

## ejercicio 2

José A. Mañas

28.2.2016

# ejercicio 2

---

- algoritmos de diccionarios
  - binario, BST
- programar y probar
- medir, tabular, representar con correlator
- complejidad media

# tabla lineal con búsqueda binaria

---

```
public class DiccionarioBinario
    implements Diccionario {
    private final CV[] datos;
    private int nDatos;

    public DiccionarioBinario(int max) {
        datos = new CV[max];
        nDatos = 0;
    }
}
```

Probablemente,  
lo más fácil es adaptar  
DiccionarioLineal

1. private int busca(...)
2. se adapta get()
3. se adapta put()
4. se adapta remove()

Se puede reutilizar las pruebas  
del ej1 para ponerlo a punto  
antes de medir.

# array - visualización

											max	nDatos
	5	14	15	35	40						10	5
get(14)												
	5	14	15	35	40						10	5
get(13)												
	5	14	15	35	40						10	5
put(13)												
	5	13	14	15	35	40					10	6
remove(14)												
	5	13	15	35	40						10	5

# puesta a punto

## self-check

chequear la salud del array

- al acabar put()
- al acabar remove()

```
private void check() {  
    int pos = 0;  
    while (pos < datos.length && datos[pos] != null) {  
        if (pos + 1 < datos.length && datos[pos + 1] != null) {  
            String k1 = datos[pos].getClave();  
            String k2 = datos[pos + 1].getClave();  
            if (k1.compareTo(k2) > 0)  
                System.err.println("ERROR: datos desordenados");  
            if (k1.equals(k2))  
                System.err.println("ERROR: datos duplicados");  
        }  
        pos++;  
    }  
    if (pos != nDatos)  
        System.err.println("ERROR: datos mal contados");  
    while (pos < datos.length) {  
        if (datos[pos] != null)  
            System.err.println("ERROR: datos en la zona vacia");  
        pos++;  
    }  
}
```

# binary search tree

---

```
public class BST
    implements Diccionario {
    private Nodo root;
    private int nDatos;

    public BST() {
        root = null;
        nDatos = 0;
    }
```

```
class Nodo {
    String clave;
    String valor;
    Nodo izq;
    Nodo der;

    Nodo(String clave, String valor) {
        this.clave = clave;
        this.valor = valor;
    }
}
```

Tome este código a título de inventario.

Lo más práctico es buscar código fuente en Internet y adaptarlo a la interfaz Diccionario.

Se puede reutilizar las pruebas del ej1 para ponerlo a punto antes de medir.

# BST - visualización

---

- manual
  - <https://www.cs.usfca.edu/~galles/visualization/BST.html>
- vídeo
  - <https://www.youtube.com/watch?v=qHCELIYY08w>

# puesta a punto

---

## self-check

chequear la salud del árbol

- al acabar put()
- al acabar remove()

### class Nodo

```
public void check() {  
    if (izq != null) {  
        if (izq.clave.compareTo(clave) > 0)  
            System.out.printf("ERROR: %s > %s%n", izq.clave, clave);  
        izq.check();  
    }  
    if (der != null) {  
        if (clave.compareTo(der.clave) > 0)  
            System.out.printf("ERROR: %s > %s%n", clave, der.clave);  
        der.check();  
    }  
}
```



# tareas

---

1. preparar las implementaciones de Diccionario
  - DiccionarioBinario implements Diccionario
  - BST implements Diccionario
2. poner a punto usando las pruebas
3. medir operaciones
  - hay que tunear los diccionarios para usar `OpMeter.compareTo()`
4. entregar

# mediciones

---

- creamos un diccionario amplio
  - dicc. binario: capacidad: 100.000
  - BST
- cargamos N datos
  - N= 100, 200, 500, 1.000, 2.000, 5.000, 10.000, 20.000, 50.000
- medimos el número de operaciones para buscar una clave

