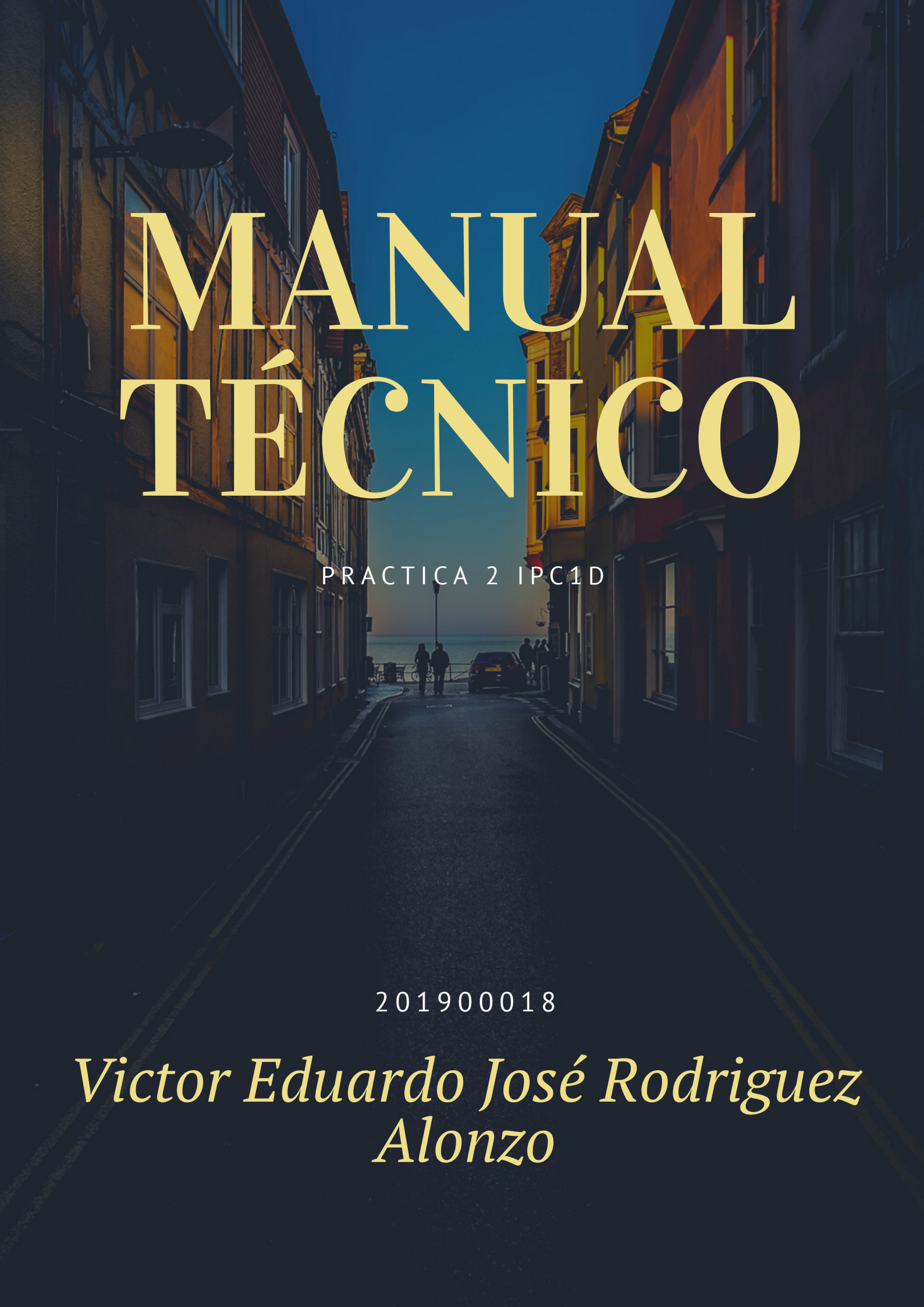
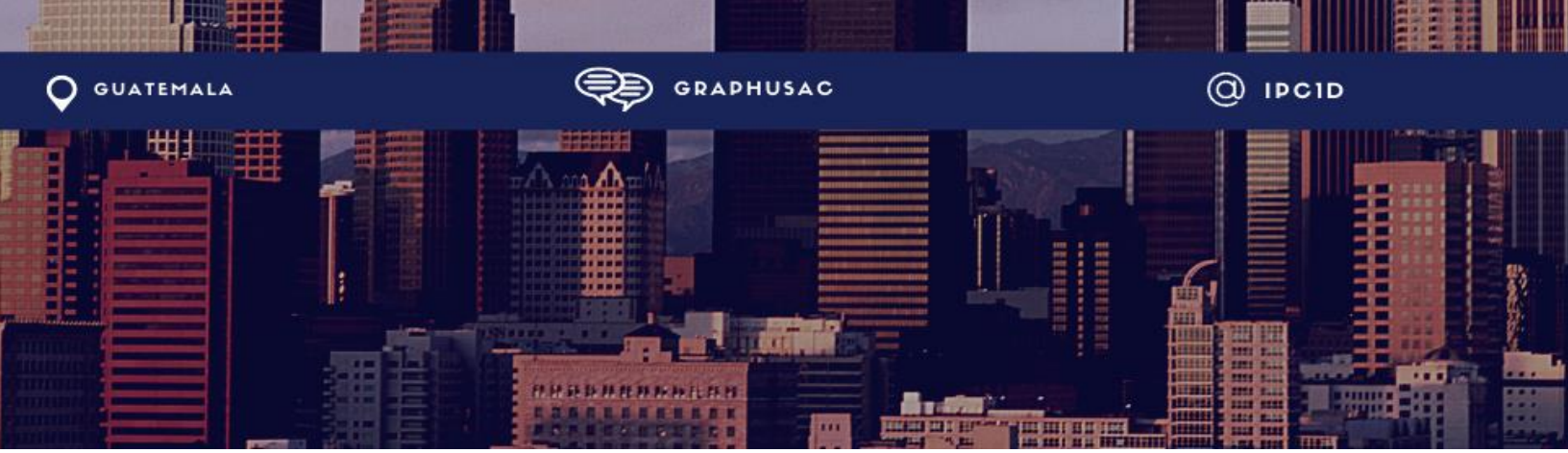


