+1)}{2}$ - 1]
T(n)= T(n - i) + $\frac{1}{2}$ ($\frac{n^2+n}{2}$ - 1)
T(n)= T(n - i) + $\frac{1}{2}$ ($\frac{1}{2} n^2 + \frac{1}{2}$ n - 1)
T(n)= T(n - i) + $\frac{1}{4} n^2 + \frac{1}{4}$ n - $\frac{1}{2}$, si n ≥ i + 1.

n - i= 1
i= n - 1.

Entonces:

T(n)= T(n - (n - 1)) + $\frac{1}{4} n^2 + \frac{1}{4}$ n - $\frac{1}{2}$
T(n)= T(n - n + 1) + $\frac{1}{4} n^2 + \frac{1}{4}$ n - $\frac{1}{2}$
T(n)= T(1) + $\frac{1}{4} n^2 + \frac{1}{4}$ n - $\frac{1}{2}$
T(n)= 2 + $\frac{1}{4} n^2 + \frac{1}{4}$ n - $\frac{1}{2}$
T(n)= $\frac{1}{4} n^2 + \frac{1}{4}$ n + $\frac{3}{4}$ ≤ O(n^2).

¿Existen constantes c > 0 y $n_0 > 0$ tales que f (n) ≤ c g (n) para todo n ≥ n_0?

Primer término:

$\frac{1}{4} n^2 \leq cn^2$
$\frac{1}{4} n^2 \leq \frac{1}{4} n^2$.

$c_1 = \frac{1}{4}$; $n_0 = 1$.

Segundo término:

$\frac{1}{4} n \leq cn^2$
$\frac{1}{4} n \leq \frac{1}{4} n^2$.

$c_2 = \frac{1}{4}$; $n_0 = 1$.

Tercer término:

$\frac{3}{4} \leq cn^2$
$\frac{3}{4} \leq \frac{3}{4} n^2$.

$c_3 = \frac{3}{4}$; $n_0 = 1$.

Entonces:

$\frac{1}{4} n^2 + \frac{1}{4} n + \frac{3}{4} \leq c_1 n^2 + c_2 n^2 + c_3 n^2$
$\frac{1}{4} n^2 + \frac{1}{4} n + \frac{3}{4} \leq (c_1 + c_2 + c_3)\, n^2$
$\frac{1}{4} n^2 + \frac{1}{4} n + \frac{3}{4} \leq (\frac{1}{4} + \frac{1}{4} + \frac{3}{4})\, n^2$
$\frac{1}{4} n^2 + \frac{1}{4} n + \frac{3}{4} \leq \frac{5}{4} n^2$, con c= $\frac{5}{4}$ para todo n $\geq n_0$, con $n_0 = 1$.

Por lo tanto, T(n) es O(n^2), ya que existen constantes c= $\frac{5}{4}$ y $n_0 = 1$ tales que f (n) ≤ c g (n) para todo n ≥ n_0.

(3) *T(n)*= $\begin{cases} 1, n = 1 \\ 2\, T(\frac{n}{4}) + \sqrt{n}, n \geq 2 \end{cases}$.

(4) *T(n)*= $\begin{cases} 1, n = 1 \\ 4\, T(\frac{n}{2}) + n^2, n \geq 2 \end{cases}$.

Ejercicio 11.

Calcular el tiempo de ejecución de los métodos buscarLineal y buscarDicotomica de la clase BuscadorEnArrayOrdenado. Comparar el tiempo con los valores obtenidos, empíricamente, en el Ejercicio 2.

***buscarLineal*:**

T(n)= $cte_1 + \sum_{i=1}^{n} cte_2$
T(n)= cte_1 + n $cte_2 \leq$ O(n).

***buscarDicotomica*:**

T(n)= $\begin{cases} cte_1, n \leq 1 \\ T(\frac{n}{2}) + cte_2, n > 1 \end{cases}$.