# contador de operaciones

---

```
public class OpMeter {  
    private static long ops;  
  
    public static long reset() {  
        ops = 0;  
        return ops;  
    }  
  
    public static long getOps() {  
        return ops;  
    }  
  
    public static int stringCompareTo(String s1,  
                                     String s2) {  
        ops++;  
        return s1.compareTo(s2);  
    }  
}
```

# entrega

---

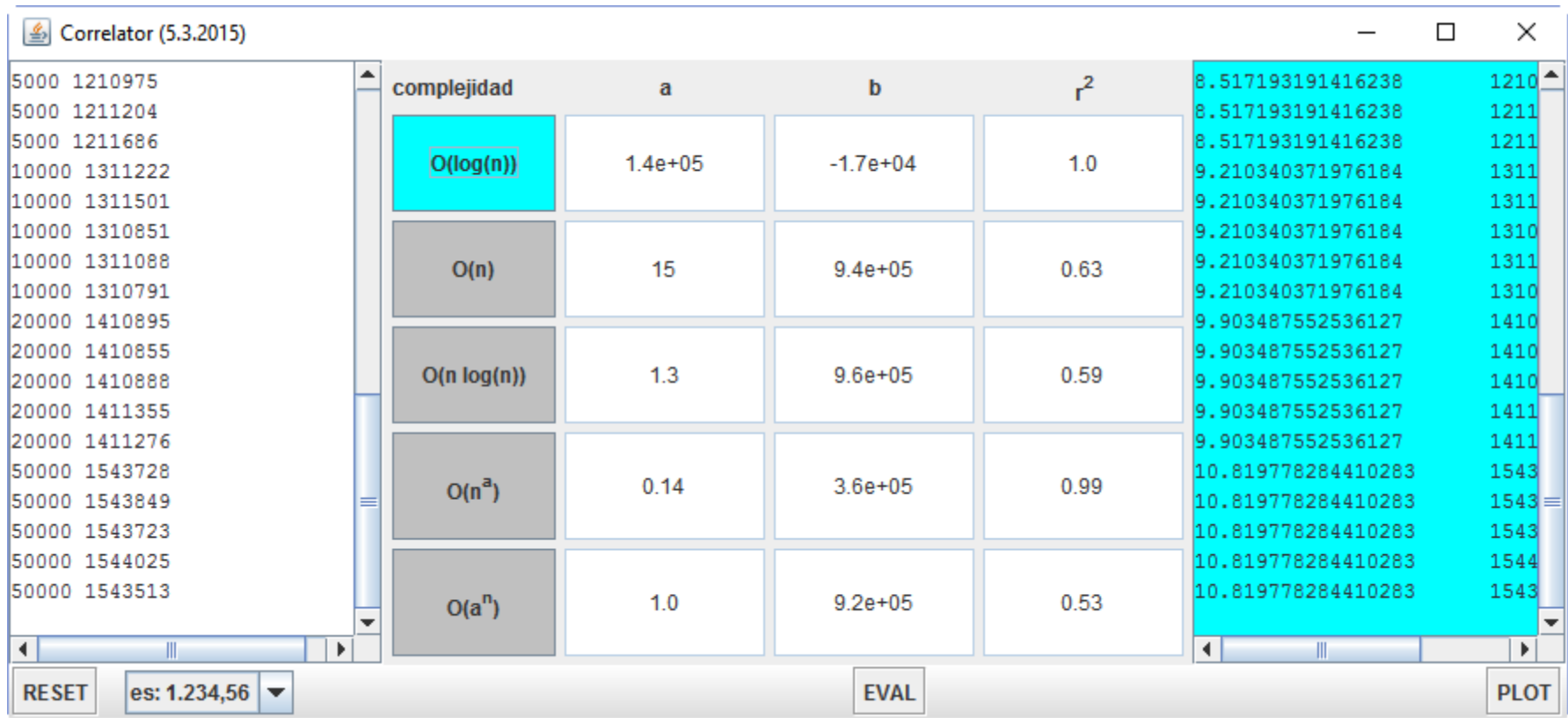
- DiccionarioBinario.java
- BST.java
  - si Nodo es una clase separada, entréguela también
- Gráficas de correlación para ambos algoritmos donde se vea claramente que son  $O(\log n)$ 
  - en formato PDF
    - lo puede generar con Powerpoint, Word, o similares

