Taller 1 - Fundamentos de programación funcional y concurrente

Carlos Andres Delgado S
carlos.andres.delgado@correounivalle.edu.co

Septiembre 2023

1. Ejercicios de programación: ordenando una lista de enteros

Dada una secuencia de enteros diferentes $< x_1, x_2, \ldots, x_n >$ no necesariamente ordenada, queremos:

- Calcular **el k-ésimo elemento**, es decir, el elemento de la secuencia que iría en la posición k si estuviera ordenada ($1 \leq k \leq n$), sin necesidad de ordenar la secuencia, y
- Ordenarla, es decir construir la secuencia $< y_1, y_2, \ldots, y_n >$ tal que $\forall i \in 1..(n-1) : y_i < y_{i+1}$ y $\{x_1, x_2, \ldots, x_n\} = \{y_1, y_2, \ldots, y_n\}$.

Para ello vamos a representar una secuencia de enteros con una lista de Scala.
Una lista en Scala es el resultado de invocar al constructor $List$ con sus elementos. Por ejemplo $List(10, 8, 5, 9, 7)$ representa la secuencia de 5 elementos $< 10, 8, 5, 9, 7 >$.
Para este ejercicio utilizaremos solamente las siguientes funciones (métodos) para manipular listas de enteros ($List[Int]$):

- l.isEmpty: Boolean (devuelve si una lista l está vacía)
- l.head: Int (devuelve el primer elemento de la lista l)
- l.tail: List[Int] (devuelve la lista sin el primer elemento l)
- $x :: l$ devuelve la lista que representa la secuencia $< x, x_1, x_2, \ldots, x_n >$ si l es la lista que representa la secuencia $< x_1, x_2, \ldots, x_n >$
- $l1 + +l2$ devuelve la lista que representa la concatenación de las secuencias representadas por $l1$ y $l2$

Su tarea es construir en Scala las siguientes funciones sobre listas, especificadas para resolver los problemas mencionados.

1.1. Calcular el tamaño de una lista con un proceso iterativo

Una manera de calcular el tamaño de una lista se puede ver en el siguiente programa Scala:

```
def tamR(l:List[Int]): Int = {
  // devuelve el numero de elementos de l
  if (l.isEmpty) 0
  else 1 + tamR(l.tail)
}
```

Sin embargo este programa da origen a una recursión lineal. Su tarea consiste en desarrollar la función `tamI` tal que $tamI(l) == tamR(l)$ pero $tamI$ genere un proceso iterativo usando recursión de cola, en este caso se requiere una función auxiliar `tamIaux`

1.2. Dividiendo una lista en dos sublistas a partir de un pivote

Implemente usando recursión, las funciones `menoresQue` y `noMenoresQue`, tales que:

- $menoresQue$ recibe una lista l y un valor v y devuelve una lista con los elementos de la lista original que sean menores que v
- $noMenoresQue$ recibe una lista l y un valor v y devuelve una lista con los elementos de la lista original que no sean menores que v

Al valor v se le suele denominar un pivote.

```
  def menoresQue(l:List[Int], v:Int): List[Int] = {
    // devuelve la lista de elementos de l que son menores  que v
...
  }

  def noMenoresQue(l:List[Int], v:Int): List[Int] = {
    // devuelve la lista de elementos de l que no son menores que v
...
  }
```

1.3. Calculando el k-ésimo elemento de una lista

Utilice las funciones implementadas en los ejercicios anteriores para implementar la función `k_elem` que recibe una lista l no vacía de enteros, un entero $0 < k \leq tamI(l)$ y devuelve el k-ésimo elemento de la secuencia si estuviera ordenada (ver definición arriba) que representa la lista l.**Pista:** Para este punto use la división de mayor y menor con el tamaño de la lista, el enfoque de este ejercicio es la búsqueda binaria, cada vez que divides la lista vas obteniendo los mayores y menores de una lista cada vez más pequeña.

```
  def k_elem(l:List[Int],k:Int):Int = {
    // devuelve el k-esimo elemento de l; supone 0< k <= longitud de l
...
  }
```

