

GUÍA METODOLÓGICA PARA LA REALIZACIÓN Y ENTREGA DE LOS LABORATORIOS DE ESTRUCTURAS DE DATOS Y ALGORITMOS

Texto Para Estudiantes



Tabla de Contenido

Sección 0: Anotación del Docente	<u>Pág. 3</u>
Sección 1: Código de Ética	<u>Pág. 5</u>
Sección 2: Requisitos de Entrega	<u>Pág. 7</u>
Sección 3: Rúbricas de Calificación	<u>Pág. 9</u>
Sección 4: Desarrollo de Procedimientos	<u>Pág. 16</u>

SECCIÓN 0: ANOTACIÓN DEL DOCENTE

Sección 0: Anotación del Docente

Con este Texto – Guía se pretende facilitar la realización de la práctica de los laboratorios de la asignatura de Estructura de Datos y Algoritmos, y en general, ser útil en la comprensión de la metodología de trabajo para favorecer la obtención de los mejores resultados de los estudiantes

A los **estudiantes**, el **docente les entregará** un archivo ZIP, que contiene:

- ☒ Archivo PDF del taller
- ☒ Archivo Word llamado "***plantilla-laboratorios.doc***"
- ☒ PDF de la lectura recomendada
- ☒ Carpetas de código (Opcionalmente)
- ☒ Guía de laboratorios

Dentro de este ZIP, de manera opcional, también habrá traducciones al español de los ejercicios en línea.

En resumen, y una vez realizadas las actividades planteadas para cada laboratorio, los estudiantes **deberán entregar al docente** un archivo ZIP que contiene:

- ☒ Archivo PDF con informe de laboratorio usando la plantilla
- ☒ Archivo ZIP con dos códigos

SECCIÓN 1: CÓDIGO DE ÉTICA

Sección 1: Código de Ética



Cada **pareja** de estudiantes **debe dar cuenta completa de todos y cada uno** de los ejercicios que entregan en el informe



Si **utilizan código creado por ingenieros o estudiantes avanzados** (amigos, familiares, etc.) u otras fuentes, -incluyendo las librerías de Java y el código que entregue el profesor-, según el tipo de citación correspondiente (código tomado de internet, código dentro de un código, código tomado del profesor).

Si el código que cita es el de un **compañero del curso**, deben referenciarlo, comparar la eficiencia en tiempo y memoria del código con respecto al suyo, y escribirlo en su informe [\(Véase sección 4, numerales 4.15 y 4.16\)](#)



Si un estudiante no entiende el código que entregó, se manejará como fraude, de acuerdo con el Capítulo 1, Artículos 99 y 100 del Reglamento Estudiantil. Para leer más al respecto, visite: <http://bit.ly/2fm6kh2> o <http://bit.ly/2ge8vaV>

SECCIÓN 2: REQUISITOS DE ENTREGA

Sección 2: Requisitos de Entrega

Lea atenta y completamente la presente Guía Metodológica. De su comprensión y el acatamiento de las instrucciones, dependerá el éxito de su calificación para los laboratorios que se realizarán en el transcurso del curso

2.1 El laboratorio debe ser **desarrollado en parejas**



2.2 El de laboratorio **debe ser entregado en .pdf**. No se recibirán informes en formato .docx o .txt



Para exportar a PDF utilizando Microsoft Word, haga clic en 'Archivo', 'Guardar Como', elija la ubicación de destino y el nombre del archivo. En "Tipo" escoja PDF y por último de clic en "Guardar"

2.3 El **Código fuente** debe ser entregado en **formato .zip**. No se reciben archivos en .rar



2.4 El informe de laboratorio debe ser **entregado a través de EAFIT Interactiva**. No se recibirán entregas del informe de laboratorio **por correo electrónico** o fuera del **plazo establecido**



SECCIÓN 3: RÚBRICAS DE CALIFICACIÓN

Sección 3: Rúbricas de Calificación

Los informes de laboratorio serán evaluados bajo los siguientes criterios y porcentajes de evaluación:

3.1 Formatos de Entregas:

- ✓ Entrega del **informe de laboratorio** en formato **.pdf (2%)**



- ✓ Entrega del **código de laboratorio** en formato **.zip (2%)**



- ✓ Entrega del **código del Ejercicio en Línea** en **“.java”, “.cpp” o “.py” (2%)**



3.2 Autoevaluación:

- ✓ Diligenciamiento obligatorio del **formato de autoevaluación** que se encuentra en el sitio <http://bit.ly/2g8T8O6> (4%)

3.3 Para el Ejercicio en Línea:

Entregar el **código fuente** en **.java, .cpp o .py** y **explicar brevemente en el informe PDF** dicho código, (entre 3 y 6 líneas de texto)

Aquí se evaluará lo siguiente:

- ✓ Constatación que el código **resuelva** el **problema sugerido (5%)**
- ✓ Implementación eficiente, es decir, que **cumpla con los** tiempos máximos sugeridos por el juez para el lenguaje de programación usado (5%)

Sección 3: Rúbricas de Calificación

3.4 Para el Código del Laboratorio:

3.4.1 Entregar el **código de laboratorio** en **.zip** y **explicar brevemente en el informe .pdf** dicho código y su funcionamiento (entre 3 y 6 líneas de texto)



Aquí se evaluará lo siguiente:

☒ **Correctitud del Código (15%)**

☒ **Calidad del código:** indentación, acoplamiento y cohesión **(10%)**



NOTA: Tanto para el **Ejercicio en Línea** como el **Código de Laboratorio**, si tienen **más del 40%** del código igual a otro grupo o hay **extracción de internet u otra fuente sin citación**, la **valoración será 0.0** en ambos casos. Es de carácter obligatorio hacer citación.