MANUAL TÉCNICO

PRACTICA 2 IPC1D

201900018

Victor Eduardo José Rodriguez Alonzo



INTRODUCCIÓN

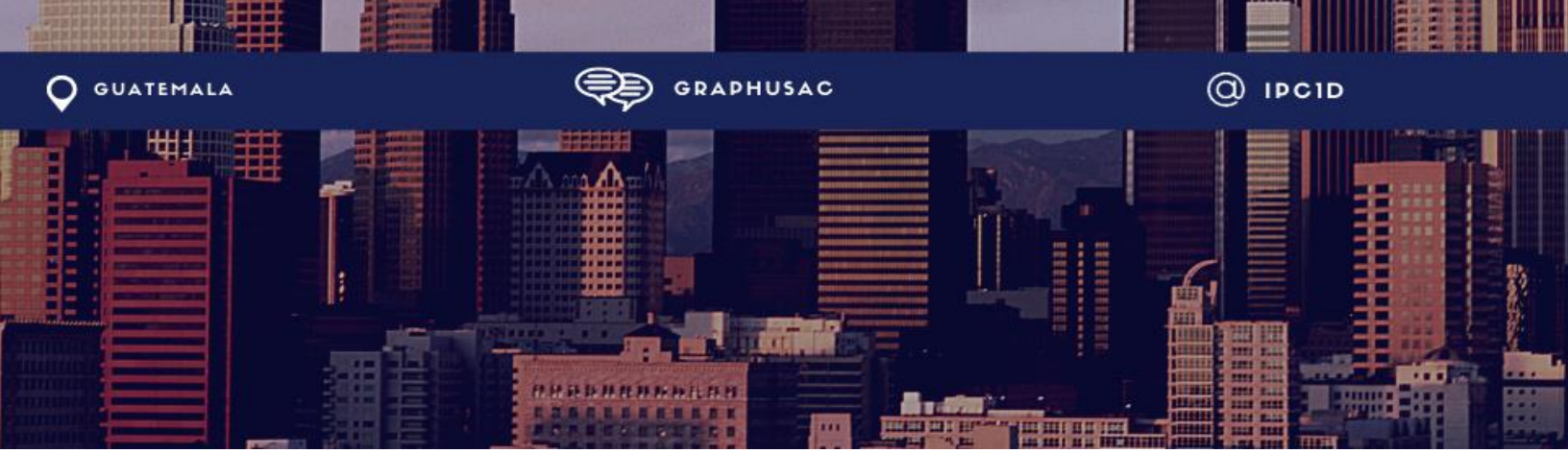
El siguiente manual es creado para poder ayudar al programador a entender la estructura de nuestra graficadora GraphUsac, la cual se realizo en un programa de programación como lo es Java, se mostrará código importante del mismo y así se mostrará un diagrama de flujo e interpretación del método de ordenamiento Quicksort, espero el siguiente Manual sea de toda su ayuda.

OBJETIVO

Mostrar el código mas importante de la graficadora para que el programador que lea este manual pueda utilizarlo de la forma que desee y pueda crear o mejorar este mismo.

OBJETIVOS ESPECIFICOS.

-Mostrar código en Java de interfaz gráfica, métodos, ordenamiento, reporte.
-Especificar detalladamente el uso del ordenamiento QuickSort.
-Especificar por medio de un diagrama de flujo el ordenamiento QuickSort.



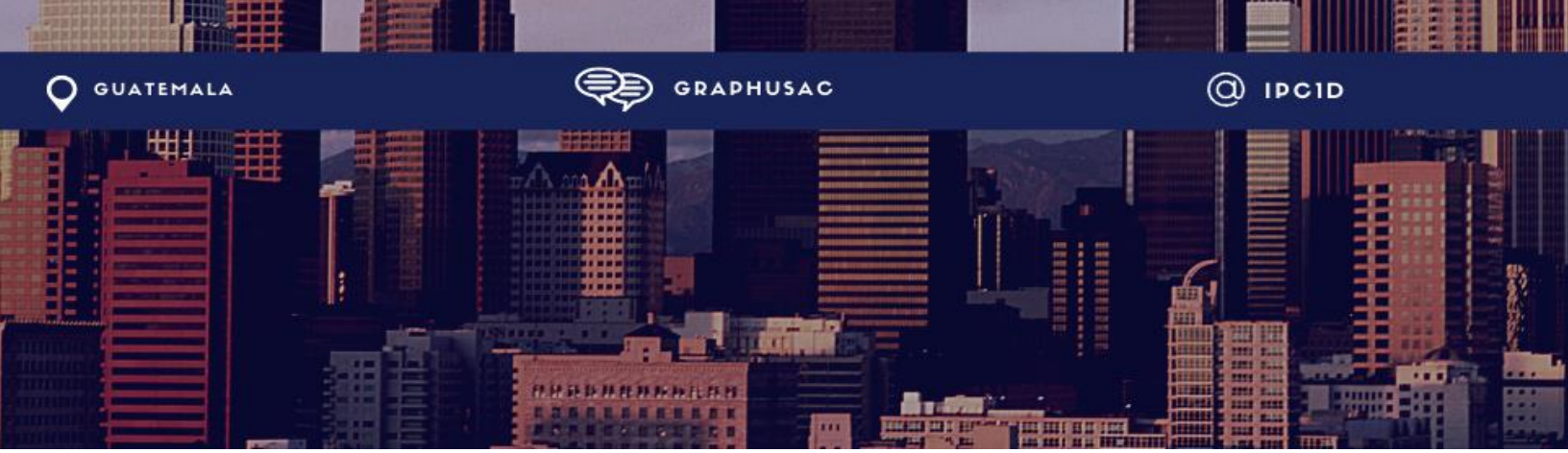
ESPECIFICACIÓN TÉCNICA

➢ Requisitos de Hardware.

- Computadora de escritorio o portátil.
- Mínimo 4GB de Memoria RAM.
- 250GB de Disco Duro o Superior.
- Procesador Inter Core I3 o Superior
- Resolución gráfica mínimo 1024 x 728 pixeles.

➢ Requisitos de Software.

- Sistema Operativo Windows 10 o Superior.
- Tener instalado Java Runtime Enviroment (JRE) versión 15.2.
- Tener instalado Java Development Kit (JDK) versión 15.2.
- Lenguaje de Programación JAVA.
- NetBeans IDE 12.6.
- Librería Externa itextpdf.
- Librería Externa gson.
- Librería Externa JFreeChart.
- Librería Externa JCommon.
- Librería Interna Javax.Swing.
- Librería Interna IO.
- Acrobat Reader DC o algún lector PDF compatible.



ORDENAMIENTO QUICKSORT

El algoritmo trabaja de la siguiente forma:

- Elegir un elemento del conjunto de elementos a ordenar, al que llamaremos pivote.
- Resituar los demás elementos de la lista a cada lado del pivote, de manera que a un lado queden todos los menores que él, y al otro los mayores. Los elementos iguales al pivote pueden ser colocados tanto a su derecha como a su izquierda, dependiendo de la implementación deseada. En este momento, el pivote ocupa exactamente el lugar que le corresponderá en la lista ordenada.
- La lista queda separada en dos sublistas, una formada por los elementos a la izquierda del pivote, y otra por los elementos a su derecha.
- Repetir este proceso de forma recursiva para cada sublista mientras éstas contengan más de un elemento. Una vez terminado este proceso todos los elementos estarán ordenados.

