GUÍA 3 – DESARROLLO DE SOFTWARE

Oscar Andres Velandia Castellanos

Universidad EAN

Dilsa Enith Triana Martinez

Septiembre 13 de 2024

**Link repositorio actividad 2:** [**https://github.com/elgordowsky/guia3desarrollodesoftware.git**](https://github.com/elgordowsky/guia3desarrollodesoftware.git)

ACTIVIDAD 1 – ESTRUCTURAS DE DATOS NO LINEALES

**SERIE DE PROBLEMAS DE CÁLCULO DE COMPLEJIDAD COMPUTACIONAL**

1. Para cada uno de los tiempos que toma un algoritmo en terminar, demostrar que cada uno es de la complejidad dada:

Respuesta:

Para el desarrollo de este punto debemos tener en cuenta que, para demostrar que un algoritmo tiene complejidad f(n), se debe buscar un punto n0 sobre el eje del tamaño del problema, a partir del cual se pueda garantizar el acotamiento de TA(n) por la función f(n), ignorando los factores constantes de esta última (sólo interesa la forma de la función y no su valor exacto). (Villalobos, 1996)

Por lo tanto, sólo con establecer cuál es la mayor complejidad presente en el algoritmo, esta determinará la complejidad total del mismo. Como podemos observar en la imagen 1, dependiendo del grado de complejidad se afectará la eficiencia de este, siendo los más eficientes los de grado 1 y grado n, y los de mayor complejidad y menos eficientes los de grado n3 y 2n.

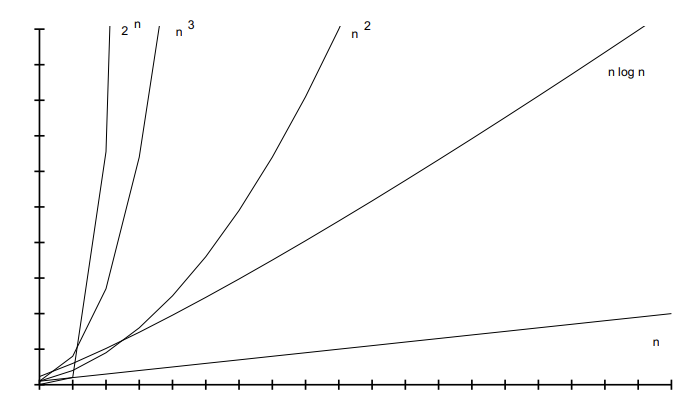


Imagen 1 (Villalobos, 1996)

Teniendo en cuenta lo anterior, podemos establecer para cada ejercicio lo siguiente:



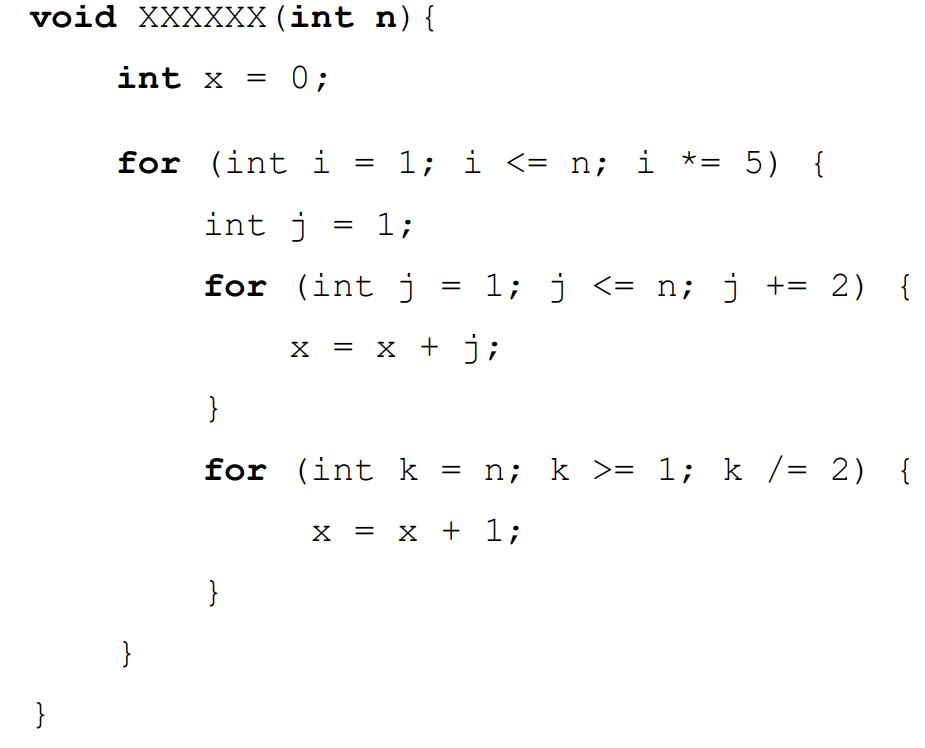
Este algoritmo esta conformado por tres partes una de complejidad *n3*, una de complejidad n2 y otra de complejidad 1. Por tanto, al ser *n3* el parámetro de mayor complejidad dentro del algoritmo, se puede establecer que este algoritmo tiene una complejidad general de *O(n3).*