El **porcentaje de autenticidad** se verificará con la herramienta **Moss** de la **Universidad de Stanford**

3.4.2 Entregar el **código de laboratorio** con documentación en **HTML**



3.4.2.1 Para generar la **documentación en HTML**, se recomiendan las siguientes herramientas:



Java	Javadoc	o Netbeans	o BlueJ
Python	Pydoc		
C++	Doxygen		

3.4.2.2 Para generar la **documentación en JAVA**, se recomiendan las siguientes herramientas:

Si usa Java, en la sección 3.1 se explica cómo escribir la documentación

Sección 3: Rúbricas de Calificación

Tener en cuenta los siguientes **criterios de evaluación de la documentación HTML** del código:

Netbeans	Usar icono: Martillo con escoba	
	No usar icono: Flecha de Play	
BlueJ	Control + J	

- ☑ **Descripción de cada clase** con su propósito (4%)
- ☑ **Descripción de cada método** incluyendo los parámetros que recibe y el valor que retorna (4%)
- ☑ **Información general del código** en la clase principal, incluyendo autor y versión del código (2%)

3.5 Para el Informe de Laboratorio:

3.5.1 Dar las respuestas usando el archivo "**plantilla-laboratorios.doc**" que encuentra en el ZIP que el docente entrega. **No** contiene aplicación de **normas Icontec**



3.5.2 Responder y sustentar las preguntas que encuentra en cada taller de laboratorio. Use entre 3 y 6 líneas de texto para hacerlo, solo aquellas en que se le solicita explicar el **código del laboratorio**



Aquí se evaluará lo siguiente:

- ☑ **Correctitud de las respuestas (15%)**
- ☑ **Calidad de la argumentación (10%)**
- ☑ El **análisis de complejidad** y una **explicación de las variables** en las que quedó definida qué es "n", "m", etc... (10%)
- ☑ **Quiz sobre conocimientos teóricos (10%)**

3.6 [Ejercicio Opcional] Lectura Recomendada: Cada laboratorio contiene una **lectura recomendada en formato .pdf**



Si deciden **hacer la lectura**, pueden **sumar puntos adicionales**, realizando las siguientes actividades:

- a) Resumen de lectura **(+6%)**
- b) Realización de mapa conceptual **(+4%)**



NOTA: Véase la **Sección 4**, numeral **4.19** de la presente guía, [un ejemplo concreto](#) para la realización de estas actividades

3.7 [Ejercicio Opcional] *Trabajo en Equipo y Progreso Gradual*

Trabajo en equipo. En este criterio se evalúa que hayan trabajado en equipo

Aquí se evaluará lo siguiente:

- a) Entregar **copia** de todas las **actas de reunión**, con fecha, hora e integrantes que participaron **(+3%)**
- b) El **reporte de git, svn o mercurial** con los cambios en el código y quién hizo cada cambio, con fecha, hora e integrantes que participaron **(+4%)**
- c) El reporte de cambios del informe de laboratorio que se genera Google docs o herramientas similares. **(+3%)**



NOTA: Estas respuestas también van incluidas dentro del informe PDF

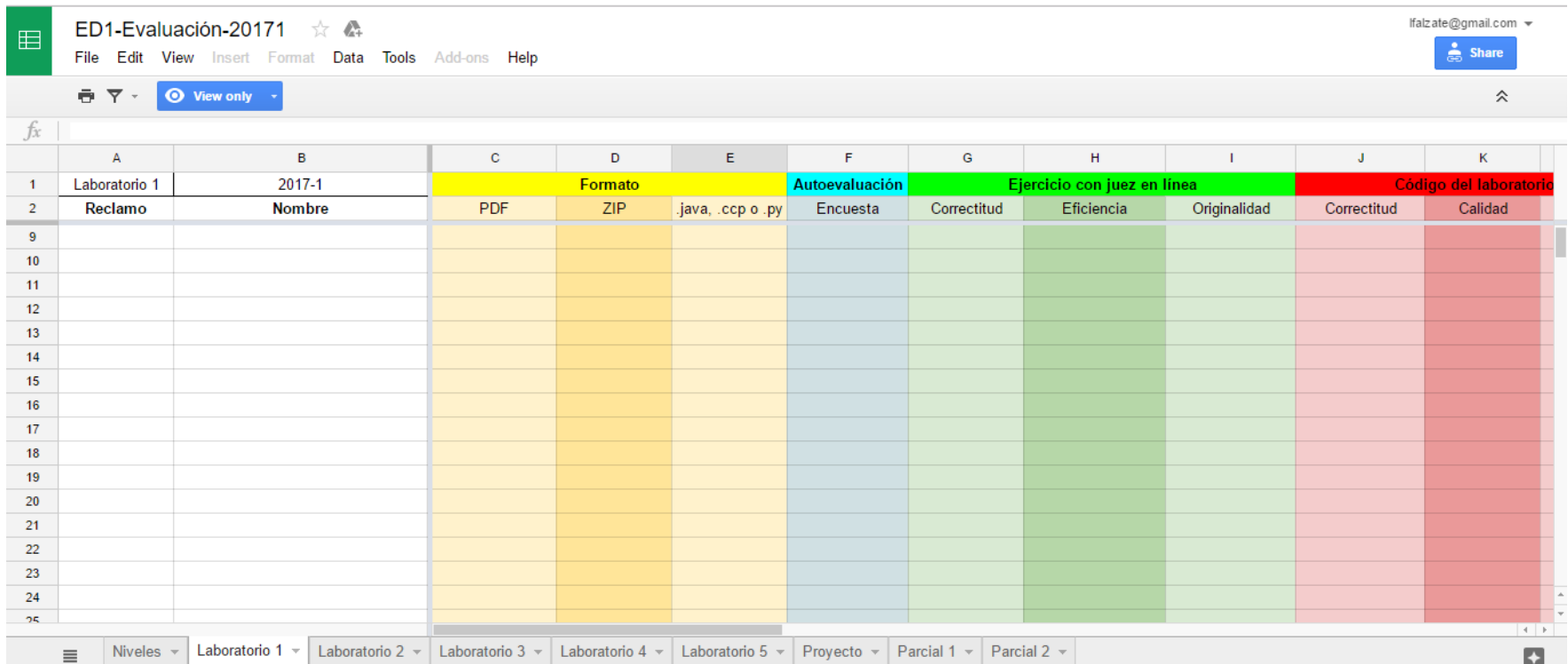
NOTA 2: Vea en la **Sección 4.21 y 4.22** de la presente guía, ***un ejemplo concreto*** para la realización de estas actividades