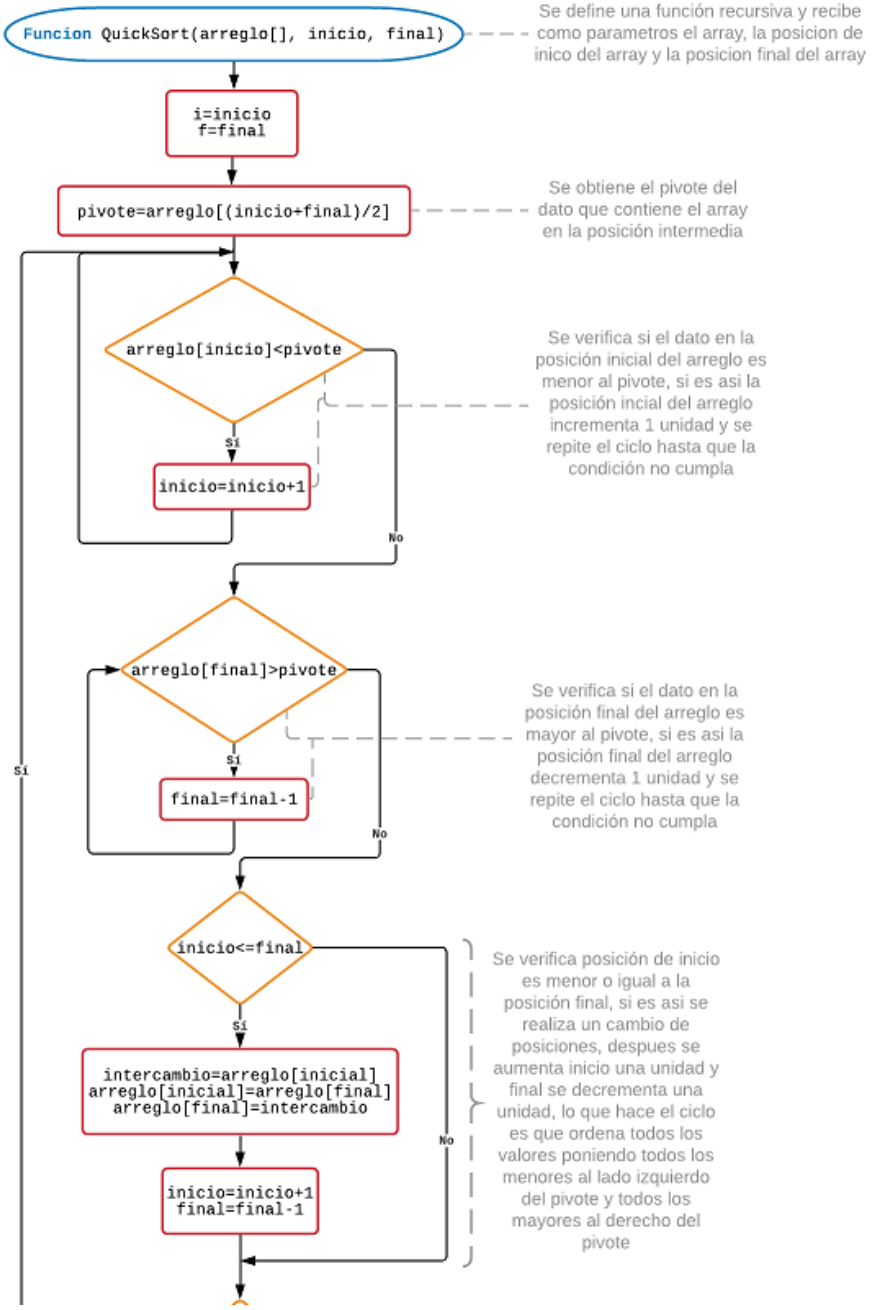
Como se puede suponer, la eficiencia del algoritmo depende de la posición en la que termine el pivote elegido.

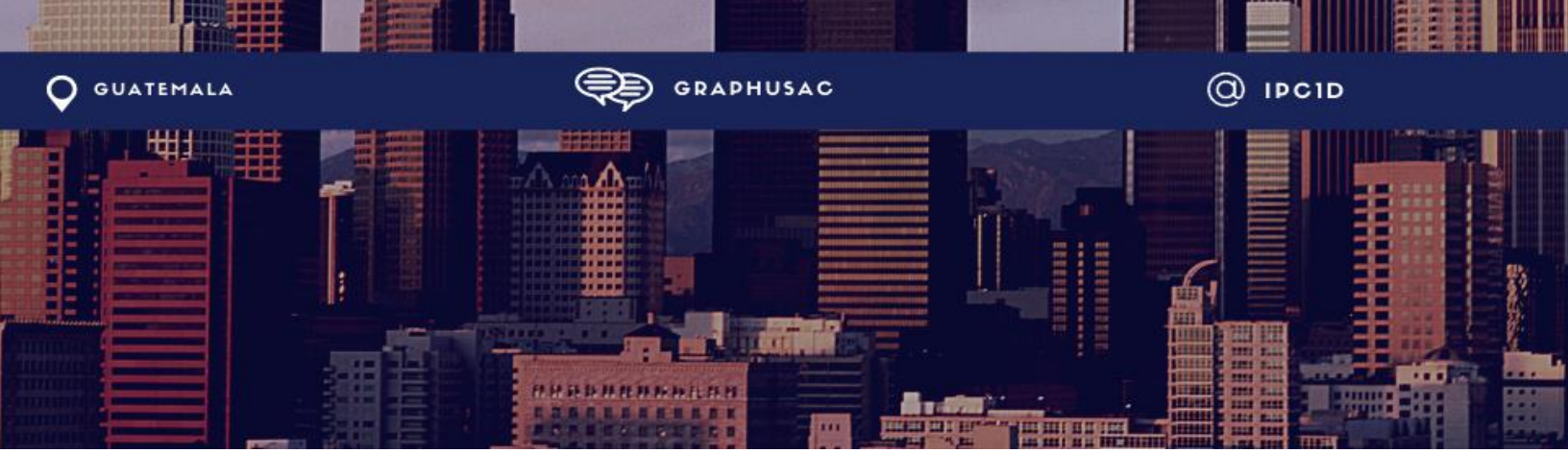
- En el mejor caso, el pivote termina en el centro de la lista, dividiéndola en dos sublistas de igual tamaño. En este caso, el orden de complejidad del algoritmo es $O(n \cdot \log n)$.
- En el peor caso, el pivote termina en un extremo de la lista. El orden de complejidad del algoritmo es entonces de $O(n^2)$. El peor caso dependerá de la implementación del algoritmo, aunque habitualmente ocurre en listas que se encuentran ordenadas, o casi ordenadas. Pero principalmente depende del pivote, si por ejemplo el algoritmo implementado toma como pivote siempre el primer elemento del array, y el array que le pasamos está ordenado, siempre va a generar a su izquierda un array vacío, lo que es ineficiente.