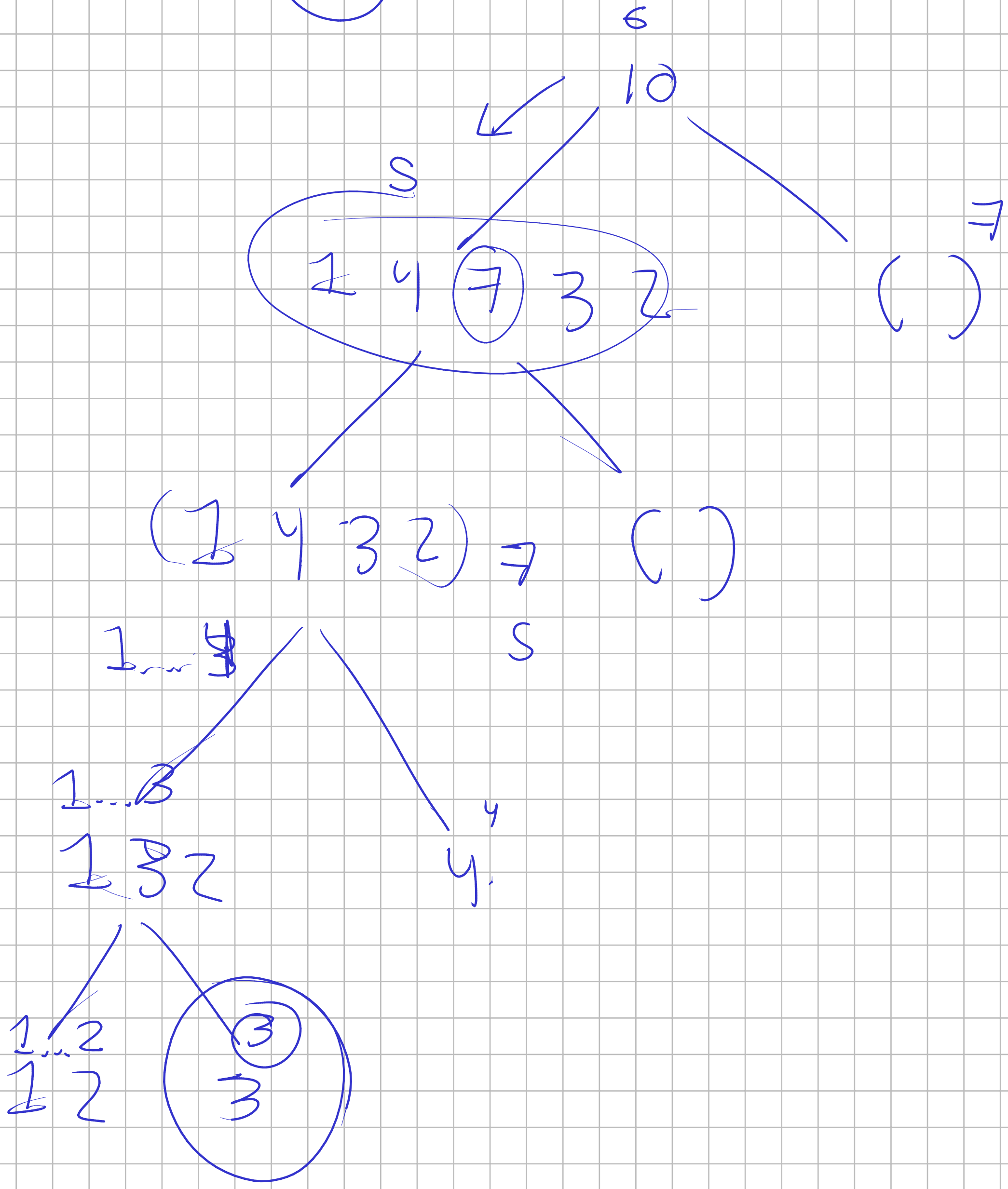
0 1 2 3 4 5

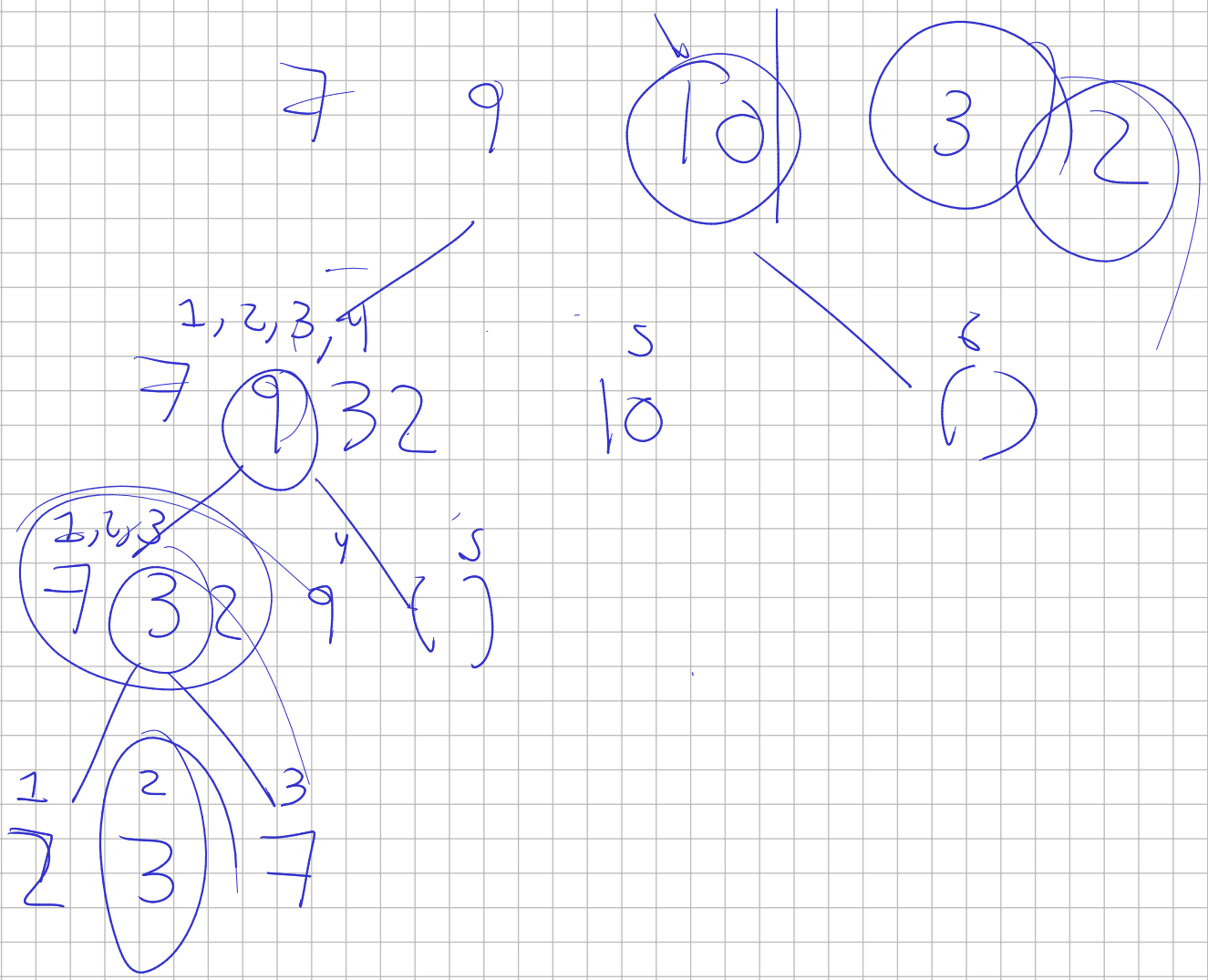
10 1 4 7 3 2

3ero

1 2 (3) 4 7 10

3 - 6 elem




7 9
10
3 2
1, 2, 3, 4
7 9 3 2
5
10
6
( )
2, 2, 3
4
5
7 3 2 9 ( )
1
2
3
2 3 7

1.4. Ordenando una lista

Utilice las funciones `menoresQue, noMenoresQue` implementadas en un ejercicio anterior para implementar la función `ordenar` que recibe una lista l no vacía de enteros, y devuelve la secuencia que representa la lista l ordenada.

La idea central es tomar la lista de entrada l y partirla, usando las funciones `menoresQue, noMenoresQue`, en dos sublistas disyuntas l_1 y l_2 de l tales que:

- los elementos de l_1 unidos a los de l_2 son los elementos de la lista l, y
- todos los elementos de la lista l_1 sean menores que todos los elementos de la lista l_2.

Y ordenar recursivamente esas dos sublistas para luego mezclarlas y obtener la lista deseada ordenada.

```
  def ordenar(l:List[Int]): List[Int] = {
...
  }
```

2. Entrega

Para esto usted debe editar los archivos SizeList.scala el cual contiene el primer punto del taller y el archivo SortList.scala el cual contiene el segundo punto, varias anotaciones:

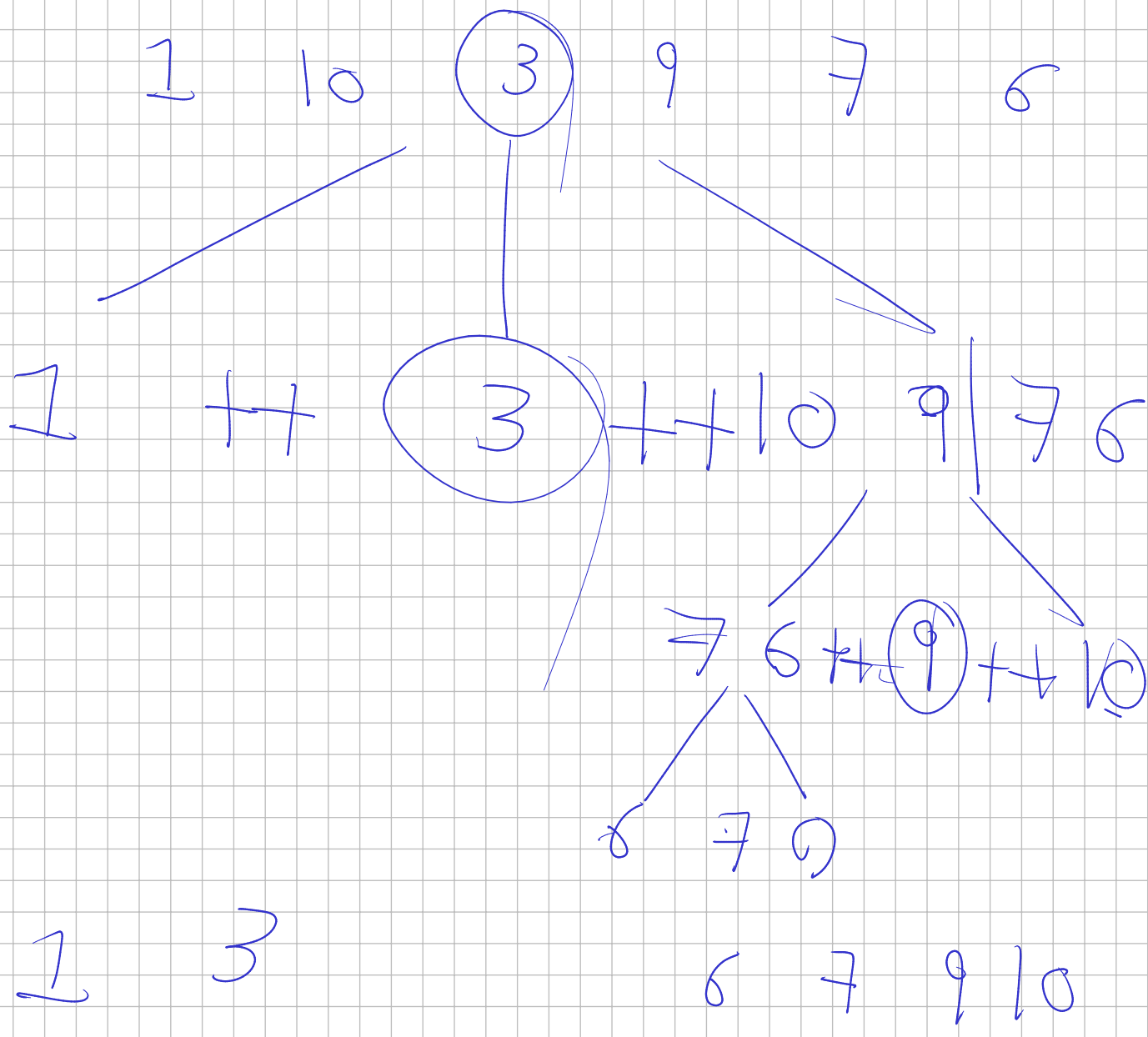
1. La fecha de entrega es **Viernes, 22 de Septiembre de 2023 a las 23:58:59** para el grupo del Martes y **Lunes 25 de Septiembre de 2023 a las 23:58:59** para el grupo del Jueves por el enlace haciendo push en el repositorio creado por el enlace creado por el docente, este trabajo es individual. **No se aceptan entregas por ningún otro medio-**

2. **Importante:** No es permitido editar otros archivos del proyecto, esto es causal para no recibir el taller y asignar una nota de 0.0, por considerarse fraude en la actividad al ser un sabotaje, situación considerada en el acuerdo 009. Cualquier falla que considere tenga el taller informar al docente para que este dé las instrucciones en caso de que sea necesario hacer algún ajuste.

3. Editar las funciones especificadas en el taller, si desea agregar alguna auxiliar hacerlo, sin embargo esto no es estrictamente necesario para completar el taller.

4. Si necesita funciones auxiliares adicionales implementarlas dentro de las funciones, no es necesario crearlas externamente.

5. Tener en cuenta que se debe seguir las indicaciones sobre programación funciona: no usar variables mutables, no utilizar ciclos iterativos (for o while) o colocar return en las funciones, si necesita usar una secuencia de instrucciones, puede usar bloques $\{\ldots\}$.



1 3 6 7 9 10

2.1. Calificación

Debe tener en cuenta que las pruebas incluyen el linting o evaluación de código, no es suficiente con que pasen las pruebas si no que su código siga el paradigma funcional.

Criterio	**Puntos**
Pasa pruebas de tamaño lista para TamI y TamIAux	20
Pasa pruebas de mayorQue y menorQue	25
Pasa pruebas de k-esimo elemento	25
Pasa pruebas de ordenar	30

Nota: El docente puede revisar cómo resolvio el problema y de acuerdo a si siguio la idea de desarrollo puede aplicar cambios en la calificación de cada uno de los puntos.

La nota n con respecto a la sumatoria de puntos obtenidos p es:

$$n = 5{,}0\frac{n}{100}$$