Sección 3: Rúbricas de Calificación

3.8 Visualización de Calificaciones

A través de **Eafit Interactiva** encontrarán **un enlace** que les permitirá **ver un registro de las calificaciones** que **emite el docente** para cada taller de laboratorio y según las rubricas expuestas

El enlace les llevará a un archivo como el que relacionamos a continuación y que les permitirá llevar dicho registro:



	A	B	C	D	E	F	G	H	I	J	K
1	Laboratorio 1	2017-1	Formato			Autoevaluación	Ejercicio con juez en línea			Código del laboratorio	
2	Reclamo	Nombre	PDF	ZIP	.java, .ccp o .py	Encuesta	Correctitud	Eficiencia	Originalidad	Correctitud	Calidad
9											
10											
11											
12											
13											
14											
15											
16											
17											
18											
19											
20											
21											
22											
23											
24											
25											

SECCIÓN 4: DESARROLLO DE PROCEDIMIENTOS

4.1 Cómo escribir la documentación HTML de un código usando JavaDoc

Veamos en primer lugar **qué se debe incluir al documentar una clase y sus métodos**:

- a) Nombre de la clase, descripción general, número de versión, nombre de autores
- b) Documentación de cada constructor o método (especialmente los públicos), incluyendo: nombre del constructor o método, tipo de retorno, nombres y tipos de parámetros si los hay, descripción general, descripción de parámetros (si los hay), descripción del valor que devuelve

En la página siguiente, observe la tabla donde se muestran algunas de las palabras reservadas (tags):



NOTA: Recuerde documentar con Doxygen si usan C++ o con Pydoc si usan Python

Sección 4: Desarrollo de Procedimientos

TAG	DESCRIPCIÓN	COMPRENDE
@author	Nombre del desarrollador.	Nombre autor o autores
@deprecated	Indica que el método o clase es obsoleto (propio de versiones anteriores) y que no se recomienda su uso.	Descripción
@param	Definición de un parámetro de un método, es requerido para todos los parámetros del método.	Nombre de parámetro y descripción
@return	Informa de lo que devuelve el método, no se aplica en constructores o métodos "void".	Descripción del valor de retorno
@see	Asocia con otro método o clase.	Referencia cruzada referencia (#método()); clase#método(); paquete.clase; paquete.clase#método()).
@version	Versión del método o clase.	Versión

Tomado de <http://bit.ly/2gcuaH>

4.2 Cómo generar arreglos con valores aleatorios en Java

```
import java.util.Random;
public class test {
    public static void main(String[] args) {
        int size = 1000;
        int max = 5000;
        int[] array = new int[size];
        Random generator = new Random();
        for (int i = 0; i < size; i++)
            array[i] = generator.nextInt(max);
    }
}
```

Sección 4: Desarrollo de Procedimientos

4.3 Cómo aumentar el tamaño del *heap* en Java

Si aparece un error “java.lang.OutOfMemoryError: Java *heap* space” error (64MB heap size), a continuación las soluciones:

- ☑ En **BlueJ**, no hemos podido solucionarlo, hay varios acercamientos en Internet, pero hasta ahora no funcionan.
- ☑ En **Netbeans**, mirar aquí cómo se hace: <http://bit.ly/2fYXHud>

Lo que se muestra en el sitio web referenciado anteriormente, es **cómo aumentar el tamaño del *heap* con la directriz -Xmx**, la cual simplemente **dice cuál es el tamaño máximo** que puede tener el *heap*.

El *heap* es donde se guardan los objetos al ser creados, es decir, cuando ejecutan Objeto o = new Objeto(). Allí también se guardan los arreglos y otras variables. La directriz -Xms es el tamaño inicial del *heap*. Asignándole 512MB (-Xms512m) no deberían tener ningún problema.



NOTA: Java limita el *heap* a 64MB y un arreglo de 100 millones ocupa 400 MB, por esto podría ser el error

Sección 4: Desarrollo de Procedimientos

4.4 Cómo aumentar el tamaño del *heap* y del *stack* en Java

Recuerden que el ***heap*** es donde se guardan los objetos y arreglos que uno crea con new. Si crean arreglos muy grandes, no duden en incrementar el ***heap*** así: **-Xmx512m**, por ejemplo

Ahora, el ***stack*** es donde se guarda la pila de la recursión. Si usan recursiones que hagan muchos millones de llamados, no duden en incrementar la pila así: **-Xss512m**, por ejemplo



NOTA: SS = Stack Size.

4.5 Cómo visualizar el montículo (*heap*) y el *stack*, y el consumo total de memoria de Java

Si tienen curiosidad o presentan otros problemas, les recomiendo esta herramienta para visualizar gráficamente lo que está pasando con la máquina virtual de Java: <http://visualvm.java.net/>. VisualVM permite ver el montículo (*heap*), la pila y hace una gráficas a ver cómo se están llenando

Puede pasar que se llene la RAM y el Sistema Operativo pase parte de sus arreglos a Disco Duro, volviendo lenta la ejecución. Si es así, revisar en el monitor de actividad que ofrece su Sistema Operativo

