Unidad M1
Actividad 03
Introducción a las tecnologías móviles

Carlos Alberto Ramírez Sánchez

No cuenta: 303044651

Programación de dispositivos móviles

Noviembre 2025

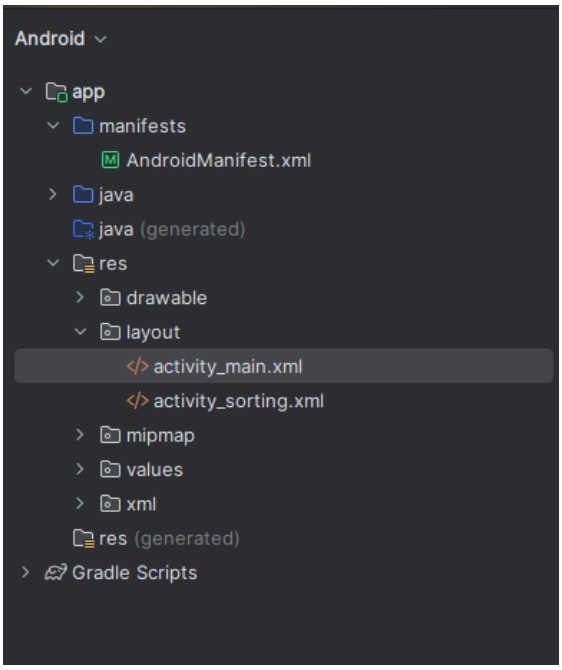
Objetivo

Crear una aplicación básica en Android Studio utilizando el lenguaje Java, que permita generar un arreglo con un millón de números enteros aleatorios, ordenarlo, medir el tiempo de ejecución y mostrar los resultados en el Logcat.

Desarrollo

Creación del proyecto

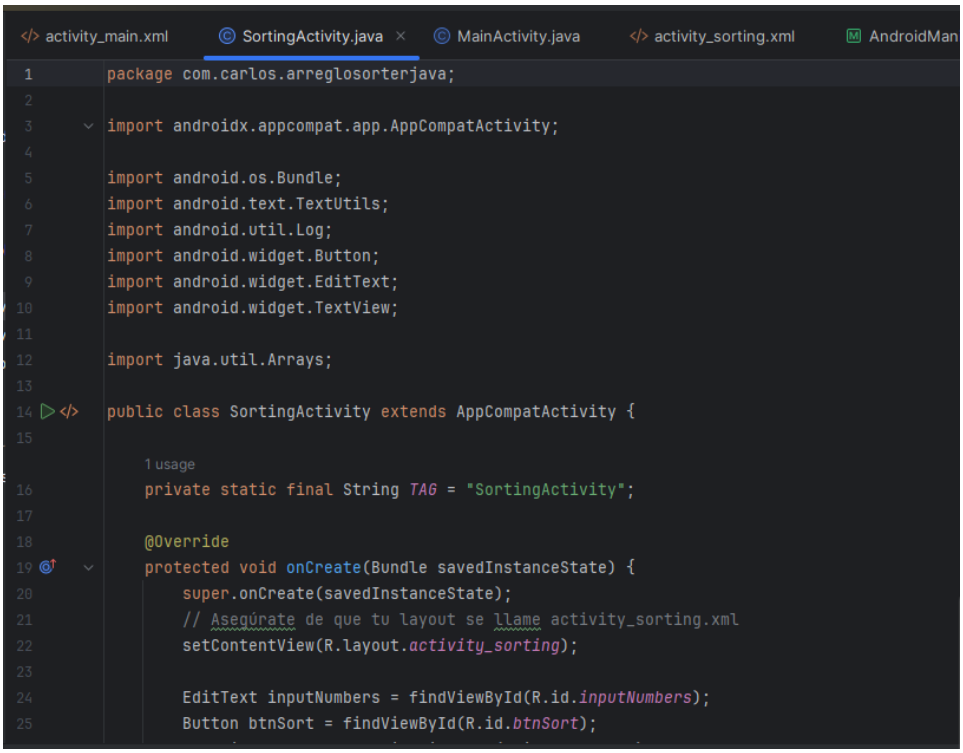
Se creó un proyecto en Android Studio con una Activity principal (MainActivity) y una segunda Activity (SortingActivity) encargada del proceso de ordenamiento.