1. , esta ecuación puede reescribirse así:

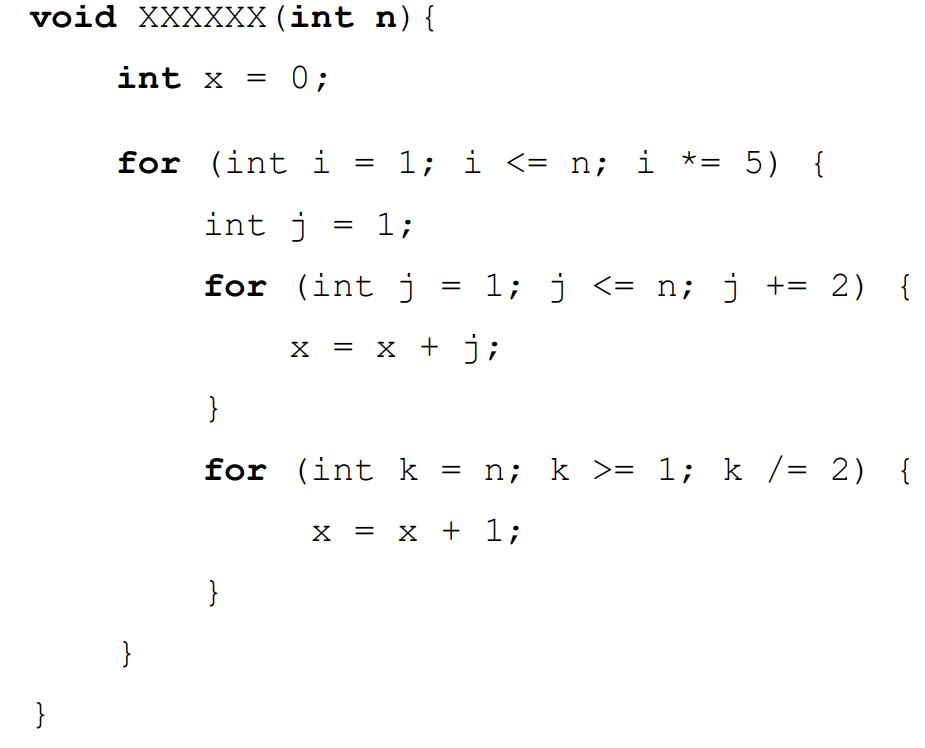
Donde podemos ver que se compone de una complejidad *n5*, una complejidad valor 1 con constante 17 que podemos considerar como 1, y una complejidad *n1/2*. Como el mayor grado de complejidad esta dado por *n5* podemos concluir que todo el algoritmo tiene complejidad *O(n5)*.

Este algoritmo está conformado también por tres partes, una con complejidad *nlogn,* otra con complejidad *n4*, y otra con complejidad n. Como la mayor complejidad esta dada por *n4*, podemos establecer que la complejidad de este algoritmo es de *O(n4)*.

1. Calcule paso a paso la complejidad del siguiente algoritmo:



Respuesta: para encontrar la complejidad del algoritmo dado debemos tener en cuenta que éste está compuesto por tres ciclos, en donde dos de ellos están anidados en el ciclo principal. Por tanto, debemos determinar la complejidad de cada ciclo para encontrar la complejidad total del algoritmo.