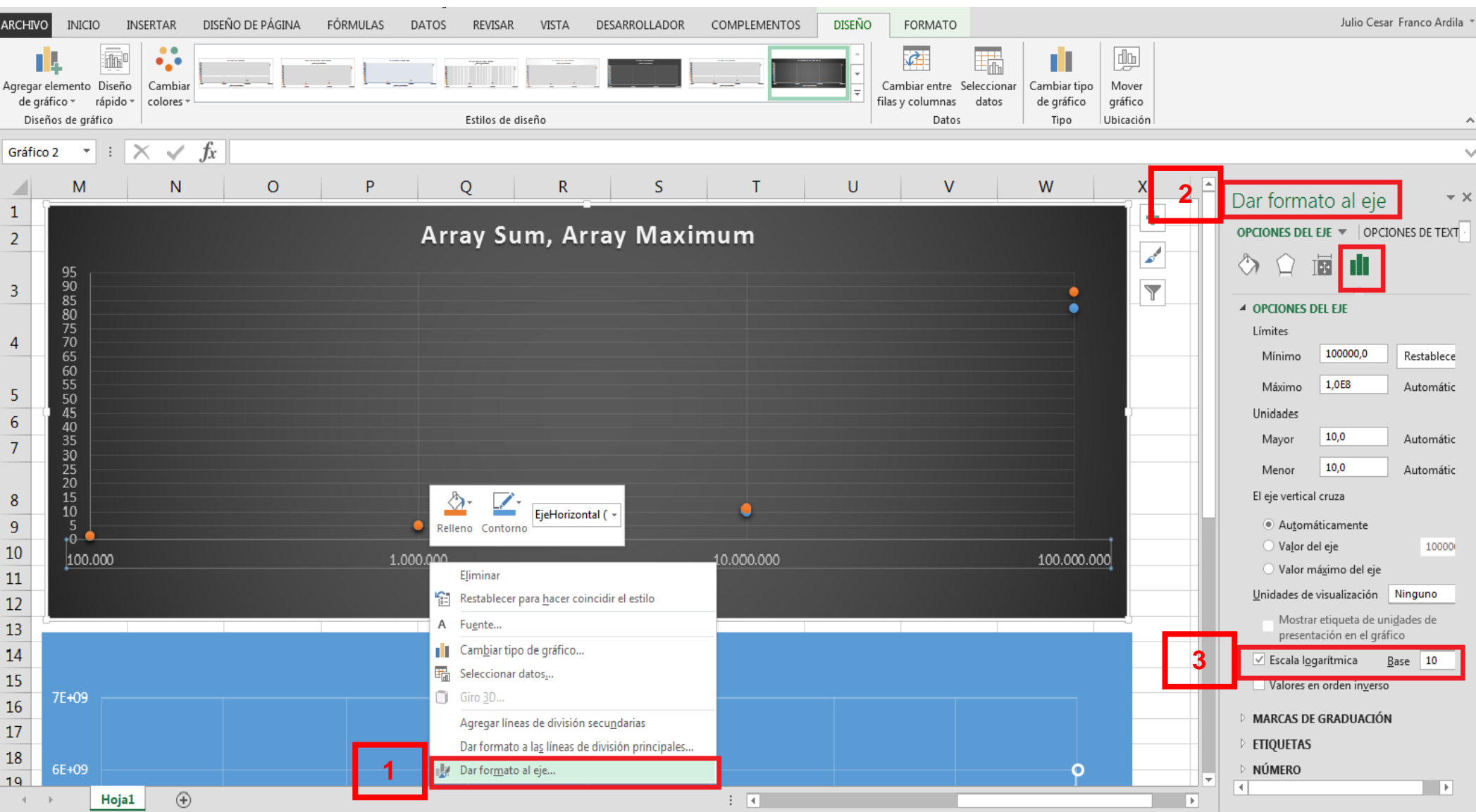
4.6 Cómo usar la escala logarítmica en Microsoft Excel 2013

Lo primero que hay que hacer es **dar clic derecho en el eje X de la gráfica** (por ejemplo, en algún número del eje x). Luego, **clic izquierdo en la opción "Dar formato al eje..."**.

En la versión de Excel 2013, inmediatamente en la parte derecha de la pantalla, **aparecerá un cuadro de opciones del formato del eje**, en el que se podrá **configurar el valor inicial y final** que tendrá el eje, entre otros. Antes de la sección "**Marcas de Graduación**", en negrilla aparece "**Escala Logarítmica**", **en base 10**. Sólo es necesario para el eje X debido a que los valores de éste eje son potencias de 10

En la página siguiente, observe gráficamente el procedimiento

Sección 4: Desarrollo de Procedimientos



4.7 Cómo calcular el tiempo que toma un código en ejecutarse en Java

```
long startTime = System.currentTimeMillis();  
// ... do something ...  
long estimatedTime = System.currentTimeMillis() - startTime;
```

4.8 Cómo definir una clase Pareja en Java

En Java, **no es posible utilizar tuplas**, por ejemplo, (1,2) o (2,3), como sucede en lenguajes dinámicamente tipados como Python o Ruby. **Para poder guardar dos valores** en un **solo nodo** de una lista o para poder **retornar dos valores**, es necesario **crear un objeto de una clase** que llamaremos **Pareja**. Para poder **guardar el vértice de destino y el peso** en una lista de adyacencia, podemos **crear una lista de listas de parejas**:

```
LinkedList<LinkedList<Pareja>> listaDelistas = new LinkedList<LinkedList<Pareja>>();
```

La clase pareja se define de la siguiente forma:

```
class Pareja{  
    public int peso;  
    public int vertice;  
}
```

Otra alternativa es usar `javafx.util.Pair`

Sección 4: Desarrollo de Procedimientos

4.9 Cómo hacer una lista de Neveras

En Java, si se quiere **representar una nevera con su descripción y código**, una forma común es **hacer una clase Nevera** y luego **crear una lista de Neveras**.

No es posible guardar dos objetos en el nodo de una lista, así que uno puede hacer una **clase Nevera** (que contiene dos cosas) y luego hacer una **lista donde cada elemento es una Nevera**.

```
class Nevera
{
    public String descripcion;
    public int codigo;
    public Nevera(int c, String d) {descripcion = d; codigo = c;}
}

...

LinkedList<Nevera> neveras = new LinkedList<Nevera>();
```

Sección 4: Desarrollo de Procedimientos

4.10 Cómo usar clases abstractas en Java

Una **clase abstracta** es una clase que **funciona como una especificación para que otras clases la implementen**. Por ejemplo, en Java, la clase **AbstractList** sirve como especificación para diversos tipos de listas como son **ArrayList** y **LinkedList**.

La ventaja de la clase abstracta es que **el programador puede realizar un código genérico** que funcione para **ArrayList** y **LinkedList**. De igual manera, en este laboratorio, haremos lo mismo para dígrafos.