Paso 1:

T(n)= T($\frac{n}{2}$) + cte_2, si n > 1.

Paso 2:

T(n)= T($\frac{\frac{n}{2}}{2}$) + cte_2 + cte_2
T(n)= T($\frac{n}{4}$) + 2cte_2, si n > 2.

Paso 3:

T(n)= T($\frac{\frac{n}{4}}{2}$) + cte_2 + 2cte_2
T(n)= T($\frac{n}{8}$) + 3cte_2, si n > 3.

Paso i (Paso general):

T(n)= T($\frac{n}{2^i}$) + icte_2, si n > i.

$\frac{n}{2^i}$= 1
1 * 2^i= n
2^i= n
log_2 2^i= log_2 n
i log_2 2= log_2 n
i * 1= log_2 n
i= log_2 n.

Entonces:

T(n)= T($\frac{n}{2^{log_2 n}}$) + log_2 n * cte_2
T(n)= T($\frac{n}{n}$) + log_2 n * cte_2
T(n)= T(1) + log_2 n * cte_2
T(n)= cte_1 + log_2 n * cte_2 ≤ O(log_2 n).

Ejercicio 12.

Calcular el tiempo de ejecución de procesarMovimientos y procesarMovimientosOptimizado del Ejercicio 1. Comparar el tiempo con los valores obtenidos empíricamente.

procesarMovimientos:

T(n)= $cte_1 + \sum_{i=1}^{n}(cte_2 + \sum_{i=1}^{n+1} cte_3)$
T(n)= $cte_1 + \sum_{i=1}^{n}[cte_2 + (n+1)\ cte_3]$
T(n)= cte_1 + n $[cte_2$ + (n + 1) $cte_3]$
T(n)= cte_1 + n cte_2 + n^2 cte_3 + n cte_3
T(n)= cte_1 + n $(cte_2 + cte_3)$ + n^2 $cte_3 \leq$ O(n^2).

procesarMovimientosOptimizado:

T(n)= $cte_1 + \sum_{i=1}^{n} cte_2 + \sum_{i=1}^{n} cte_3$
T(n)= cte_1 + n cte_2 + n cte_3
T(n)= cte_1 + n $(cte_2 + cte_3) \leq$ O(n).

Ejercicio 13.

Resolver las recurrencias y calcular el orden. Para cada recurrencia, se muestra, a modo de ejemplo, el código correspondiente.

`int recursivo(int n){`
`if (n <= 1)`
`  return 1;`
`else`
`  return (recursivo (n-1));`
`}`
$T(n)=\begin{cases}1, n \leq 1 \\ T(n-1)+c, n \geq 2\end{cases}$
	`int recursivo(int n){`
	`if (n == 1)`
	`  return 1;`
	`else`
	`  return (recursivo (n/2));`
	`}`
	$T(n)=\begin{cases}1, n = 1 \\ T(n/2)+c, n \geq 2\end{cases}$

`int recur (int n){`
`if (n == 1)`
`  return 1;`
`else`
`  return(recur(n/2)+recur(n/2));`
`}`
$T(n)=\begin{cases}1, n = 1 \\ 2T(n/2)+c, n \geq 2\end{cases}$
	`int recursivo(int n){`
	`if (n <= 5)`
	`  return 1;`
	`else`
	`  return (recursivo (n-5));`
	`}`
	$T(n)=\begin{cases}1, n \leq 5 \\ T(n-5)+c, n \geq 6\end{cases}$

`int recur (int n){`
`if (n == 1)`
`  return 1;`
`else`
`  return(recur(n-1)+recur(n-1));`
`}`
$T(n)=\begin{cases}1, n = 1 \\ 2T(n-1)+c, n \geq 2\end{cases}$
	`int recursivo(int n){`
	`if (n <= 7)`
	`  return 1;`
	`else`
	`  return (recursivo (n/8));`
	`}`
	$T(n)=\begin{cases}1, n \leq 7 \\ T(n/8)+c, n \geq 8\end{cases}$

(a) $T(n)=\begin{cases}1, n \leq 1 \\ T(n-1)+cte, n \geq 2\end{cases}$.