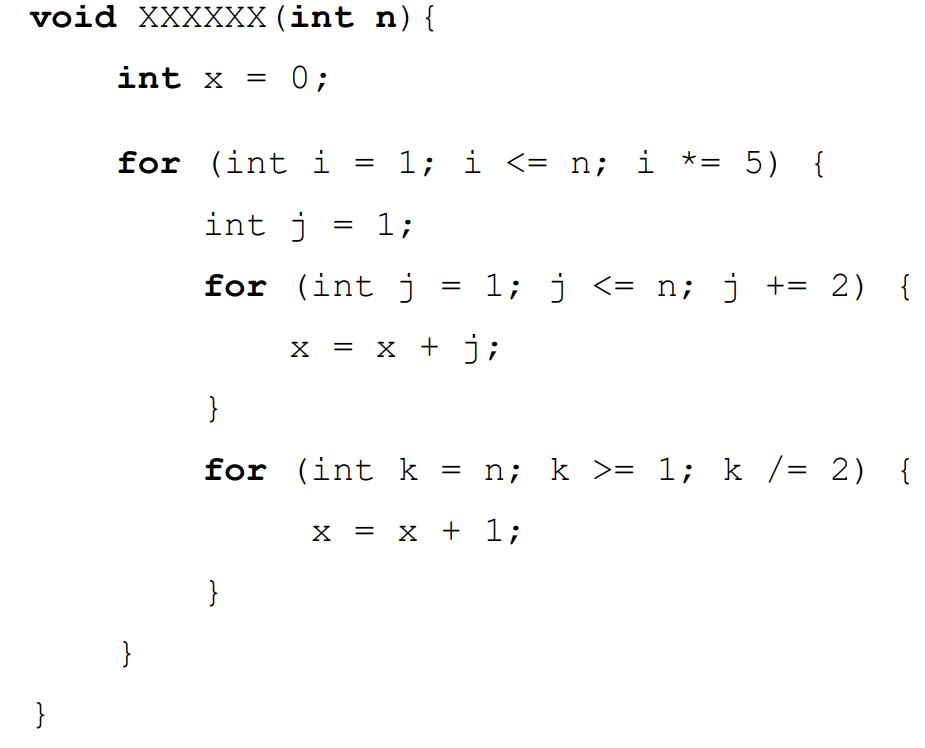


En el primer ciclo *for* encontramos que inicia con i = 1 y se va multiplicando por 5 en cada iteración, por tanto, para saber el número de iteraciones que se realizarán para cada n, se puede aplicar *Log5(n*), lo que nos permite identificar que la complejidad de este primer ciclo está dada por *O (Log n).*

A white paper with black text and numbers

Description automatically generated

Continuando con el segundo ciclo, encontramos que éste comienza en j=1 y aumenta de 2 en 2 hasta llegar a n. Por tanto, al ir en 2 en 2 tomará la mitad del tiempo para llegar a n, por lo que las iteraciones necesarias serán *n/2* veces. Ignorando la constante vemos que es grado n, por tanto, la complejidad de este segundo ciclo es *O(n).*



Finalmente, para el tercer ciclo encontramos que se inician en *k=n* y se divide por 2 en cada iteración hasta llegar a un valor mayor o igual que 1 (*k>=1*). Por tanto, para saber la cantidad de iteraciones necesarias para que n al dividirse por 2 sea mayor o igual que 1, podemos aplicar *Log2(n).* Lo que nos lleva a determinar que la complejidad de este ciclo es *O (Log n).*

Así, tenemos que las complejidades para cada ciclo son:

Ciclo 1 (Ciclo externo): complejidad *O (Log n)*

Ciclo 2 (Ciclo interno): complejidad *O (n)*

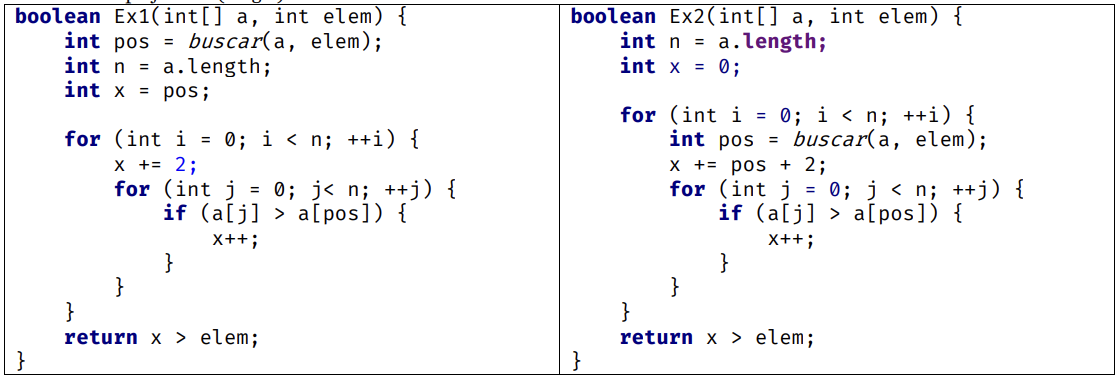
Ciclo 3 (Ciclo interno): complejidad *O (Log n)*

Como los ciclos 2 y 3, que corresponden a ciclos internos son independientes entre sí, debemos escoger el mayor grado de complejidad existente entre ambos, que para este caso corresponde a *O (n)*, dado que *O(n)* es de mayor complejidad que *O (Log n).*

Ya conociendo la complejidad de los ciclos internos, multiplicamos este valor de complejidad por la complejidad del ciclo externo, obteniendo:

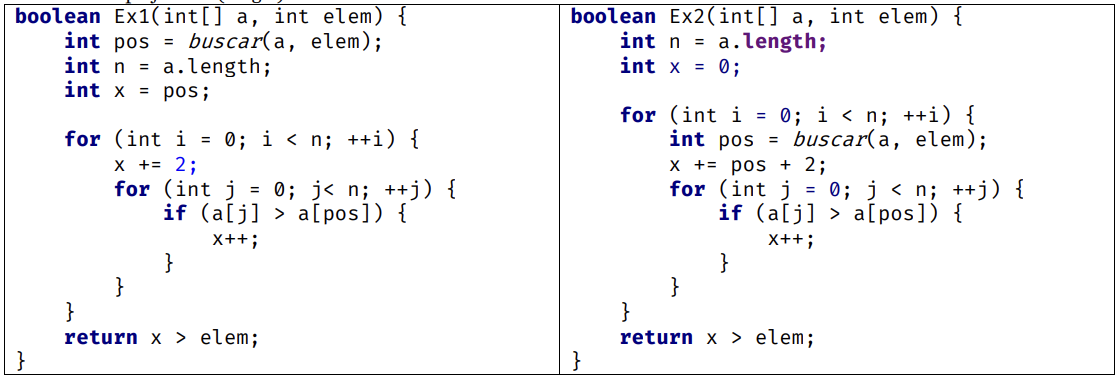
Así, tenemos que la complejidad del algoritmo dado es de orden *O (n Log n)*.

1. Suponga que se tienen los siguientes dos algoritmos para resolver el mismo problema, y suponga que la función buscar tiene complejidad O (n log n). Indique con cual de los dos algoritmos se queda para resolver el problema. Justifique su respuesta.



Respuesta:

Para resolver el problema debemos escoger el algoritmo que tenga mayor eficiencia tanto en tiempo como en espacio, por lo tanto, debemos encontrar el algoritmo de menor complejidad. Para esto analizaremos la complejidad de cada algoritmo dado.

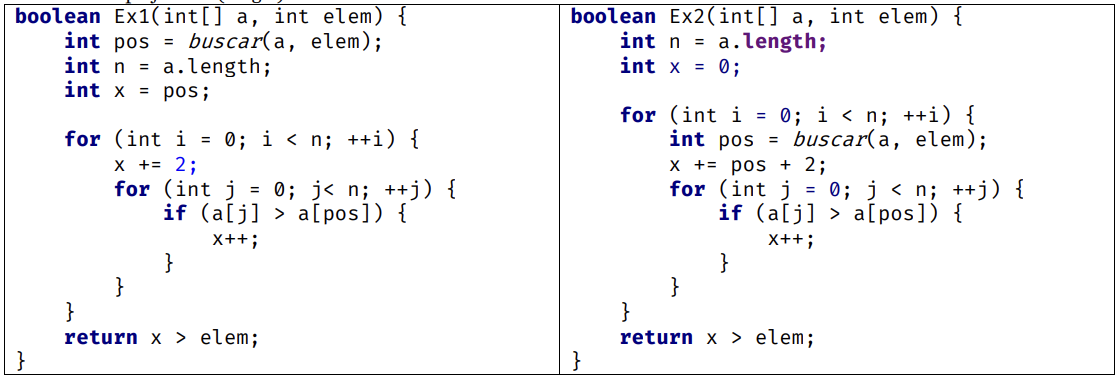


Para el primer algoritmo (Ex1), tenemos que primero se realiza un llamado a la función buscar, que de acuerdo con el enunciado tiene una complejidad de *O (n Log n)*.

Luego encontramos dos ciclos consecutivos, uno dentro de otro, para el ciclo externo encontramos que inicia en i=0 y va aumentando de 1 en 1 hasta llegar a n, realizando n iteraciones, por tanto, su complejidad es *O (n).* En el ciclo interno, encontramos que inicia en j=0 y va aumentando de 1 en 1 hasta llegar a n, realizando n iteraciones, por tanto, su complejidad también es *O (n)*. Como los dos ciclos están anidados, es decir tenemos un ciclo (interno) dentro de otro ciclo (externo), debemos multiplicar la complejidad de ambos ciclos para encontrar su complejidad total. Así tenemos que:

La complejidad del ciclo total es de *O (n2).*

Hasta ahora tenemos que, la complejidad de la función buscar es O (n Log n) y la complejidad del ciclo total es O (n2), para determinar la complejidad total del algoritmo debemos escoger el grado de mayor complejidad, que en este caso corresponde a *O (n2).* Por tanto, la complejidad del algoritmo Ex1 es de *O (n2)*.



Para el segundo algoritmo (Ex2), tenemos que la complejidad del algoritmo esta dada por la complejidad del ciclo completo. Realizando el análisis tenemos:

Para el ciclo externo, tenemos que se inicia en i=0 y va aumentando de 1 en 1 hasta llegar a n, realizando n iteraciones, por tanto, su complejidad es *O (n).* Luego dentro del ciclo, se llama la función buscar que de acuerdo con el enunciado tiene una complejidad de *O (n Log n).*

Para el ciclo interno, tenemos que inicia en j=0 y va aumentando de 1 en 1 hasta llegar a n, realizando n iteraciones, por tanto, su complejidad también es *O (n)*. Para determinar la complejidad al interior del ciclo externo, debemos escoger la mayor complejidad entre la función buscar *O (n log n)* y el ciclo interno *O (n)*, que para este caso es la complejidad *O (n log n)* la de mayor grado.