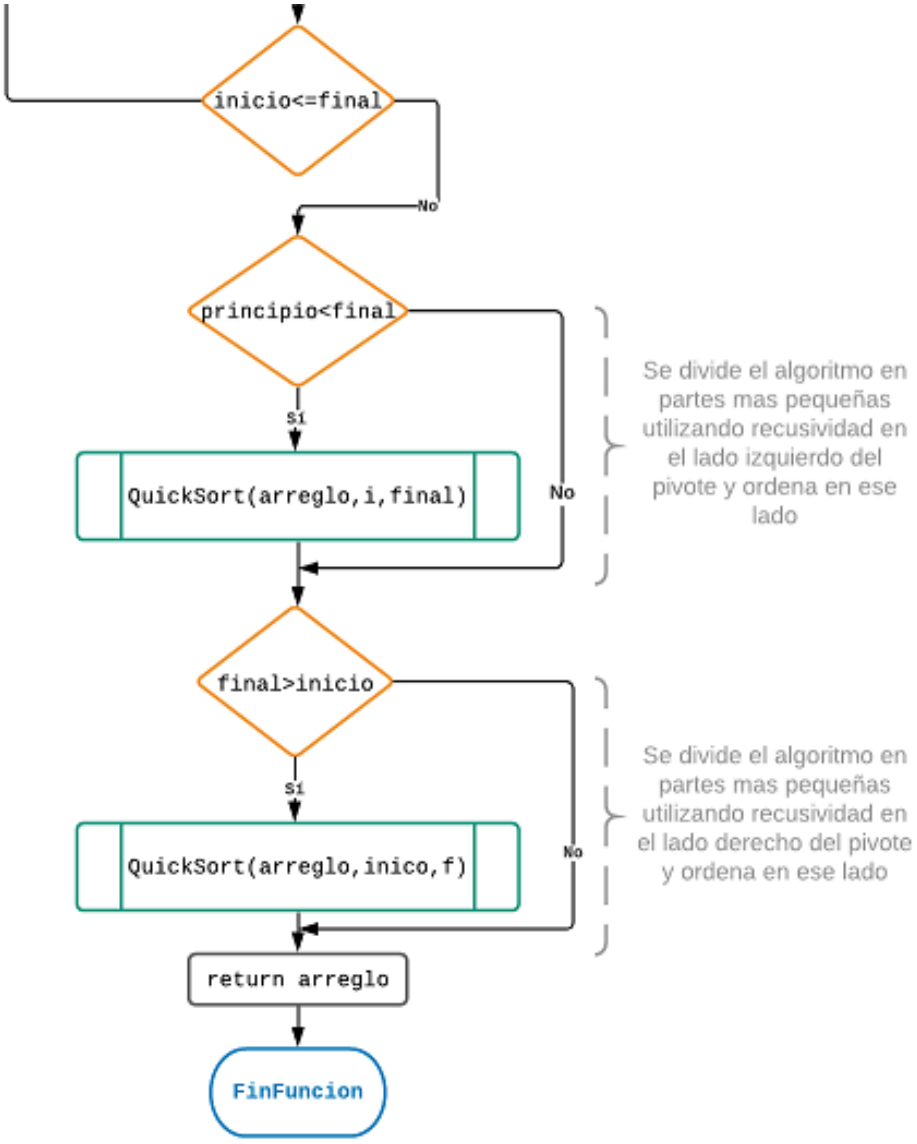
- En el caso promedio, el orden es $O(n \cdot \log n)$.

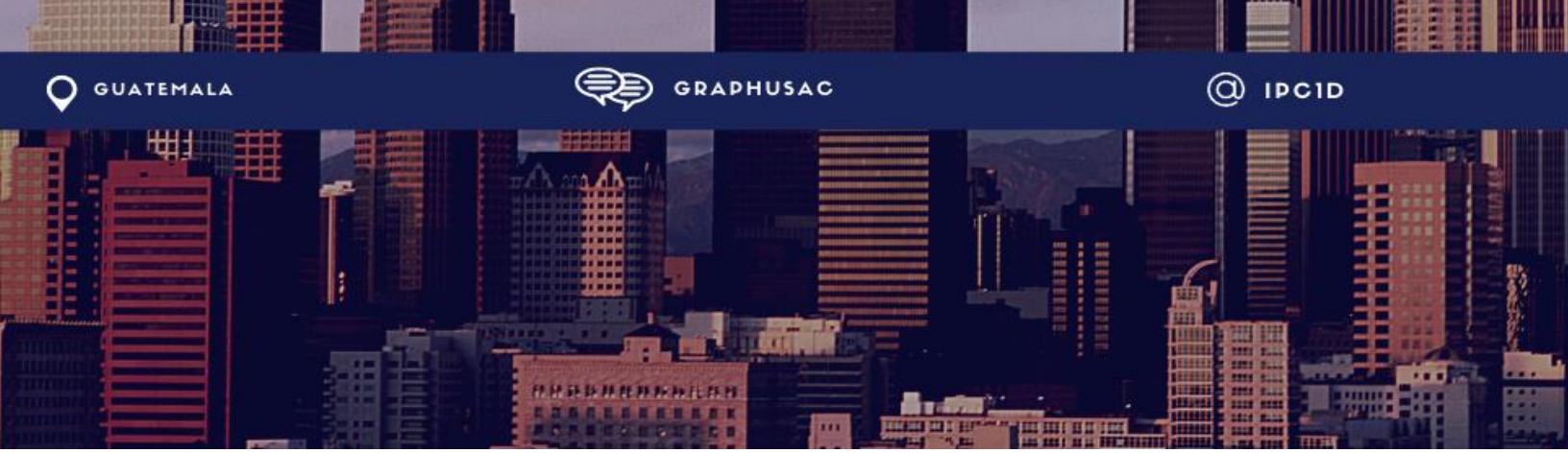
No es extraño, pues, que la mayoría de las optimizaciones que se aplican al algoritmo se centren en la elección del pivote.

DIAGRAMA DE FLUJO QUICKSORT



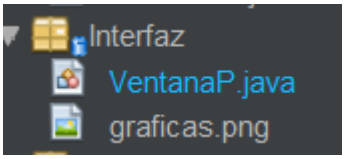



inicio<=final
No
principio<final
Si
QuickSort(arreglo,i,final)
No
Se divide el algoritmo en partes mas pequeñas utilizando recusividad en el lado izquierdo del pivote y ordena en ese lado
final>inicio
Si
QuickSort(arreglo,inico,f)
No
Se divide el algoritmo en partes mas pequeñas utilizando recusividad en el lado derecho del pivote y ordena en ese lado
return arreglo
FinFuncion



INTERFAZ GRÁFICA.

Es aquella que nos ayuda a que el usuario no vea toda la codificación al momento de querer correr el programa, más bien ayuda a que el usuario trabaje directamente solo con el manual usuario sin tener que preocuparse por la codificación, ahora especificaremos que se realizó en este apartado.

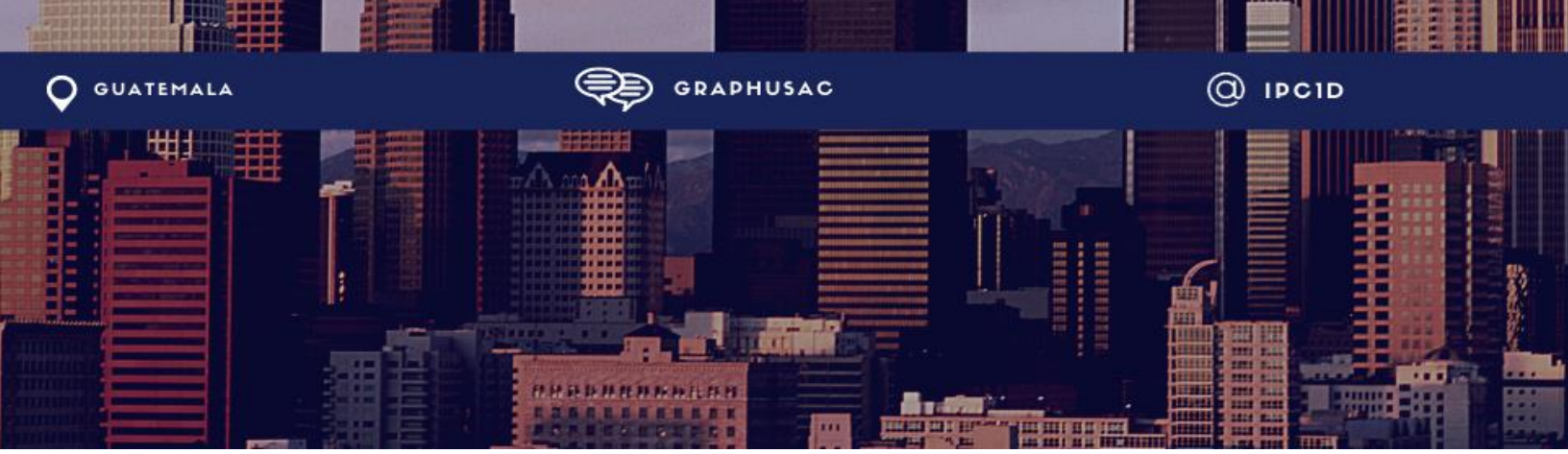


Creamos una carpeta dedicada solamente para la interfaz para este proyecto se trabajo En cada una de estas se trabajó lo que fueron JTextField, JBotton, JTable, JFrame, JPanel etc. Se mostrará cómo funciona cada uno, pero especificaremos que se tiene que realizar en todas según lo solicitado en Enunciado.