# ejemplos

---


- siguen algunos ejemplos de resultados obtenidos por los profesores
- úselos a modo de guía
  - de qué tiene que hacer con SUS datos y
  - lo que tiene que presentar como memoria del ejercicio

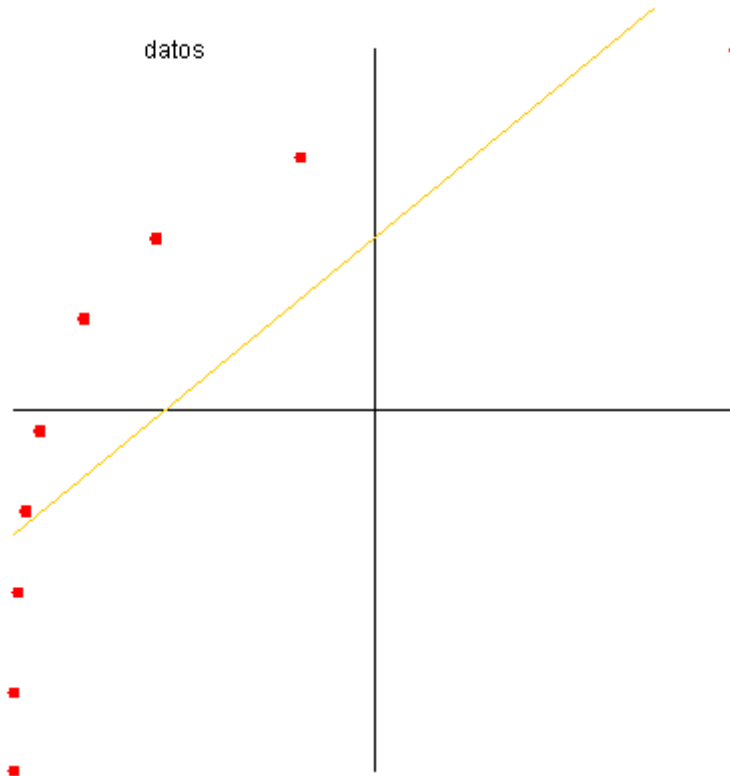
# búsqueda binaria



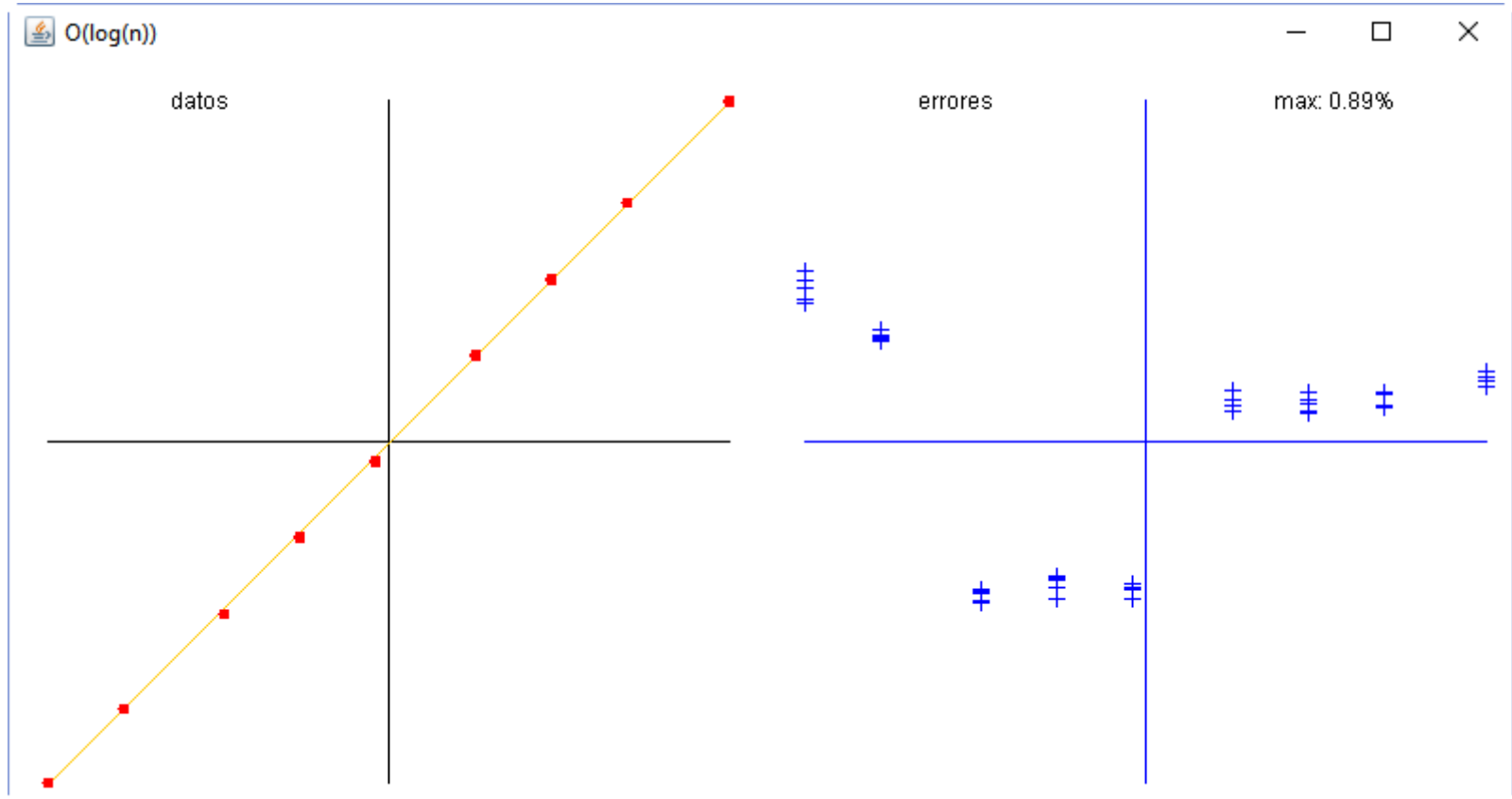
# búsqueda binaria

---