Paso 1:

T(n)= T(n - 1) + cte, si n ≥ 2.

Paso 2:

T(n)= T(n - 1 - 1) + cte + cte
T(n)= T(n - 2) + 2cte, si n ≥ 3.

Paso 3:

T(n)= T(n - 2 - 1) + cte + 2cte
T(n)= T(n - 3) + 3cte, si n ≥ 4.

Paso i (Paso general):

T(n)= T(n - i) + i cte, si n ≥ i + 1.

n - i= 1
i= n - 1.

Entonces:

T(n)= T(n - (n - 1)) + (n - 1) cte
T(n)= T(n - n + 1) + (n - 1) cte
T(n)= T(1) + (n - 1) cte
T(n)= 1 + (n - 1) cte ≤ O(n).

¿Existen constantes c > 0 y $n_0 > 0$ tales que f (n) ≤ c g (n) para todo n ≥ n_0?

Primer término:

1 ≤ cn
1 ≤ 1n
1 ≤ n.

c_1= 1; n_0= 1.

Segundo término:

cte (n - 1) ≤ cn
cte (n - 1) ≤ cte n.

c_2= cte; n_0= 1.

Entonces:

1 + (n - 1) cte ≤ c_1n + c_2n
1 + (n - 1) cte ≤ (c_1 + c_2) n
1 + (n - 1) cte ≤ (1 + cte) n, con c= 1 + cte para todo n ≥ n_0, con n_0= 1.

Por lo tanto, T(n) es O(n), ya que existen constantes c= 1 + cte y n_0= 1 tales que f (n) ≤ c g (n) para todo n ≥ n_0.

Por lo tanto, la función de tiempo de ejecución es T(n)= 1 + (n - 1) cte y el orden es O(n).

(b) $T(n) = \begin{cases} 1, n = 1 \\ T(\frac{n}{2}) + cte, n \geq 2 \end{cases}$.

Paso 1:

T(n)= T($\frac{n}{2}$) + cte, si n ≥ 2.

Paso 2:

T(n)= T($\frac{\frac{n}{2}}{2}$) + cte + cte

T(n)= T($\frac{n}{4}$) + 2cte, si n ≥ 3.

Paso 3:

T(n)= T($\frac{\frac{n}{2}}{4}$) + cte + 2cte
T(n)= T($\frac{n}{8}$) + 3cte, si n ≥ 4.

Paso i (Paso general):

T(n)= T($\frac{n}{2^i}$) + i cte, si n ≥ i + 1.

$\frac{n}{2^i}$= 1
1 * 2^i= n
2^i= n
i log_2 2= log_2 n
i * 1= log_2 n
i= log_2 n.

Entonces:

T(n)= T($\frac{n}{2^i}$) + i cte
T(n)= T($\frac{n}{2^{log_2 n}}$) + log_2 n * cte
T(n)= T($\frac{n}{n}$) + log_2 n * cte
T(n)= T(1) + log_2 n * cte
T(n)= 1 + log_2 n * cte ≤ O(log_2 n).

¿Existen constantes c > 0 y n_0 > 0 tales que f (n) ≤ c g (n) para todo n ≥ n_0?

Primer término:

1 ≤ c log_2 n
1 ≤ 1 log_2 n
1 ≤ log_2 n.

c_1= 1; n_0= 2.

Segundo término:

cte log_2 n ≤ c log_2 n
cte log_2 n ≤ cte log_2 n.

c_2= cte; n_0= 1.

Entonces:

1 + log_2 n * cte ≤ c_1 log_2 n + c_2 log_2 n

1 + log_2 n * cte ≤ (c_1 + c_2) log_2 n
1 + log_2 n * cte ≤ (1 + cte) log_2 n, con c= 1 + cte para todo n ≥ n_0, con n_0= 2.