CREACIÓN DEL BOTON.

```
examinar = new JButton("EXAMINAR");
examinar.setBounds(10, 10, 150, 25);
examinar.setFont(new Font("Franklin Gothic Medium", Font.BOLD, 14));
examinar.setBackground(grisito);
examinar.setForeground(Color.white);
examinar.setVisible(true);
examinar.addActionListener(this);
this.add(examinar);
```

-Primera línea especifica la variable que se le asigna al botón y el titulo que lleva.
-Segunda línea son las dimensionales del botón.
-Tercera línea es el tipo de letra y tamaño
-Cuarta línea es el color que se le asigna
-Quinta línea muestra si es visible en la pantalla
-Sexta línea lo adhiere a la ventana.



CREACIÓN DE COMO ESCOGER UNA OPCIÓN.

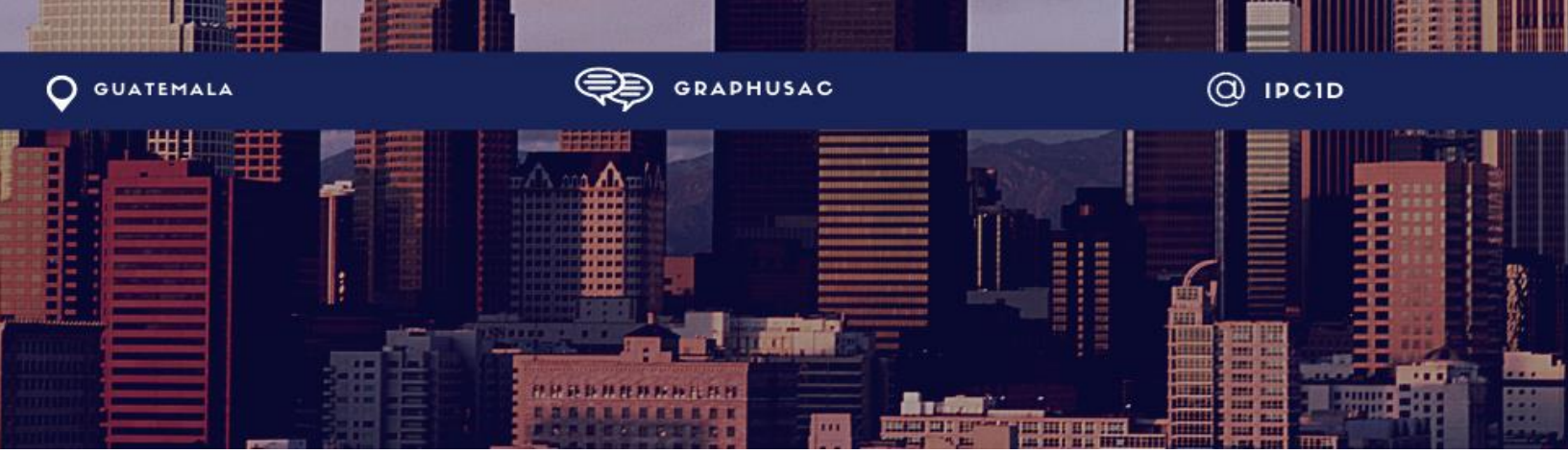
```
descendente = new JRadioButton("Descendente", false);
descendente.setFont(new Font("Franklin Gothic Medium", Font.BOLD, 14));
descendente.setBackground(grisito);
descendente.setForeground(Color.white);
descendente.setBounds(10, 150, 150, 25);
this.add(descendente);

ButtonGroup bg = new ButtonGroup();
bg.add(ascendente);
bg.add(descendente);
```

Para crear un JRadioButton se realiza de la misma manera que un botón solamente que con la diferente que se tienen que unir los botones como se ve en la imagen anterior.

CREACIÓN DE TITULO.

```
ruta = new JLabel("LINK:");
ruta.setBounds(210, 10, 100, 25);
ruta.setForeground(Color.white);
ruta.setFont(new Font("Verdana", Font.BOLD, 12));
ruta.setVisible(true);
this.add(ruta);
```

Esto nos sirve para colocar un título de lo que necesitemos en la ventana.



CREACIÓN DE PANEL.

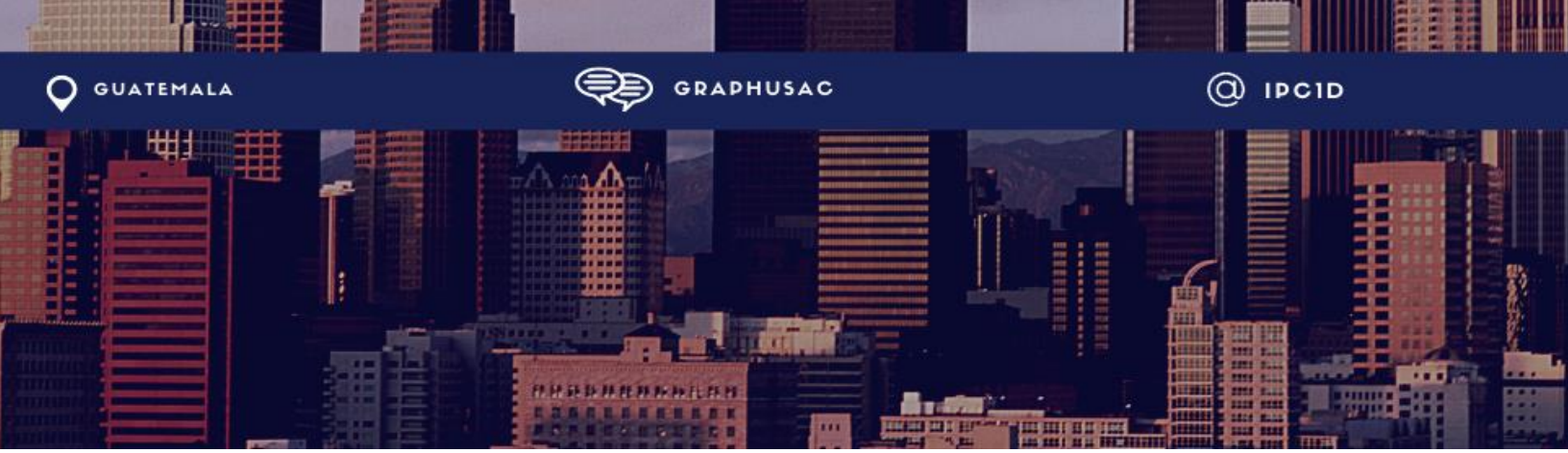
```
Color blanco = new Color(251, 252, 252);
cuadro = new JPanel();
cuadro.setBounds(10, 180, 1300, 500);
cuadro.setBackground(blanco);
cuadro.setVisible(true);
this.add(cuadro);
```

Esta creación nos sirve para crear un panel dentro de nuestra ventana principal para poder colocar la gráfica que veremos más adelante.