4.11 Cómo escribir la complejidad de un ejercicio en línea

1. Programa iterativo con una sola entrada, usualmente llamada “n”

a) Escribir el programa o algoritmo

```
for (int i = 1; i <= n; i++)
    for (int j = 1; j <= n; j++)
        print(i+"*"+j+"="+i*j);
```

Sección 4: Desarrollo de Procedimientos

b) Etiquetar el programa:

```
for (int i = 1; i <= n; i++) // C1*n
    for (int j = 1; j <= n; j++) //C2*n²
        print(i+"*"+j+"="+i*j); //C3*n²
```

c) Calcular el T(n) sumando todas las anotaciones:

$$T(n) = (C2 + C3)n^2 + C1.n$$

d) Aplicar la notación O:

$$T(n) \text{ es } O((C2 + C3)n^2 + C1.n)$$

e) Aplicar la regla de la suma:

$$T(n) \text{ es } O((C2 + C3)n^2)$$

f) Aplicar la regla del producto:

$$T(n) \text{ es } O(n^2)$$

Sección 4: Desarrollo de Procedimientos

2. Programa iterativo con varias entradas, usualmente llamadas “n”, “m”... o “V”, “E”

a) Escribir el programa o algoritmo

```
for (int i = 1; i <= n; i++)
    for (int j = 1; j <= m; j++)
        print(i+"*"+j+"="+i*j);
```

b) Etiquetar el programa

```
for (int i = 1; i <= n; i++) // C1*n
    for (int j = 1; j <= m; j++) //C2*n.m
        print(i+"*"+j+"="+i*j); //C3*n.m
```

c) Calcular el T(n) sumando todas las anotaciones

$$T(n) = (C2 + C3)n.m + C1.n$$

d) Aplicar la notación O

$$T(n) \text{ es } O((C2 + C3)n.m + C1.n)$$

e) Aplicar la regla de la suma

$$T(n) \text{ es } O((C2 + C3)n.m)$$

f) Aplicar la regla del producto

$$T(n) \text{ es } O(n.m)$$

Sección 4: Desarrollo de Procedimientos

3. Programa recursivo

a) Escribir el programa o algoritmo

```
SubProceso result <- Fibo( n )
  Definir result Como Entero;
  Si n <= 1 Entonces
    result <- n;
  Sino
    result<-Fibo(n-1)+Fibo(n-2);
```

b) Etiquetar el programa

```
SubProceso result <- Fibo( n )
  Definir result Como Entero; // C1
  Si n <= 1 Entonces // C2
    result <- n; //C3
  Sino
    result<-Fibo(n-1) +Fibo(n-2); //C4+T(n-1)+T(n-2)
```

c) Escribir el T(n) como una ecuación de recurrencia

$T(n) = C1 + C2 + C3$, si $n \leq 1$

$T(n) = C4 + T(n-1) + T(n-2)$, de lo contrario

Sección 4: Desarrollo de Procedimientos

- d) Resolver la ecuación de recurrencia usando la teoría de polinomio característico, teorema maestro o <https://www.wolframalpha.com/>

$$T(n) = C \cdot 2^n + C'$$

- e) Aplicar la notación O

$$T(n) \text{ es } O(C \cdot 2^n + C')$$

- f) Aplicar la regla de la suma

$$T(n) \text{ es } O(C \cdot 2^n)$$

- g) Aplicar la regla del producto

$$T(n) \text{ es } O(2^n)$$

Sección 4: Desarrollo de Procedimientos

4.12 Cómo resolver ecuaciones de recurrencia lineales y no lineales

Hay 2 grandes tipos de ecuaciones de recurrencia. **Las lineales tipo $T(n) = T(n-1) + C$ y las no lineales tipo $T(n) = T(2n/3) + n$.** Las primeras se resuelven por expansión (también llamado, inducción) y las no lineales son más truculentas.

Dos libros recomendados, que pueden encontrarlos en Biblioteca, para estudiar este tema son:

- ☑ Thomas Cormen, Introduction to Algorithms (3th edition), 2009. Chapter 4
- ☑ John Hopcroft et al., Data Structures and Algorithms, 1983. Chapter 9

Sobre los autores, Cormen es un ilustrado de la complejidad en MIT y Hopcroft en Cornell. Hopcroft ha visitado Eafit y hay gran aprecio por él, pero el libro no está tan actualizado.

Finalmente, y no está de más, "los chinos" hacen muy buena explicación de complejidad, incluyendo las ecuaciones de recurrencia lineales, pero no entran en las no lineales en:

- ☑ R.C.T Lee et al., Introducción al análisis y diseño de algoritmos, 2005. Capítulo 2.

Si quieren un resumen en Español del capítulo 4 de Cormen, léanlo en <http://bit.ly/2jWx4si>

Sección 4: Desarrollo de Procedimientos

4.13 Cómo usar Scanner o *BufferedReader*

Para la **lectura de datos por entrada estándar** en Java existen **dos formas** sencillas, una con la **clase Scanner** y otra con la **clase *BufferedReader***. Para conocer más visite: <http://bit.ly/2ge1maJ>

4.14 Cómo hacer pruebas unitarias en BlueJ usando JUnit

Las **pruebas unitarias** sirven para **probar el funcionamiento de un módulo de código**, por ejemplo, de un método de una clase.

Veamos, para una clase Sumador, que tiene un método que suma dos enteros, podemos diseñar pruebas, por ejemplo, que si se llama sumar (2,2) retorne 4, si se llama sumar (2,1) retorne 3, etc.

Para **hacer esto en BlueJ**, creamos una clase **SumadorTest** que sea de tipo **UnitTest**, representadas de color verde. Después, escogemos la opción “**create Test method**” en la clase **SumadorTest**.

Esto nos permite “**grabar**” una **secuencia de pasos**, obtener un **resultado** y decir **cuál debe ser el valor** de ese resultado. Como un **ejemplo**, podemos crear una instancia de la **clase Sumador**, luego **llamar el método suma** con parámetros (2,2) y el test debe **verificar** que el **valor de retorno** sea 4.