Por lo tanto, T(n) es O(log_2 n), ya que existen constantes c= 1 + cte y n_0= 2 tales que f (n) ≤ c g (n) para todo n ≥ n_0.

Por lo tanto, la función de tiempo de ejecución es T(n)= 1 + log_2 n * cte y el orden es O(log_2 n).

(c) *T(n)*= $\begin{cases} 1, n = 1 \\ 2\,T(\frac{n}{2}) + cte, n \geq 2 \end{cases}$

Paso 1:

T(n)= 2 T($\frac{n}{2}$) + cte, si n ≥ 2.

Paso 2:

T(n)= 2 [2 T($\frac{\frac{n}{2}}{2}$) + cte] + cte
T(n)= 2 [2 T($\frac{n}{4}$) + cte] + cte
T(n)= 4 T($\frac{n}{4}$) + 2cte + cte
T(n)= 4 T($\frac{n}{4}$) + 3cte, si n ≥ 3.

Paso 3:

T(n)= 4 [2 T($\frac{\frac{n}{2}}{4}$) + cte] + 3cte
T(n)= 4 [2 T($\frac{n}{8}$) + cte] + 3cte
T(n)= 8 T($\frac{n}{8}$) + 4cte + 3cte
T(n)= 8 T($\frac{n}{8}$) + 7cte, si n ≥ 4.

Paso i (Paso general):

T(n)= 2^i T($\frac{n}{2^i}$) + (2^i - 1) cte, si n ≥ i + 1.

$\frac{n}{2^i}$= 1
1 * 2^i= n
2^i= n
i log_2 2= log_2 n
i * 1= log_2 n
i= log_2 n.

Entonces:

T(n)= $2^{log_2 n}$ T($\frac{n}{2^{log_2 n}}$) + ($2^{log_2 n}$ - 1) cte
T(n)= n T($\frac{n}{n}$) + (n - 1) cte
T(n)= n T(1) + (n - 1) cte
T(n)= n * 1 + (n - 1) cte
T(n)= n + (n - 1) cte ≤ O(n).

¿Existen constantes c $>$ 0 y $n_0 > 0$ tales que f (n) ≤ c g (n) para todo n ≥ n_0?

Primer término:

n ≤ cn
n ≤ 1n
n ≤ n.

c_1= 1; n_0= 1.

Segundo término:

cte (n - 1) ≤ cn
cte (n - 1) ≤ cte n.

c_2= cte; n_0= 1.

Entonces:

n + (n - 1) cte ≤ c_1n + c_2n
n + (n - 1) cte ≤ (c_1 + c_2) n
n + (n - 1) cte ≤ (1 + cte) n, con c= 1 + cte para todo n ≥ n_0, con n_0= 1.

Por lo tanto, T(n) es O(n), ya que existen constantes c= 1 + cte y n_0= 1 tales que f (n) ≤ c g (n) para todo n ≥ n_0.

Por lo tanto, la función de tiempo de ejecución es T(n)= n + (n - 1) cte y el orden es O(n).

(d) *T(n)*= $\begin{cases} 1, n \leq 5 \\ T(n-5) + cte, n \geq 6 \end{cases}$.

(e) *T(n)*= $\begin{cases} 1, n \leq 1 \\ 2\,T(n-1) + cte, n \geq 2 \end{cases}$.

Paso 1:

T(n)= 2 T(n - 1) + cte, si n ≥ 2.

Paso 2:

T(n)= 2 [2 T(n - 1 - 1) + cte] + cte
T(n)= 2 [2 T(n - 2) + cte] + cte
T(n)= 4 T(n - 2) + 2cte + cte
T(n)= 4 T(n - 2) + 3cte, si n ≥ 3.

Paso 3:

T(n)= 4 [2 T(n - 2 - 1) + cte] + 3cte
T(n)= 4 [2 T(n - 3) + cte] + 3cte
T(n)= 8 T(n - 3) + 4cte + 3cte
T(n)= 8 T(n - 3) + 7cte, si n ≥ 4.