Para calcular la complejidad total, debemos multiplicar la complejidad del ciclo externo *O (n)* con la complejidad interna del ciclo *O (n Log n)*, encontrando que:

La complejidad total del algoritmo Ex2 es *O (n2 Log n)*.

Ahora, comparando la complejidad de ambos algoritmos tenemos que:

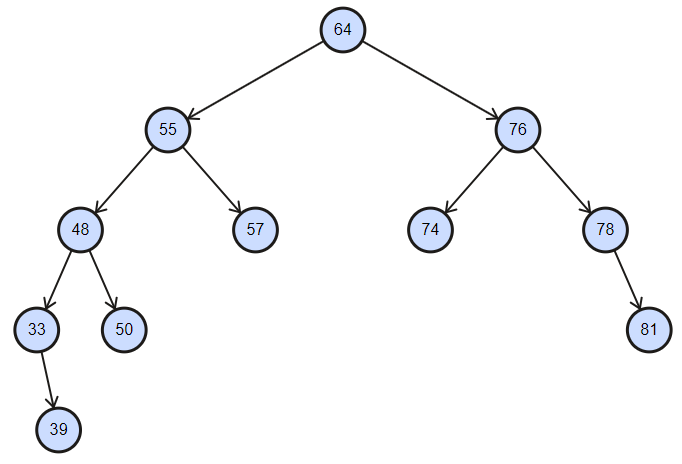
Algoritmo Ex 1: complejidad *O (n2)*

Algoritmo Ex 2: complejidad *O (n2 Log n)*

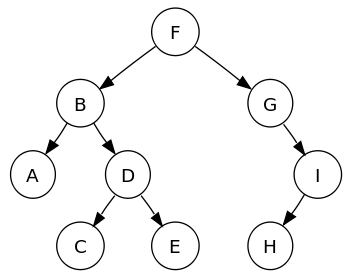
El algoritmo de menor complejidad es el Algoritmo *Ex1 O (n2),* por tanto, el mejor algoritmo para resolver el problema es el algoritmo Ex1 dado que es más eficiente.

**SERIE DE PROBLEMAS SOBRE ÁRBOLES BINARIOS**

1. Para el siguiente árbol binario, presente:



1. Peso: 11, porque tiene 11 nodos.
2. Altura: la altura se define como el número de niveles desde la raíz (64) hasta la hoja más profunda (39), que para este caso es 4.
3. Hojas: el árbol tiene 5 hojas (39, 50, 57, 74 y 81)
4. Una rama: 64 – 55 – 48 – 33 – 39
5. Recorrido en inorden: 39, 33, 48, 50, 55, 57, 64, 74, 76, 78, 81.
6. Recorrido en preorden: 64, 55, 48, 33, 39, 50, 57, 76, 74, 78, 81.
7. Recorrido en postorden: 39, 33, 50, 48, 57, 55, 74, 81, 78, 76, 64.
8. Para el siguiente árbol, indique:

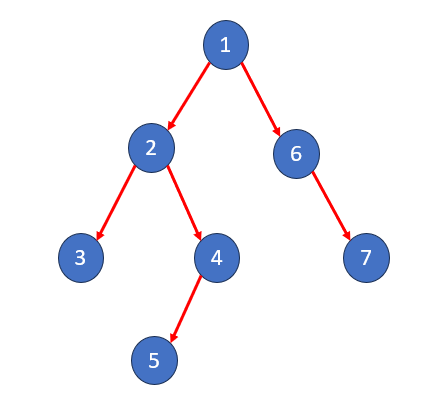


1. Altura: la altura se define como el número de niveles desde la raíz (F) hasta la hoja más profunda (C, E, H), que para este caso es 3.
2. Número de niveles: 4
3. Ancestro común de la E y la A: El ancestro común más cercano de los nodos E y A es el nodo **B**, ya que es el primer nodo compartido.
4. Peso del árbol izquierdo de la F: El árbol izquierdo de F contiene los nodos B, A, D, C, y E, por lo tanto, su peso es 5.
5. Recorrido en inorden: A – B – C – D – E – F – G – H – I
6. Recorrido en preorden: F – B – A – D – C – E – G – I – H
7. Recorrido en postorden: A – C – E – D – B – H – I – G – F
8. Recorrido por niveles: El recorrido por niveles de izquierda a derecha es F – B – G – A – D – I – C – E – H
9. Hojas: el árbol tiene 4 hojas A, C, E, H.
10. Reconstruya el árbol binario que posee los siguientes recorridos:

Preorden: 1 – 2 – 3 – 4 – 5 – 6 – 7

Inorden: 3 – 2 – 5 – 4 – 1 – 6 – 7

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | INORDEN | | | | | | |
|  |  | 3 | 2 | 5 | 4 | 1 | 6 | 7 |
| PREORDEN | 1 |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |

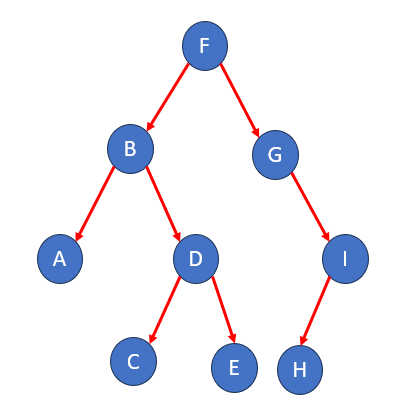


1. Reconstruya el árbol binario que posee los siguientes recorridos:

Postorden: A – C – E – D – B – H – I – G – F

Inorden: A – B – C – D – E – F – G – H – I

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | INORDEN | | | | | | | | |
|  |  | A | B | C | D | E | F | G | H | I |
| POSTORDEN | F |  |  |  |  |  |  |  |  |  |
| G |  |  |  |  |  |  |  |  |  |
| I |  |  |  |  |  |  |  |  |  |
| H |  |  |  |  |  |  |  |  |  |
| B |  |  |  |  |  |  |  |  |  |
| D |  |  |  |  |  |  |  |  |  |
| E |  |  |  |  |  |  |  |  |  |
| C |  |  |  |  |  |  |  |  |  |
| A |  |  |  |  |  |  |  |  |  |

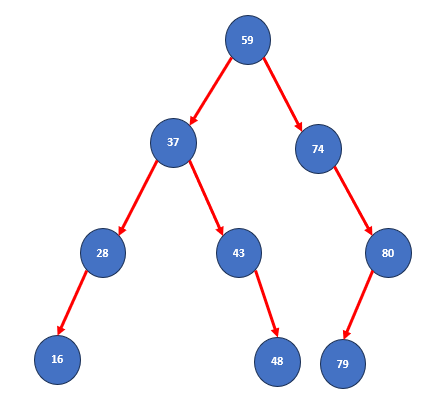


1. Reconstruya el árbol binario que posee los siguientes recorridos:

Preorden: 59 – 37 – 28 – 16 – 43 – 48 – 74 – 80 – 79

Inorden: 16 – 28 – 37 – 43 – 48 – 59 – 74 – 79 – 80

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | INORDEN | | | | | | | | |
|  |  | 16 | 28 | 37 | 43 | 48 | 59 | 74 | 79 | 80 |
| PREORDEN | 59 |  |  |  |  |  |  |  |  |  |
| 37 |  |  |  |  |  |  |  |  |  |
| 28 |  |  |  |  |  |  |  |  |  |
| 16 |  |  |  |  |  |  |  |  |  |
| 43 |  |  |  |  |  |  |  |  |  |
| 48 |  |  |  |  |  |  |  |  |  |
| 74 |  |  |  |  |  |  |  |  |  |
| 80 |  |  |  |  |  |  |  |  |  |
| 79 |  |  |  |  |  |  |  |  |  |



**Actividad 2**

**Link repositorio:** [**https://github.com/elgordowsky/guia3desarrollodesoftware.git**](https://github.com/elgordowsky/guia3desarrollodesoftware.git)

El proyecto "Central de Pacientes" es un sistema diseñado para gestionar información de pacientes en clínicas, permitiendo el manejo de datos personales y médicos de los pacientes. A nivel de código, el proyecto está organizado en términos de interfaz y lógica de negocio, facilitando la gestión centralizada de información de pacientes en un entorno clínico.

**Estructura y Organización**

Paquetes Principales:

**centralPacientes.interfaz:** Contiene las clases relacionadas con la interfaz gráfica del usuario para interactuar con el sistema. En la interfaz gráfica encontramos algunos detalles a destacar como:

Constantes:

MASCULINO y FEMENINO: Representan los géneros que puede tener un paciente.

Atributos de la Interfaz:

JTextField para el nombre y código del paciente.