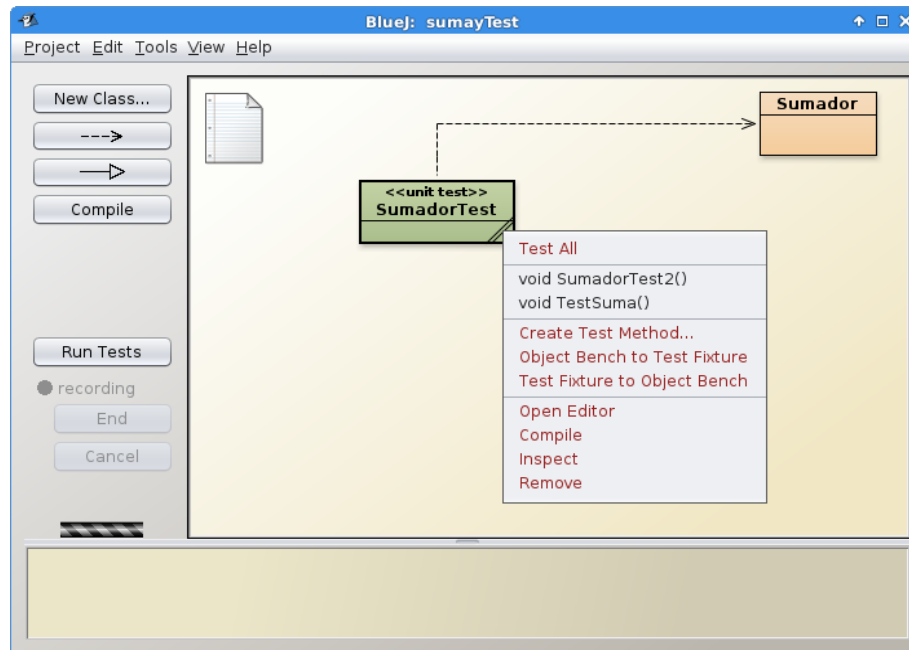
Así mismo podemos **hacer varios tests** para los otros métodos de la clase Sumador. Una vez estén hechos los tests, podemos **llamar a cada test independiente o la funcionalidad TestAll** que llama todos los tests.

Sección 4: Desarrollo de Procedimientos

Las pruebas son útiles para **verificar** que cuando uno **cambia el código** de su programa, **los métodos** sigan teniendo el **comportamiento esperado**.

Además, las pruebas **se deben diseñar incluso antes de escribir los métodos**, de la forma que se puedan **verificar** desde la **primera versión** que uno tenga de cada método de cada clase.

Finalmente, **una buena práctica es hacer tests para cada método por separado**, antes de hacer tests que involucren el llamado de varios métodos



4.15 Cómo compilar pruebas unitarias en Eclipse y en Netbeans

Eclipse: Clic derecho en el proyecto Eclipse y navegar de la siguiente forma: Properties, Java Build Path, Libraries, Add Library, Junit.

Netbeans: Click derecho en el proyecto del Laboratorio, en el panel *Projects*. Seleccionar el menú *Properties*. Seleccionar la categoría *Libraries* del panel *Categories*. Seleccionar la pestaña *Compile*. Hacer click en el botón *Add Library*. En el cuadro de diálogo *Add Library*, seleccionar la librería *JUnit 4*. Hacer click en el botón *Add Library*.

4.16 Cómo reportar código tomado de Internet

1. Para **citar código** en un reporte, **se recomienda este formato:**

<author(s) names> (<date>) <title of program/source code> (<code version>) [<type>]. Web address or publisher.

Por ejemplo,

Smith, J (2011) GraphicsDrawer source code (Version 2.0) [Source code].
<http://www.graphicsdrawer.com>

Sección 4: Desarrollo de Procedimientos

2. Para **citar código dentro de un código**, se recomienda este formato:

```

/*****
*   Title: <title of program/source code>
*   Author: <author(s) names>
*   Date: <date>
*   Code version: <code version>
*   Availability: <where it's located>
*
*****/

```

Por ejemplo,

```

*****/
*   Title: GraphicsDrawer source code
*   Author: Smith, J
*   Date: 2011
*   Code version: 2.0
*   Availability: http://www.graphicsdrawer.com
*
*****/

```

Tomado de <http://bit.ly/2fm2zYI>

4.17 Cómo reportar código tomado de un profesor

Igual que para el **código tomado de internet**, pero colocando el **nombre del profesor** como **autor**, versión del código 1.0, y disponibilidad <http://interactiva.eafit.edu.co>

Por ejemplo,

Toro, M (2016) Graphics Drawer source code (Version 1.0) [Source code].
<http://interactiva.eafit.edu.co>

4.18 Respuestas del Quiz

Quiz tipo Prueba Saber Pro con preguntas de selección múltiple con única opción de respuesta. Aquí deberá responder usando el número de pregunta, seguido de la letra que contiene la opción de respuesta que considera apropiada

Por ejemplo,

- 1) a
- 2) d
- 3) c

Sección 4: Desarrollo de Procedimientos

4.19 Ejemplos para calcular la complejidad de un ejercicio de CodingBat

4.19.1 MaxSpan

Consider the leftmost and rightmost appearances of some value in an array. We'll say that the "span" is the number of elements between the two inclusive. A single value has a span of 1. Returns the largest span found in the given array. (Efficiency is not a priority.)

```
public int maxSpan(int[] nums) {
    int max=nums.length;
    int cont1=0;
    int cont2=0;
    for(int i=0;i<max;i++){
        if(nums[i]==nums[max-1]){
            cont2=max-i;
            break;
        }
    }
    for(int i=max-1;i>=0;i--){
        if(nums[i]==nums[0]){
            cont1=i+1;
            break;
        }
    }
    if(cont1>cont2)return cont1;
    return cont2;
}
```

Sección 4: Desarrollo de Procedimientos

En este algoritmo se encuentran dos ciclos, pero estos no están anidados, sino que se encuentran por aparte, es decir que como los ciclos es lo que nos importan y ninguno contiene al otro entonces la complejidad de este algoritmo es lineal.

$$T(n) = C_1 + C_2n + C_3n$$

$$T(n) = C_1 + C_4n$$

Aplicando las reglas del producto y de la suma:

$$O(n) = C_4n$$

$$O(n) = n$$

4.19.2 Fix34

Return an array that contains exactly the same numbers as the given array, but rearranged so that every 3 is immediately followed by a 4. Do not move the 3's, but every other number may move. The array contains the same number of 3's and 4's, every 3 has a number after it that is not a 3 or 4, and a 3 appears in the array before any 4.