  $O(n)$

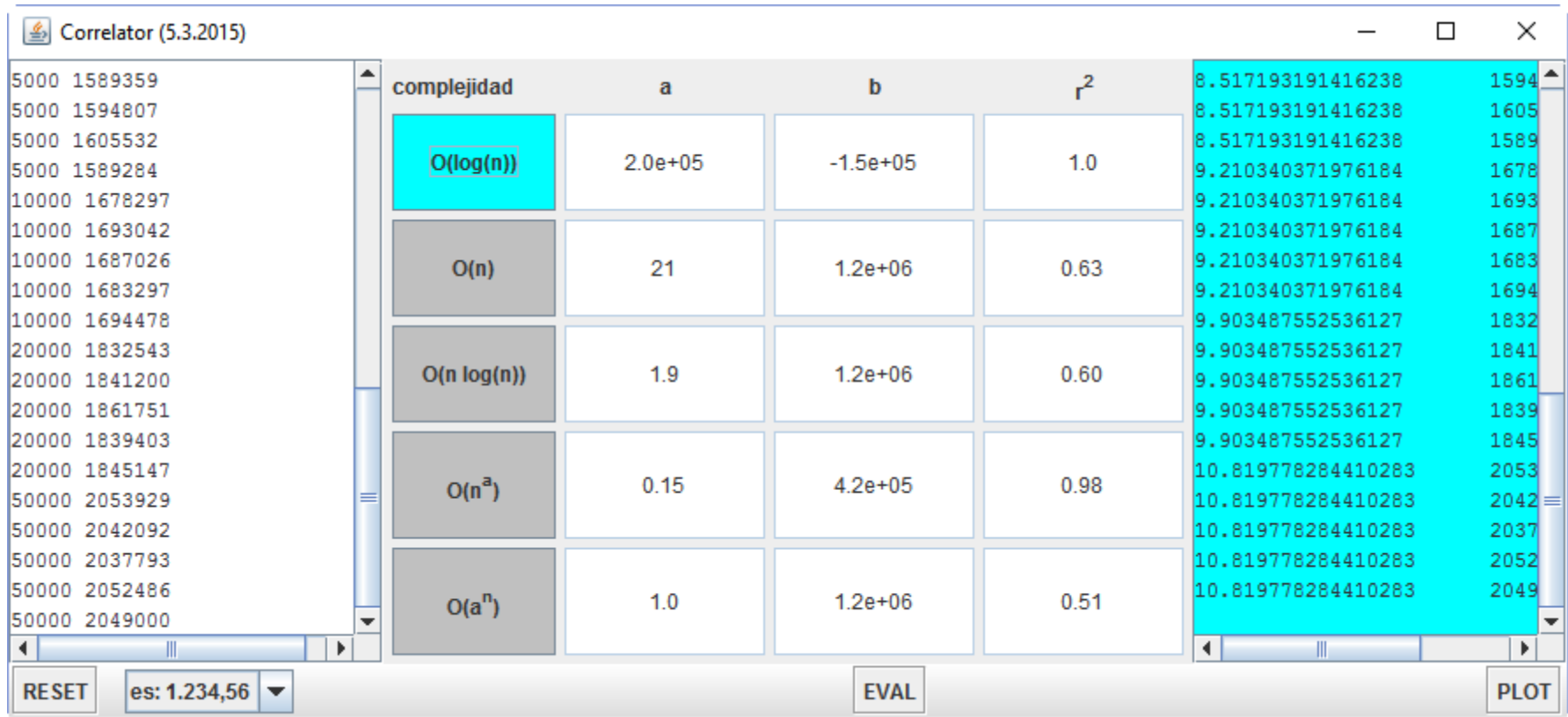


# búsqueda binaria






# BST

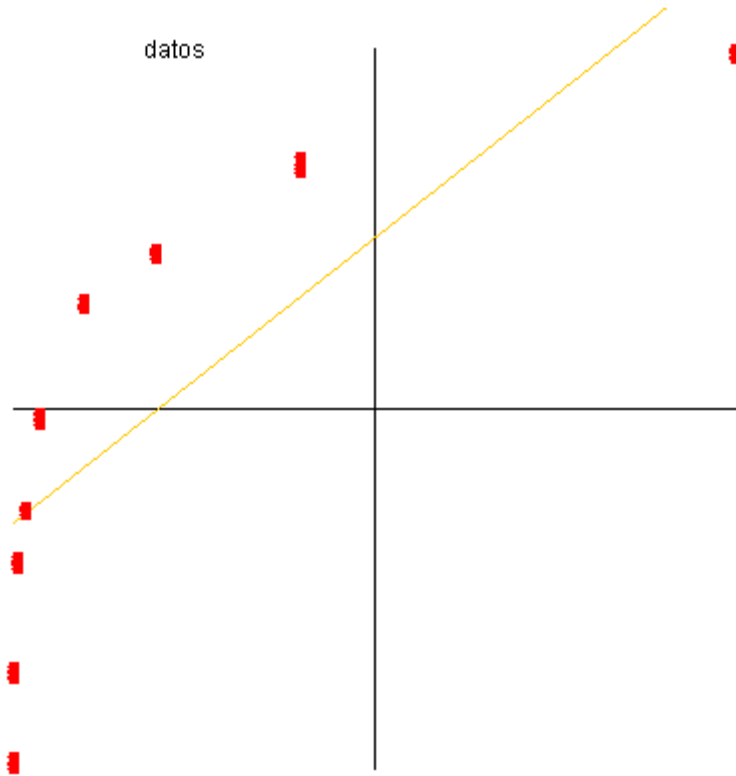


# BST

---

  $O(n)$

datos



# BST