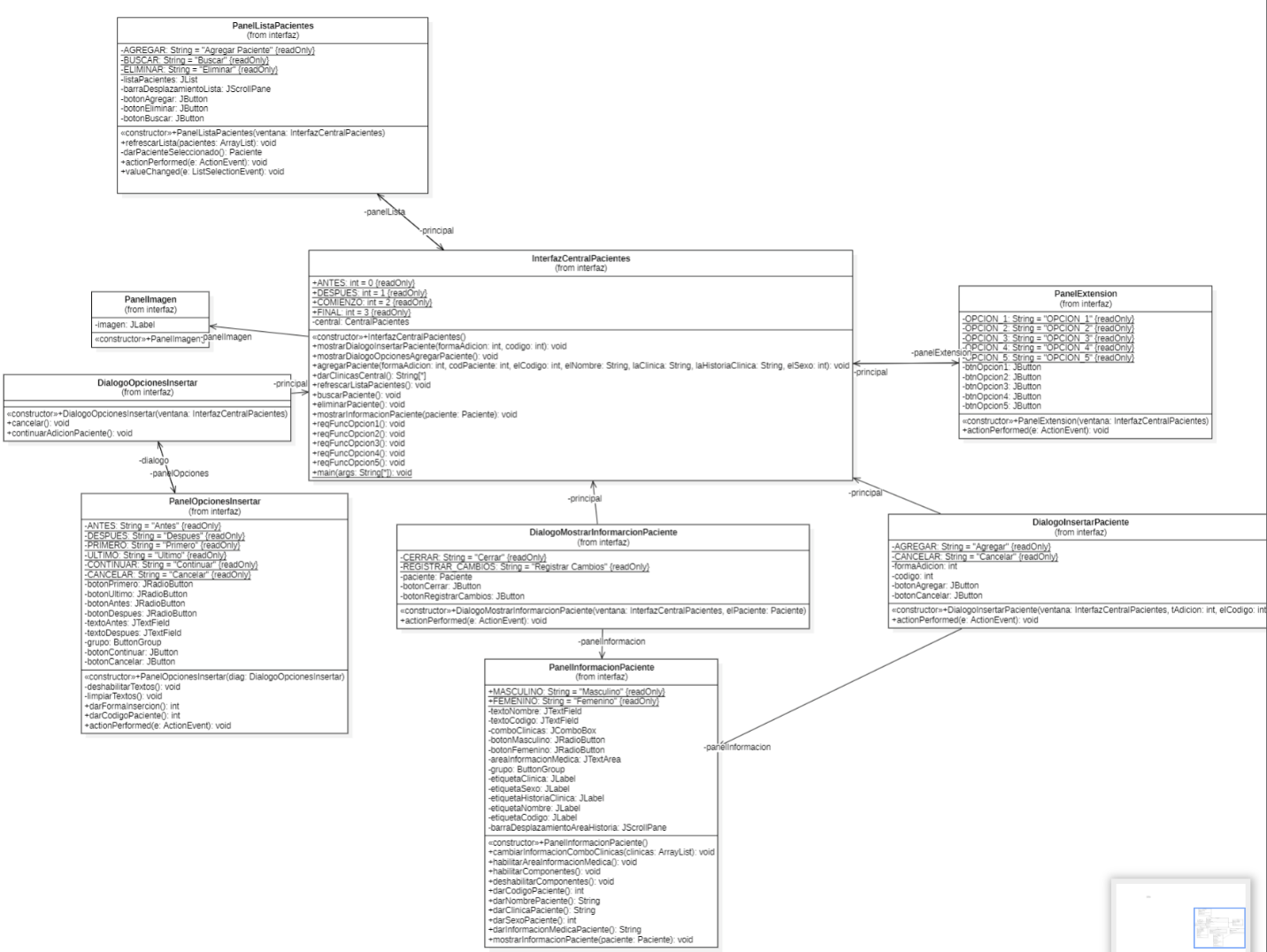
JComboBox para seleccionar clínicas.

JRadioButton para elegir el género.