CREACIÓN DE VENTANA PRINCIPAL

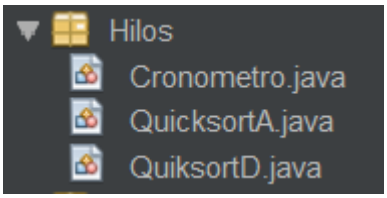
```
this.setTitle("GRAFICADORA USAC 2022");
this.setBounds(20, 20, 1340, 730);
this.getContentPane().setBackground(moradito);
this.setLayout(null);
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
this.setResizable(false);
this.setVisible(true);
this.add(cuadro);
```

Está será una de varias ventanas que podemos tener y su creación es la misma para todas.



HILOS.

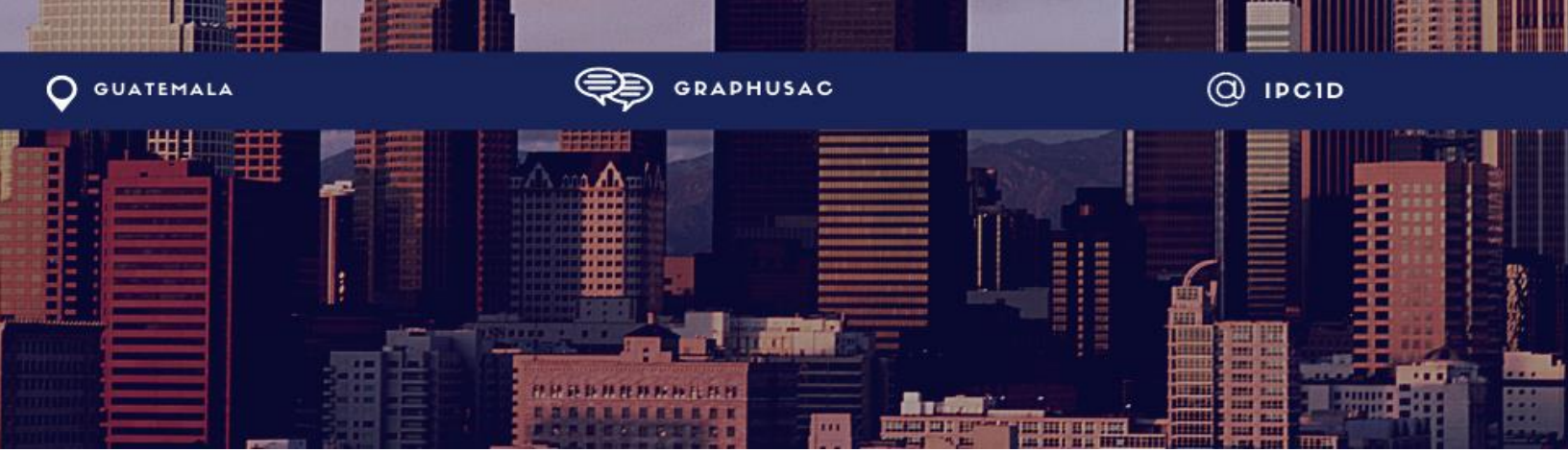
Un hilo es simplemente una tarea que puede ser ejecutada al mismo tiempo que otra tarea, los hilos de ejecución que comparten los mismos recursos, sumados a estos recursos son en conjunto conocidos como un proceso, en esta práctica utilizaremos únicamente 3 hilos que son los siguientes:



Cronometro.

```java
public class Cronometro extends Thread {
    //*************************************************************************
    //ATRIBUTOS DEL CRONOMETRO
    JLabel tiempo;
    int minutos;
    int segundos;

    //*************************************************************************
    //CONTRUCTOR
    public Cronometro(JLabel tiempo) {
        this.tiempo = tiempo;
        this.minutos = 0;
        this.segundos = 0;

    }
    //*************************************************************************
    //METODO RUN PARA QUE CORRA EL CRONOMETRO
    @Override
    public void run() {
        while (this.minutos != 60) {
            this.segundos = 0;
            while (this.segundos < 60) {
                System.out.println(this.minutos + ":" + this.segundos);
                this.tiempo.setText(this.minutos + ":" + this.segundos);
                try {
                    Thread.sleep(500);
                } catch (InterruptedException ex) {
                    Logger.getLogger(Cronometro.class.getName()).log(Level.SEVERE, null, ex);
                }
                this.segundos++;
            }
            this.minutos++;
        }
    }

}
```

Los hilos para identificarlos se hacen desde la clase extends, en este hilo se crea un constructor y un método run que hace que funcione el método.



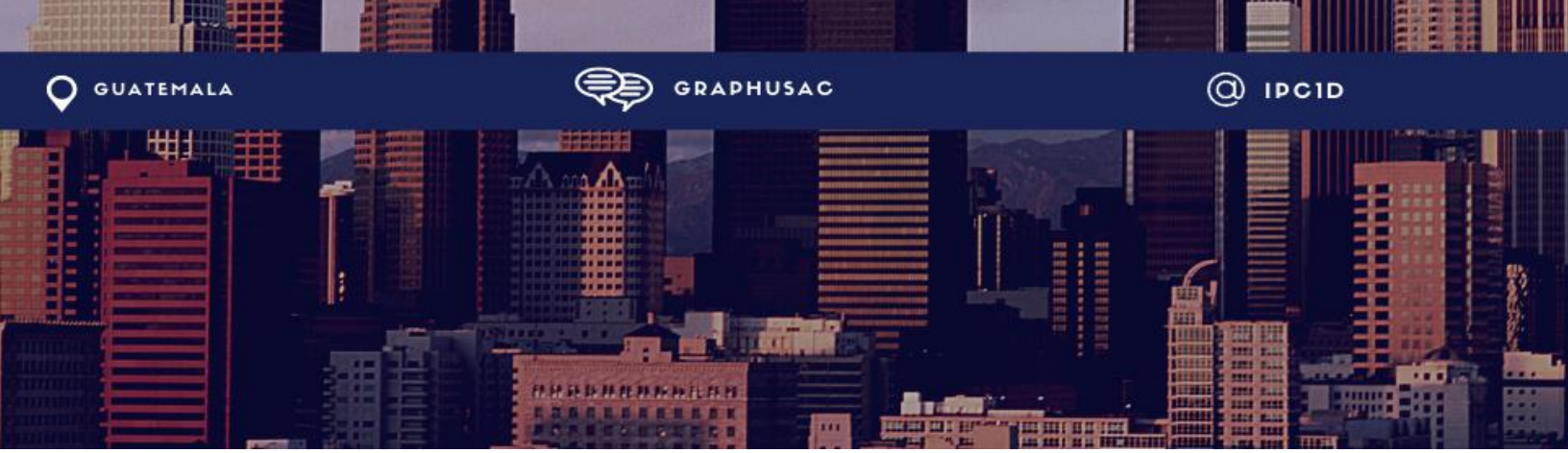
Ordenamiento QuickSort.

```
public class QuicksortA extends Thread {
    //*************************************************************************
    //ATRIBUTOS PARA EL ORDENAMIENTO
    Cronometro c = new Cronometro(tiempo1);
    private VentanaP clase;
    private int[] datos;
    private int contador;
    private int pasos = 0;

    //*************************************************************************
    //CONSTRUCTOR PARA ORDENAMIENTO
    public QuicksortA(VentanaP clase, int[] datos, int contador) {
        this.clase = clase;
        this.datos = datos;
        this.contador = contador;
    }
```