Paso i (Paso general):

T(n)= 2^i T(n - i) + (2^i - 1) cte, si n ≥ i + 1.

n - i= 1
i= n - 1.

Entonces:

T(n)= 2^{n-1} T(n - (n - 1)) + (2^{n-1} - 1) cte
T(n)= 2^{n-1} T(n - n + 1) + (2^{n-1} - 1) cte
T(n)= 2^{n-1} T(1) + (2^{n-1} - 1) cte
T(n)= 2^{n-1} * 1 + (2^{n-1} - 1) cte
T(n)= 2^{n-1} + (2^{n-1} - 1) cte ≤ O(2^n).

¿Existen constantes c > 0 y n_0 > 0 tales que f (n) ≤ c g (n) para todo n ≥ n_0?

Primer término:

$2^{n-1} \leq c\ 2^n$
$2^{n-1} \leq 1 * 2^n$
$2^{n-1} \leq 2^n$.

c_1= 1; n_0= 1.

Segundo término:

cte (2^{n-1} - 1) ≤ c 2^n
cte (2^{n-1} - 1) ≤ cte 2^n.

c_2= cte_2; n_0= 1.

Entonces:

2^{n-1} + (2^{n-1} - 1) cte$\leq c_1\ 2^n + c_2\ 2^n$
2^{n-1} + (2^{n-1} - 1) cte $\leq (c_1 + c_2)\ 2^n$
2^{n-1} + (2^{n-1} - 1) cte $\leq$ (1 + cte) 2^n, con c= 1 + cte para todo n $\geq n_0$, con n_0= 1.

Por lo tanto, T(n) es O(2^n), ya que existen constantes c= 1 + cte y n_0= 1 tales que f (n) $\leq$ c g (n) para todo n $\geq n_0$.

Por lo tanto, la función de tiempo de ejecución es T(n)= 2^{n-1} + (2^{n-1} - 1) cte y el orden es O(2^n).

(f) $T(n) = \begin{cases} 1, n \leq 7 \\ T(\frac{n}{8}) + cte, n \geq 8 \end{cases}$

Ejercicio 14.

Considerar el siguiente fragmento de código:

```
int count = 0; int n = a.length;
        for (int i = 1; i <= n; i = i*2)   {
                for (int j = 0; j < n; j+=n/2)  {
                a[j]++;
                }
        }
```

Este algoritmo se ejecuta en una computadora que procesa 1.000 operaciones por segundo. Determinar el tiempo aproximado que requerirá el algoritmo para resolver un problema de tamaño n= 4.096.

Iteraciones del primer *for*:

i= 1.
i= 2.
i= 4.
…
i= 2^{k-1}.

2^{k-1}= n
(k - 1) log_2 2= log_2 n
(k - 1) * 1= log_2 n
k - 1= log_2 n
k= log_2 n + 1.

Iteraciones del segundo *for*:

j= 0.
j= $\frac{n}{2}$.

Entonces:

T(n)= $cte_1 + \sum_{i=1}^{log_2 n+1} \sum_{j=1}^{2} cte_2$
T(n)= $cte_1 + \sum_{i=1}^{log_2 n+1} 2cte_2$
T(n)= cte_1 + (log_2 n + 1) * $2cte_2$
T(n)= cte_1 + 2 (log_2 n + 1) cte_2.

T(4096)≅ log_2 4096
T(4096)≅ 12.

1.000 operaciones por segundo.

Tiempo (en segundos)$\cong \frac{12}{1000}$
Tiempo (en segundos)$\cong$ 0,012.

Por lo tanto, el tiempo aproximado que requerirá el algoritmo para resolver un problema de tamaño n= 4.096 es 0,012 segundos.

Juan Menduiña

Trabajo Práctico N° 5:

.

Ejercicio 1.