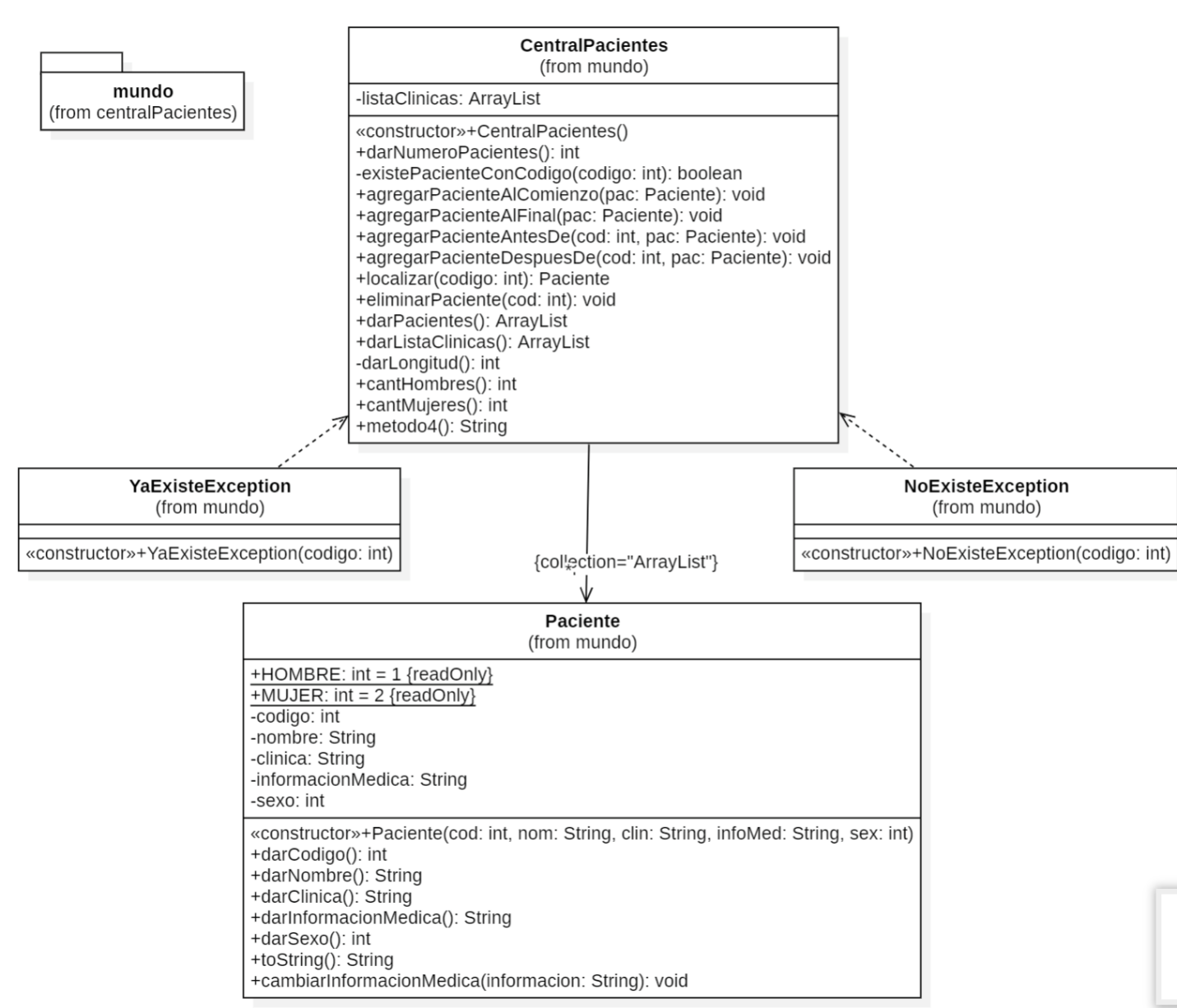
JTextArea para mostrar información médica.

Etiquetas (JLabel) para clínicas, género, historia clínica, nombre y código.

JScrollPane para agregar desplazamiento al área de historia clínica.



**centralPacientes.mundo:** Aquí se encuentran las clases del modelo que representan el dominio del problema, como la clase Paciente.



**Métodos desarrollados**

private boolean existePacienteConCodigo(int codigo) : este método se creo independiente a los marcados con TODO con el fin de retornar un boolean, indicando si existe el paciente con el código.

public void agregarPacienteAlComienzo(Paciente pac) : agrega el paciente con el con add y el parámetro (0,pac)

public void agregarPacienteAlFinal(Paciente pac) : usa el método add para añadir al final.

public void agregarPacienteAntesDe(int cod, Paciente pac) throws NoExisteException : con el código recibido se itera para encontrar la posición que será desplazada y añade en ella el nuevo paciente.

public void agregarPacienteDespuesDe(int cod, Paciente pac) throws NoExisteException : iteramos para encontrar la posición y sumando 1 situamos al nuevo paciente.

public Paciente localizar(int codigo) : con el código se busca un paciente en el arraylist de pacientes que coincida con el dato del código proporcionado.

public void eliminarPaciente(int cod) throws NoExisteException : elimina del array el paciente con el código proporcionado o activa NoExisteException.

public int cantHombres() : se itera en el array de pacientes, para conocer la cantidad de pacientes que coinciden con género HOMBRE.

public int cantMujeres() : se itera en el array de pacientes, para conocer la cantidad de pacientes que coinciden con género MUJER.

public String metodo4() : este es el método final que se ocupa calcular cual es la clínica más ocupada de entre la 6 opciones sugeridas, para ello se itera sobre el array de pacientes con un foreach y cada valor hace una comparación mediante un switch en el cual en cada case aumenta el valor en cada contador; luego realiza comparaciones usando una variable asignada para almacenar un String con la clínica con más pacientes y retorna variable que almacena el nombre de la clínica con más pacientes.

References

Villalobos S., J. A. (1996). *Diseño y Manejo de Estructura de Datos en C.*Capítulo 0, sección 0.3 y Capítulo 1, 2, 3, 4 y 6 Bogotá, Colombia: McGraw-Hill. Recuperado de [https://bit.ly/EDCVillalobosLinks to an external site.](https://bit.ly/EDCVillalobos)