Creamos la clase hilos, con sus atributos y constructor

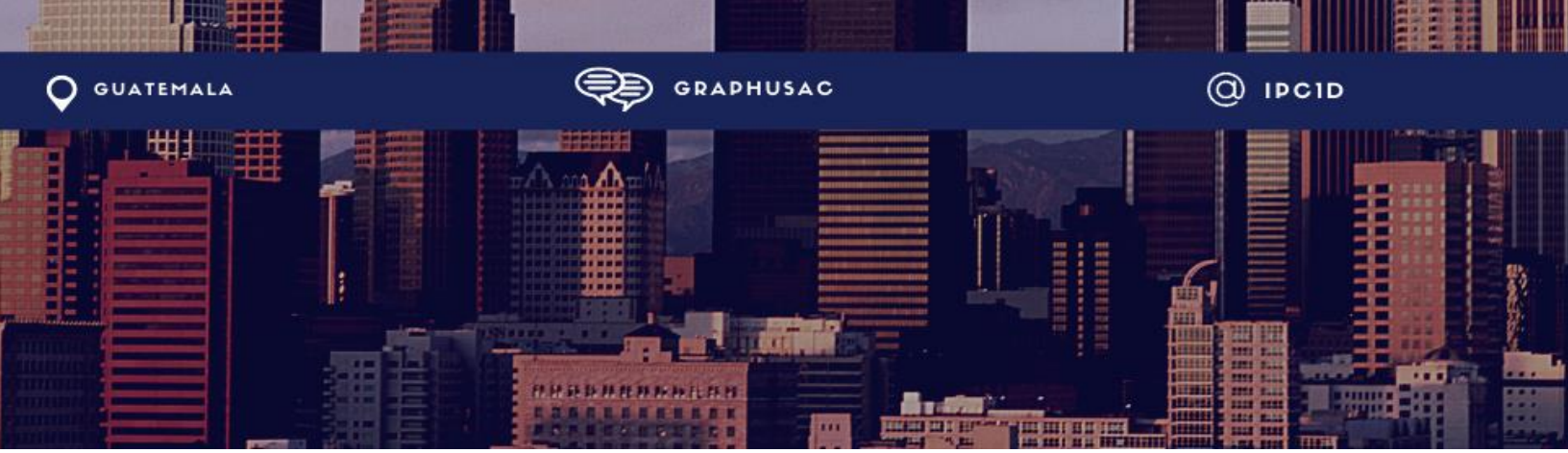
```
@Override
public void run() {
    c.start();
    quicksort(datos, 0, datos.length - 1);
    iniciar = false;
    clase.cuadro.removeAll();
    clase.cuadro.repaint();
    clase.grafica();
    c.stop();
    crearImagen(VentanaP.barChart);
    reportehtml();
}
```

El método run nos sirve para poder obtener el método de ordenamiento y obtener también todas las variables funcionales para poder que funcione nuestro ordenamiento.



```
public void quicksort(int[] datos1, int primero, int ultimo) { //algoritmo de ordenamiento quickSort como funcion recursiva
    int i, j;
    double pivote;
    int aux;
    i = primero;
    j = ultimo;
    pivote = datos1[(primero + ultimo) / 2]; //obtengo mi pivote

    do {
        while (datos1[i] < pivote) { //este ciclo me sirve para incrementar mi contador
            i++;
        }
        while (datos1[j] > pivote) { //este ciclo me sirve para decrementar mi contador
            j--;
        }
        if (i <= j) { //aca es donde hago el intercambio de valores que se ajustan a mi condicion y ajusto mis contadores
            aux = datos1[i];
            datos1[i] = datos1[j];
            datos1[j] = aux;
            try {
                Thread.sleep(500);
            } catch (InterruptedException ex) {
            }
            clase.cuadro.removeAll();
            clase.cuadro.repaint();
            clase.grafica();
            pasos++;
            clase.pasitos.setText(String.valueOf(pasos));

            i++;
            j--;
        }
    } while (i <= j && iniciar1);
    //como mi funcion es recursiva tengo que estar mandando los valores obtenidos segun mi condicion y asi encontrar los pivotes
    if (iniciar) {
        if (primero < j) {
            quicksort(datos1, primero, j);
        }
        if (i < ultimo) {
            quicksort(datos1, i, ultimo);
        }
    }

}
```

Este es el método de ordenamiento, este ordenamiento nos hace que los datos ser ordenen adecuadamente de menor a mayor, también tenemos un método de ordenamiento descendente, pero en este cambia solamente dentro de While el < por el > y el > por el <.



GRÁFICA.

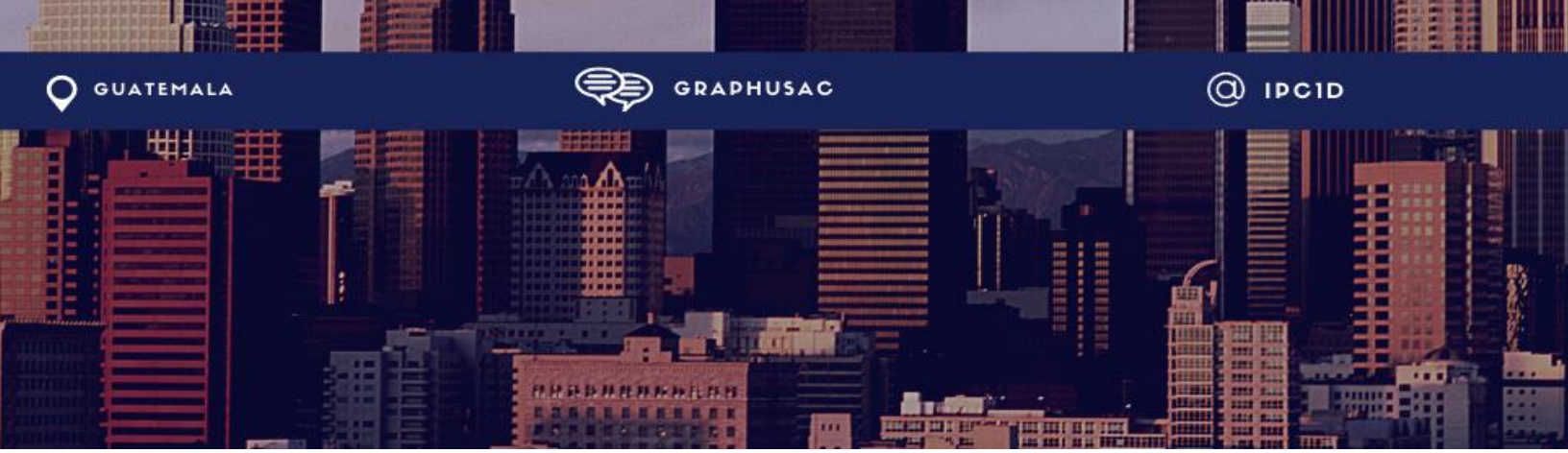
Creamos una gráfica para poder realizar el ordenamiento visto anteriormente por lo cual utilizamos la librería externa JFreeChart la cual nos ayuda a obtener una gráfica y esta misma la llamamos al final del método run de nuestro ordenamiento para que pueda utilizarse correctamente, en la siguiente imagen se muestra el método de gráfica:

```
public static void grafica() {
    String color = "";

    DefaultCategoryDataset dataset = new DefaultCategoryDataset();
    for (int i = 0; i < datos.length; i++) {
        dataset.addValue(datos[i], String.valueOf(i), color);
    }

    barChart = ChartFactory.createBarChart3D("", "", "", dataset, PlotOrientation.VERTICAL, true, true, false);
    ChartPanel panel = new ChartPanel(barChart);
    panel.setBounds(10, 50, 600, 400);
    cuadro.add(panel);

}
```

También para nuestro reporte tenemos que crear una imagen de nuestra gráfica ordenada la cual la obtenemos con el siguiente método.