Sección 4: Desarrollo de Procedimientos

```
public int[] fix34(int[] nums) {  
    int max=nums.length;  
    int arr1[]=new int[max];  
    int arr2[]=new int[max];  
    for(int i=0;i<max;i++){  
        if(nums[i]==3){  
            arr1[i+1]=nums[i+1];  
            nums[i+1]=4;  
        }  
    }  
    for(int i=0;i<max;i++){  
        if(nums[i]==4 && arr1[i]==0){  
            for(int j=0;j<max;j++){  
                if(arr1[j]!=0&&arr1[j]!=4){  
                    nums[i]=arr1[j];  
                }  
            }  
        }  
    }  
    return nums;  
}
```

Sección 4: Desarrollo de Procedimientos

En este caso el algoritmo consta de 3 ciclos, y se puede ver que un ciclo esta anidado con el otro, lo cual vuelve la complejidad cuadrática, mientras que el otro ciclo si está totalmente aparte de estos dos y no afecta en nada la complejidad que se encontró.

$$T(n) = C_1 + C_2n + C_3n^2$$

Aplicando las reglas del producto y de la suma:

$$O(n) = C_2n + C_3n^2$$

$$O(n) = C_3n^2$$

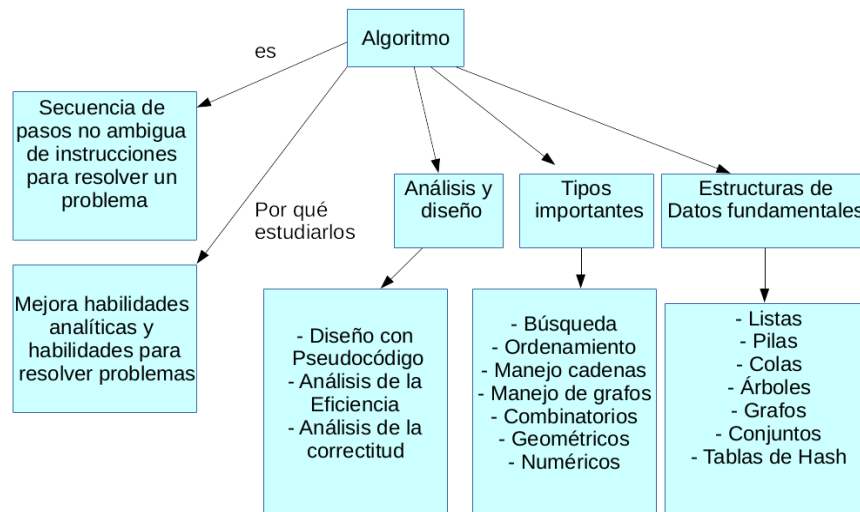
$$O(n) = n^2$$

Sección 4: Desarrollo de Procedimientos

4.20 Ejemplo para realización de actividades de las Lecturas Recomendadas

4.20.1 Ejemplo de resumen: Un algoritmo es una secuencia no ambigua de instrucciones para resolver un problema. Es importante estudiarlos porque mejora las habilidades analíticas para resolver problemas. Los tipos importantes son: de búsqueda, ordenamiento, manejo de cadenas, manejo de grafos, combinatorios, geométricos y numéricos. Las estructuras de datos fundamentales son: listas, pilas, colas, árboles, grafos, conjuntos y tablas de hash. Para diseñar un algoritmo se utiliza una notación llamada pseudocódigo. Para analizar los algoritmos existen dos tipos de análisis: de eficiencia y de correctitud.

4.20.2 Representación gráfica de los conceptos más importantes del artículo *



NOTA: La gráfica de conceptos o mapa conceptual, **no será un “copy paste”** que hagan desde la web.

Deben **construir su propio mapa de conceptos** a partir de herramientas como las que encuentra en <https://caco.com/>

*Ejemplo a partir de la lectura '*Introduction to the design and analysis of algorithms. Chapter 1*'

Sección 4: Desarrollo de Procedimientos

4.21 Ejemplo de cómo hacer actas de trabajo en equipo usando Tablero Kanban

Integrante	Fecha	Hecho	Haciendo	Por Hacer
Mateo	01/08/2016	Implementé matrices de adyacencia		Probar Dijkstra sobre mi implementación
Catalina	01/08/2016	Implementó listas de adyacencia		Probar Dijkstra sobre mi implementación
Mateo	03/08/2016	Dijkstra funcionó	Revisar el trabajo de Catalina	Resolver ejercicios en línea
Catalina	03/08/2016	Dijkstra funcionó	Revisar el trabajo de Mateo	Resolver ejercicios en línea
Mateo	06/08/2016	Resolver ejercicios en línea	Resolviendo las preguntas entre los dos	Hacer la lectura cada uno
Catalina	06/08/2016	Resolver ejercicios en línea	Resolviendo las preguntas entre los dos	Hacer la lectura cada uno

4.22 Cómo ver el historial de revisión de un archivo en Google Docs

1. Abre un documento, una hoja de cálculo, una presentación o un dibujo
2. Haz clic en **Archivo** > **Ver historial de revisión**
3. Haz **clic en una marca de tiempo** en el panel de la derecha para ver una versión anterior del archivo. También verás quién ha editado el archivo **debajo de la marca de tiempo**. Las modificaciones que cada persona ha hecho **se muestran en el color** que aparece junto a su nombre
4. En la parte superior derecha, **usa las flechas para desplazarte** por la lista de cambios
Nota: El texto añadido se resalta en otro color y el eliminado aparece tachado
5. Para volver a la versión que estás viendo, haz clic en **Restaurar esta revisión**
6. Para volver a la versión actual del documento, **haz clic en la flecha hacia atrás** situada en la parte superior izquierda