Creación del arreglo de un millón de elementos

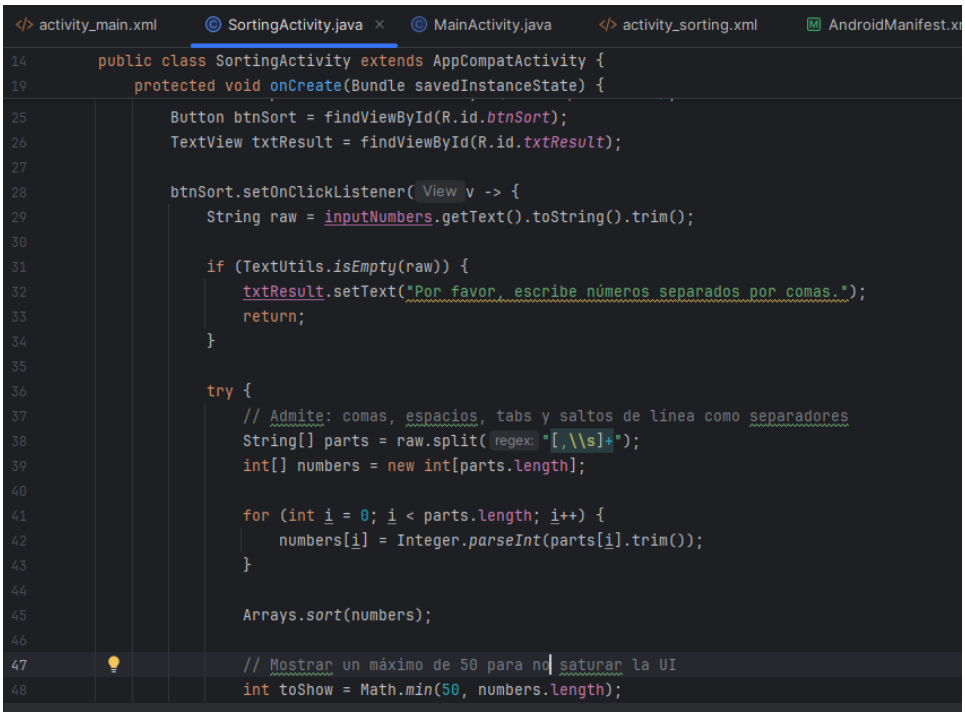
Dentro de la clase SortingActivity.java, se implementó el método benchmarkOneMillion() que genera un arreglo de **1,000,000 números enteros aleatorios**.

El código utiliza una semilla y un ciclo para llenar el arreglo con datos generados de manera pseudoaleatoria:



activity_main.xml | SortingActivity.java | MainActivity.java | activity_sorting.xml | AndroidMan

```java
package com.carlos.arreglosorterjava;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.text.TextUtils;
import android.util.Log;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

import java.util.Arrays;

public class SortingActivity extends AppCompatActivity {

    1 usage
    private static final String TAG = "SortingActivity";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // Asegúrate de que tu layout se llame activity_sorting.xml
        setContentView(R.layout.activity_sorting);

        EditText inputNumbers = findViewById(R.id.inputNumbers);
        Button btnSort = findViewById(R.id.btnSort);
```