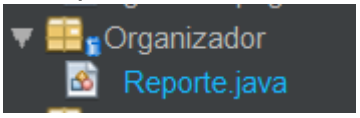
```
public static void crearImagen(JFreeChart a) {
    ChartRenderingInfo info = new ChartRenderingInfo(new StandardEntityCollection());
    File archivo = new File("C:\\Users\\Victor Rodriguez\\OneDrive\\Documentos\\IPC1_Practica2_201900018\\Practica2\\imagen.jpeg");
    try {
        ChartUtilities.saveChartAsJPEG( archivo , barChart , 760,400 );
    } catch (IOException ex) {
    }

}
```

Esto hace que al momento de generar la gráfica cree una imagen de la misma ordenada ahora pasaremos a verificar los reportes.



REPORTES.

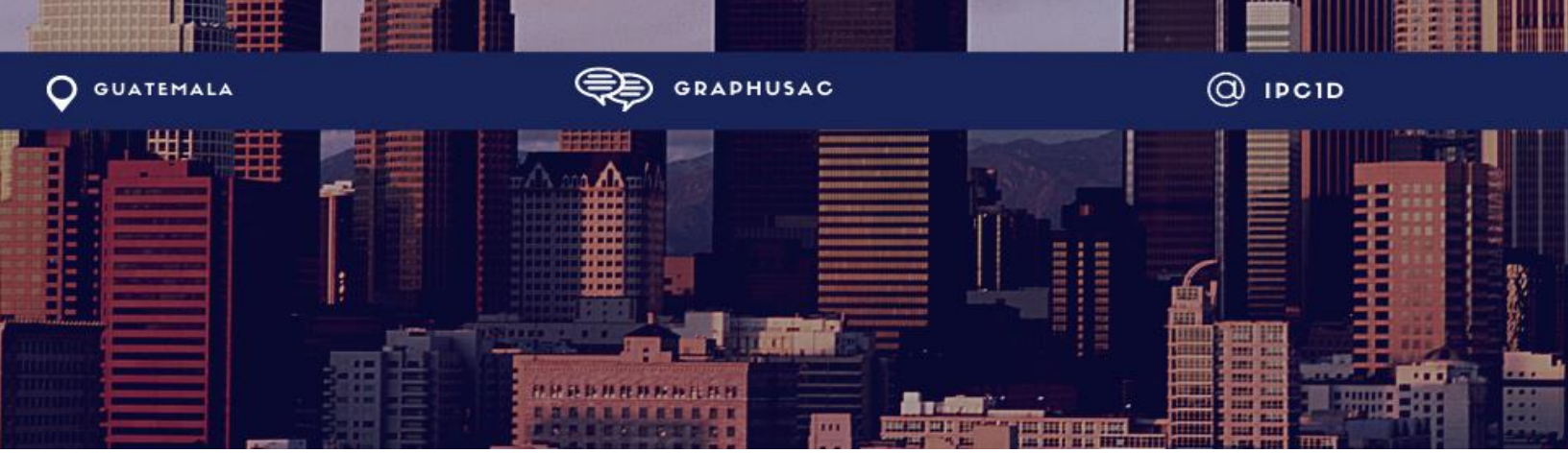
Para esta graficadora es necesario crear un Reporte HTML y así mismo un Reporte PDF para esto creamos un paquete y creamos una clase llamada Reporte.



Veremos como crear un Reporte HTML y para esto podemos obtener un método para crear el mismo el cual sera dividido en 2 imágenes.

```
public static void reportehtml() {
    String nombreReporte;
    File reporte;
    FileWriter fw;
    BufferedWriter br;
    String cadenaHTML;

    try {
        DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy_MM_dd_HH_mm_ss");
        String nombrereporte = (dtf.format(LocalDateTime.now()));
        reporte = new File(nombrereporte + ".HTML");
        fw = new FileWriter(reporte);
        br = new BufferedWriter(fw);

        cadenaHTML = "<html>"
                + "    <head>"
                + "<h1>Victor Eduardo José Rodriguez Alonzo</h1><h2>201900018</h2></div>"
                + "<h1> Ordenamiento Quicksort</h1></div>"
                + "<p><b>Tiempo:</b>" + tiempo1.getText() + "</p>"
                + "<p><b>Pasos:</b> " + pasitos.getText() + "</p>"
                + "<p><b>Datos desordenados:</b> </p>"
                + "    <body>";
        for (int i = 0; i < desordenados().length; i++) {
            cadenaHTML += "            <b>"
                    + "  <b>" + String.valueOf(desordenados()[i]) + "</b>" //llamamos lo que contiene la tabla
                    + "            </b>";
        }
        cadenaHTML += " <p><b>Datos Ordenados:</b> </p>";
        for (int i = 0; i < ordenados().length; i++) {
            cadenaHTML += "            <b>"
                    + "  <b>" + String.valueOf(ordenados()[i]) + "</b>" //llamamos lo que contiene la tabla
                    + "            </b>";
        }
```

```
        }
        cadenaHTML += " <p><b>Grafica Generada:</b> </p>"
                +"<img src=\"C:\\Users\\Victor Rodriguez\\OneDrive\\Documentos\\IPC1_Practica2_201900018\\Practica2\\imagen.jpeg\"/>";

        cadenaHTML += "</head"
                + "    </body>"
                + "</html>";

        br.write(cadenaHTML);

        br.close();
        fw.close();

        pdfgraf(cadenaHTML);

    } catch (IOException ex) {
        System.out.println("error escribiendo el reporte. Detalles " + ex.getMessage());
    }
}
```



Es un método muy fácil de utilizar el cual fue llamado como lo pudimos ver anteriormente en el método de QuickSort, ahora veremos el método para crear un PDF para esto utilizamos una librería llamada IText que es externa.

```
public static void pdfgraf(String html){
    try{ //E métodos para generar 3 reportes de arriba y abajo
       DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy_MM_dd_HH_mm_ss");
        String nombre = (dtf.format(LocalDateTime.now()));

            Document document = new Document(PageSize.LETTER);
            PdfWriter.getInstance(document, new FileOutputStream(nombre+".pdf"));

            document.open();
            document.addAuthor("Victor Rodriguez");
            document.addCreator("Victor Rodriguez");
            document.addSubject("Victor Rodriguez");
            document.addCreationDate();
            document.addTitle("Victor Rodriguez");

            HTMLWorker htmlWorker = new HTMLWorker(document);
            htmlWorker.parse(new StringReader(html));
            Image imagen = Image.getInstance("C:\\Users\\Victor Rodriguez\\OneDrive\\Documentos\\IPC1_Practica2_201900018\\Practica2\\imagen.jpeg");
            imagen.scaleToFit(500, 300);
            document.add(imagen);

            document.close();

    }catch(Exception e){
    }
}
```

Este método es llamado en el anterior para poder crearlo al finalizar el ordenamiento.