Nombre y Apellido: ....**.....** N°Legajo: **.....**

***Primer Parcial de Estructuras de Datos y Algoritmos 72.34***

***Primer Cuatrimestre de 2013 – 23/04/2013***

| *Ejercicio 1* | *Ejercicio 2* | *Ejercicio 3* | *Nota* |
|---|---|---|---|
| | | | |

**Condición Mínima de Aprobación: Tener por lo menos dos ejercicios con B-**

**Consideraciones a tener en cuenta. MUY IMPORTANTE**

- **El ejercicio que no respete estrictamente el enunciado será anulado.**
- **Se puede entregar el examen escrito en lápiz.**
- **Se tendrán en cuenta la eficiencia y el estilo de programación.**
- **Los teléfonos celulares deben estar apagados.**
- **Entregar los ejercicios en hojas separadas**

***Ejercicio 1***

En una lista lineal ordenada simplemente encadenada la inserción tiene complejidad temporal N. Para que la inserción se realice más rápido se decide agregarle a cada nodo un puntero al "siguiente del siguiente". De esta forma, se puede ubicar el lugar en el que se va a insertar el nuevo valor en la mitad de pasos, ya que se va recorriendo de a 2 nodos.

Implementar esta estructura, respetando la siguiente interfaz. La clase debe recibir un comparador por constructor.

```
public interface SortedFastList<T> {

	/** Agrega el elemento en la posición que corresponda (según el comparador recibido
	 * por constructor). */
	public void add(T value);

	/** Imprime por consola todos los elementos de la lista. */
	public void print();
}
```

No se puede utilizar la API de Java en este ejercicio.

**El método add debe recorrer a lo sumo N/2 +1 nodos.**

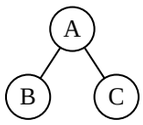
***Ejercicio 2***

Se tiene una lista que contiene los valores de un árbol binario por niveles, y de izquierda a derecha. El árbol no necesariemente es completo, pero sí se sabe que respeta la estructura de un heap (completo por niveles y de izquierda a derecha). Implementar un algoritmo que a partir de esta lista construya el árbol correspondiente.

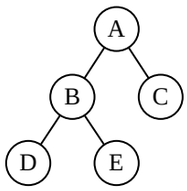
```
public class BinaryTree<T> {

	private T value;
	private BinaryTree<T> left;
	private BinaryTree<T> right;

	public BinaryTree(T value, BinaryTree<T> left, BinaryTree<T> right) {
		this.value = value;
		this.left = left;
		this.right = right;
	}

	public static <T> BinaryTree<T> buildFromList(List<T> values) {
		// IMPLEMENTAR
	}

}
```

A continuación se muestran ejemplos de listas de valores, y la salida esperada:

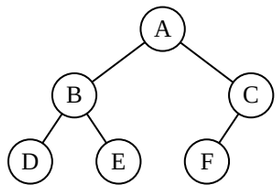
{A, B, C}




{A, B, C, D, E}




{A, B, C, D, E, F}

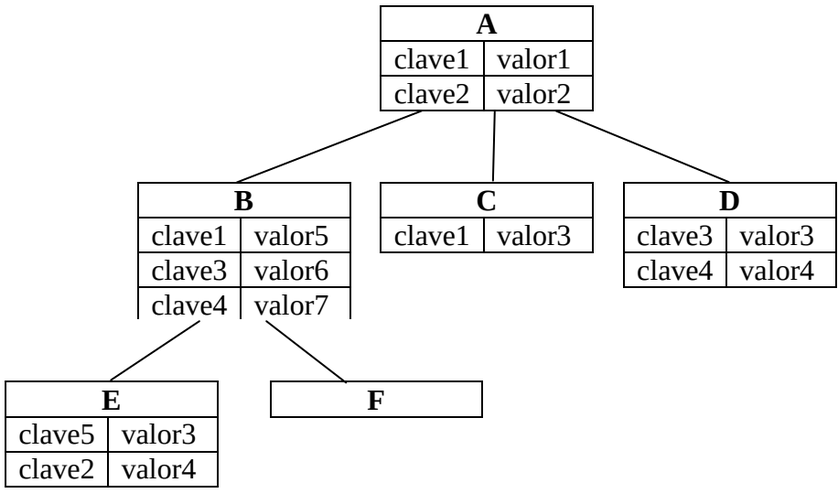



***Ejercicio 3***

Se tiene un sistema que guarda propiedades (clave/valor) para un conjunto de entidades. Cada entidad tiene un nombre único (string). Una entidad puede "heredar" de otra, agregándole propiedades o bien redefiniendo el valor de alguna propiedad ya existente.

Para representar esto se decide implementar una estructura de árbol, en donde cada entidad es un nodo (almacena el nombre y el conjunto de pares clave/valor), y los hijos de un nodo representan herencia. Por lo tanto, las propiedades de una entidad van a estar dadas por aquellas definidas en su nodo, y en toda la rama hasta la raíz.

A continuación se muestra un ejemplo de esta estructura:




Se pide:

a) Implementar una clase que represente esta estructura de datos (solamente la estructura: variables de instancia y constructor).
b) Implementar un método para consultar las propiedades de cierta entidad. Debe recibir por parámetro el nombre de la entidad a buscar, y retornar un mapa con los pares clave/valor que le corresponden.
c) Indicar el orden de complejidad temporal del método implementado en el inciso anterior.

Las siguientes son las salidas esperadas del método que se pide implementar, para cada una de las distintas entradas posibles:

A

| clave1 | valor1 |
|---|---|
| clave2 | valor2 |

B y F

| clave1 | valor5 |
|---|---|
| clave2 | valor2 |
| clave3 | valor6 |
| clave4 | valor7 |

C

| clave1 | valor3 |
|---|---|
| clave2 | valor2 |

D

| clave1 | valor1 |
|---|---|
| clave2 | valor2 |
| clave3 | valor3 |
| clave4 | valor4 |

E

| clave1 | valor5 |
|---|---|
| clave2 | valor4 |
| clave3 | valor6 |
| clave4 | valor7 |
| clave5 | valor3 |