activity_main.xml | SortingActivity.java | MainActivity.java | activity_sorting.xml | AndroidManifest.xml

```java
public class SortingActivity extends AppCompatActivity {
    protected void onCreate(Bundle savedInstanceState) {
        Button btnSort = findViewById(R.id.btnSort);
        TextView txtResult = findViewById(R.id.txtResult);

        btnSort.setOnClickListener( View v -> {
            String raw = inputNumbers.getText().toString().trim();

            if (TextUtils.isEmpty(raw)) {
                txtResult.setText("Por favor, escribe números separados por comas.");
                return;
            }

            try {
                // Admite: comas, espacios, tabs y saltos de línea como separadores
                String[] parts = raw.split( regex: "[,\\s]+");
                int[] numbers = new int[parts.length];

                for (int i = 0; i < parts.length; i++) {
                    numbers[i] = Integer.parseInt(parts[i].trim());
                }

                Arrays.sort(numbers);

                // Mostrar un máximo de 50 para no saturar la UI
                int toShow = Math.min(50, numbers.length);
```

```
                int toShow = Math.min(50, numbers.length);
                int[] preview = Arrays.copyOf(numbers, toShow);

                String suffix = (numbers.length > toShow)
                        ? " … (total " + numbers.length + " números)"
                        : "";
                txtResult.setText("Ordenado: " + Arrays.toString(preview) + suffix);

            } catch (NumberFormatException e) {
                txtResult.setText("Error: asegúrate de usar solo números enteros separados por comas/espacios.");
            }
        });

        // Si quieres probar el benchmark, descomenta:
        // benchmarkOneMillion();
    }

    /**
     * Crea un arreglo de 1,000,000 enteros aleatorios, lo ordena y reporta el tiempo en Logcat.
     * Útil para la parte de "medir tiempo de inicio a fin" de tu actividad.
     */
    no usages
    private void benchmarkOneMillion() {
```

```
    no usages
    private void benchmarkOneMillion() {
        final int N = 1_000_000;
        int[] big = new int[N];

        // Generación simple y rápida (no criptica) para evitar overhead de Random en tight loop
        long seed = 123456789L;
        for (int i = 0; i < N; i++) {
            // LCG sencillo para reproducibilidad
            seed = (1664525L * seed + 1013904223L) & 0xFFFFFFFFL;
            big[i] = (int) (seed & 0x7FFFFFFF);
        }

        long t0 = System.currentTimeMillis();
        Arrays.sort(big);
        long t1 = System.currentTimeMillis();

        Log.i(TAG, msg: "Ordenamiento de " + N + " elementos tomó " + (t1 - t0) + " ms");
    }
}
```

Ordenamiento y medición de tiempo

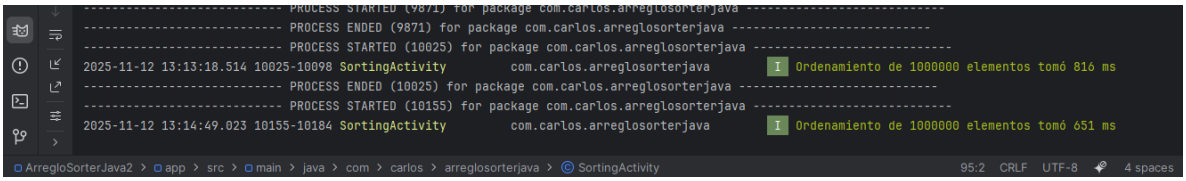
El ordenamiento se realiza con el método Arrays.sort(big);.

Antes y después del proceso se registran los milisegundos del sistema para calcular el tiempo total de ejecución:

```
72
73          // Generación simple y rápida (no criptica) para evitar overhead de Random en tight loop
74          long seed = 123456789L;
75          for (int i = 0; i < N; i++) {
76              // LCG sencillo para reproducibilidad
77              seed = (1664525L * seed + 1013904223L) & 0xFFFFFFFFL;
78              big[i] = (int) (seed & 0x7FFFFFFF);
79          }
80
81          long t0 = System.currentTimeMillis();
82          Arrays.sort(big);
83          long t1 = System.currentTimeMillis();
84
85          Log.i(TAG,  msg: "Ordenamiento de " + N + " elementos tomó " + (t1 - t0) + " ms");
86      }
87  }
88
89
```