En la página siguiente, observe gráficamente la Revisión del Historial

Tomado de <http://bit.ly/20vEuRa>

Sección 4: Desarrollo de Procedimientos

Revision history	
November 12, 7:04 PM	<div><div></div> Mateo Agudelo Toro</div> <div><div></div> Catalina Patiño</div>
November 10, 6:03 PM	<div><div></div> Catalina Patiño</div>
November 10, 5:29 PM	<div><div></div> Catalina Patiño</div>
November 10, 4:28 PM	<div><div></div> Catalina Patiño</div>
November 3, 8:21 PM	<div><div></div> Catalina Patiño</div>
November 3, 6:30 PM	<div><div></div> Catalina Patiño</div> <div><div></div> Mateo Agudelo Toro</div>
November 3, 4:20 PM	<div><div></div> Catalina Patiño</div>
November 3, 3:34 PM	<div><div></div> Mateo Agudelo Toro</div> <div><div></div> Catalina Patiño</div>
November 3, 9:17 AM	<div><div></div> Catalina Patiño</div>
October 28, 7:52 PM	<div><div></div> Catalina Patiño</div> <div><div></div> Mateo Agudelo Toro</div>
October 28, 2:14 PM	<div><div></div> Mateo Agudelo Toro</div> <div><div></div> Catalina Patiño</div>
October 28, 8:02 AM	
<input checked="" type="checkbox"/> Show changes	
Show more detailed revisions	

4.23 Cómo generar el historial de cambios en el código de un repositorio que está en svn

Se hace ejecutando el siguiente comando:

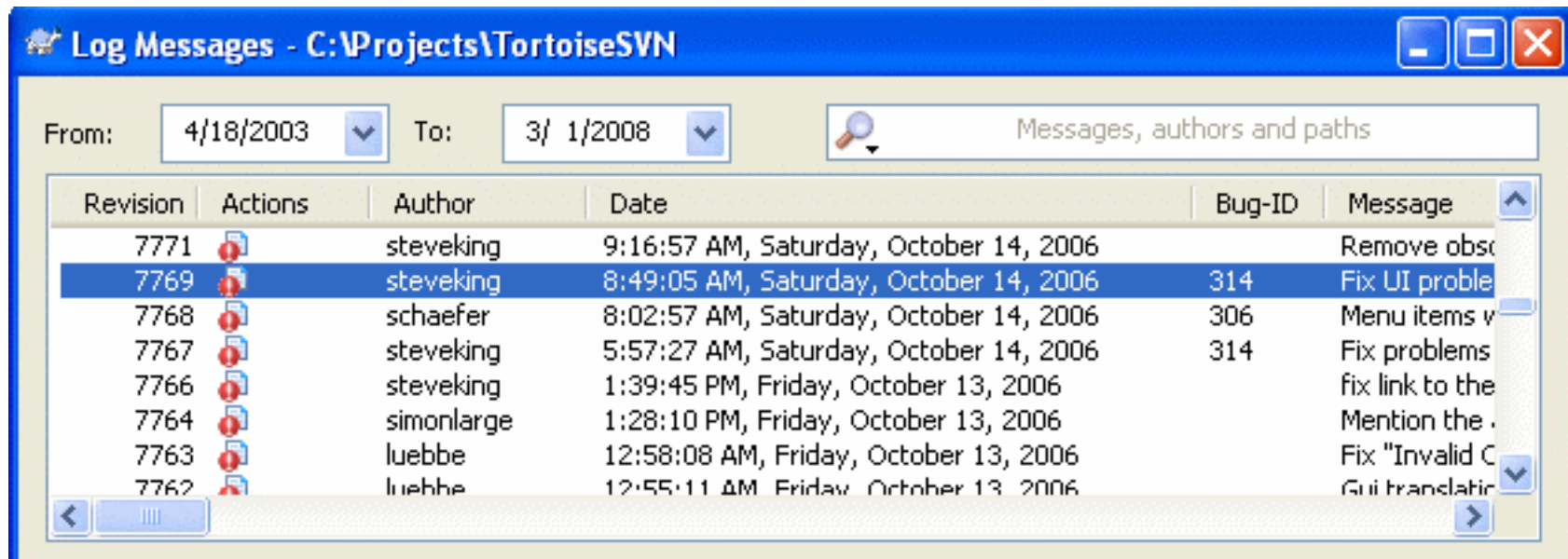
```
svn log -v
```

Y Se obtiene algo así:

```
-----  
r33 | estudiante1 | 2012-02-28 16:10:41 +0600 (Вт, 28 фев 2012) | 1 line  
  
One more change  
-----  
r32 | estudiante2 | 2011-12-27 17:37:31 +0600 (Вт, 27 дек 2011) | 1 line  
  
Cleanups  
-----  
r31 | estudiante1 | 2011-12-27 17:29:00 +0600 (Вт, 27 дек 2011) | 1 line  
  
Purification  
-----  
r30 | estudiante2 | 2011-10-19 16:23:52 +0600 (Ср, 19 окт 2011) | 1 line  
  
Try fix FS #2  
-----  
r29 | estudiante1 | 2011-10-19 16:18:43 +0600 (Ср, 19 окт 2011) | 1 line
```

Sección 4: Desarrollo de Procedimientos

Si utilizan *svn tortoise*, se verá así



¿Alguna inquietud?

CONTACTO

Docente Mauricio Toro Bermúdez

Teléfono: (+57) (4) 261 95 00 Ext. 9473

Correo: mtorobe@eafit.edu.co

Oficina: 19- 627

Agende una cita con él a través de <http://bit.ly/2gzVg10> , en la pestaña *Semana*.
Si no da clic en esta pestaña, parecerá que toda la agenda estará ocupada.



Realizadores

**Este texto fue escrito y corregido
por:**

**Mauricio Toro Bermúdez
Luis A. Gallego
Luisa Fernanda Alzate Sánchez**