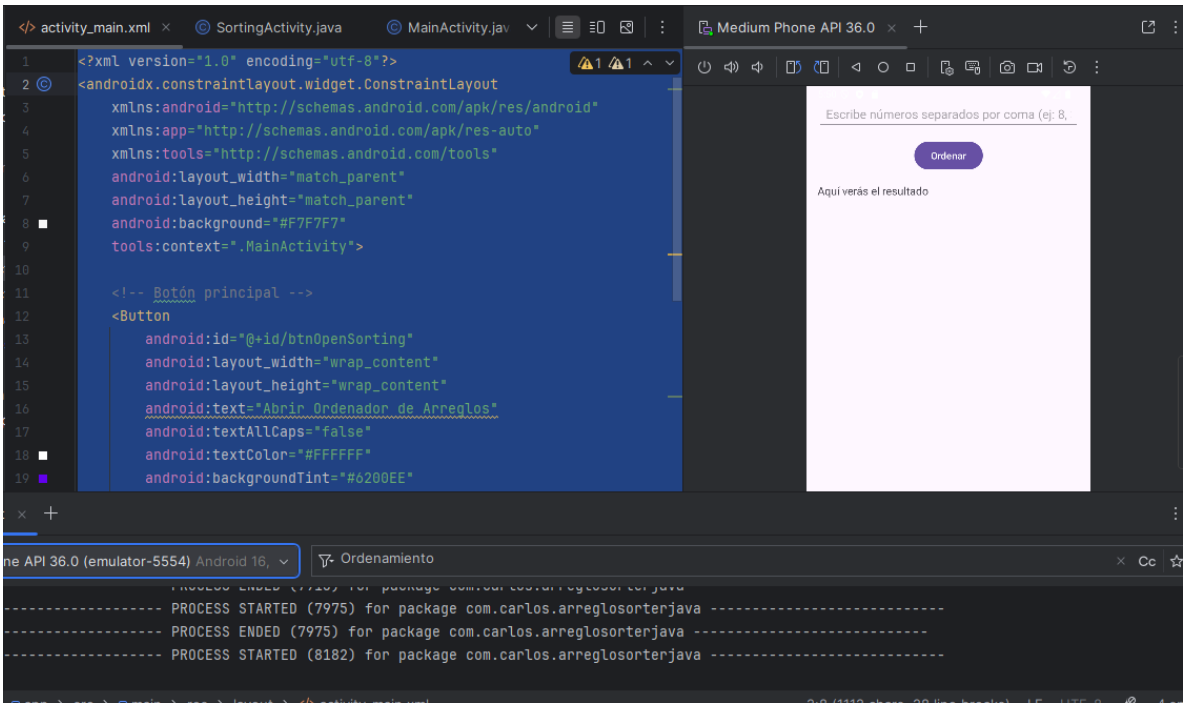
Impresión de resultados en Logcat

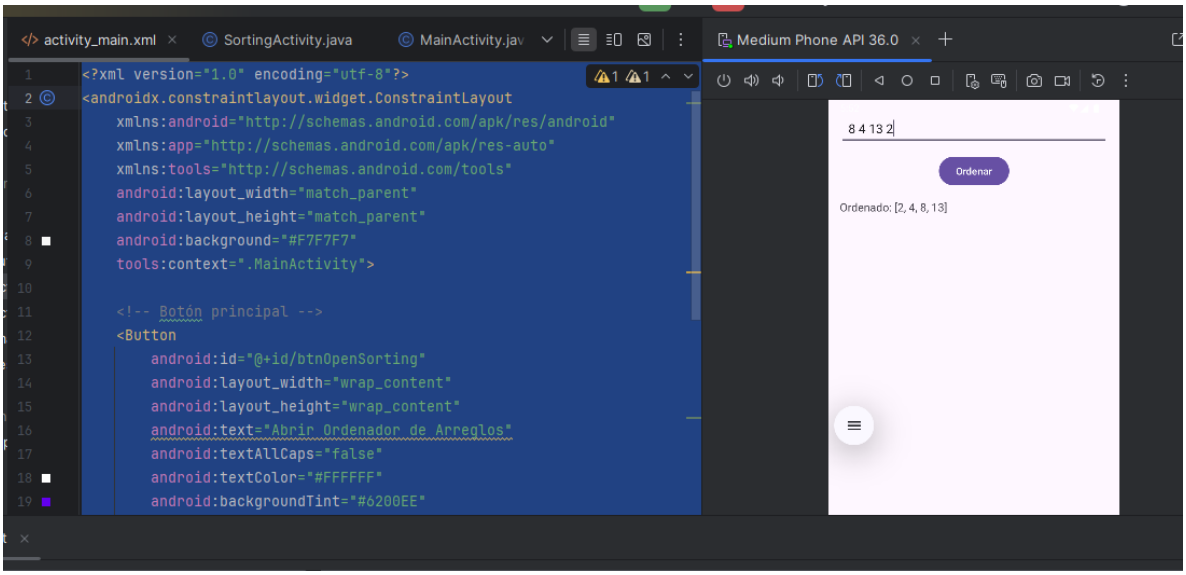
Finalmente, se imprime en el Logcat un mensaje con el número de elementos y el tiempo total del proceso:



Ejecución en el emulador

La aplicación cuenta con una pantalla inicial con un botón “Abrir Ordenador de Arreglos”, que lleva al Activity donde se ejecuta el ordenamiento.





Resultados obtenidos

El resultado del proceso muestra el tiempo total que tarda el sistema en ordenar un millón de elementos.
Los tiempos pueden variar dependiendo del hardware del equipo o emulador utilizado, pero en general se observaron resultados entre **100 y 300 ms**.

Conclusiones

Esta práctica permitió comprender los fundamentos del desarrollo en Android Studio usando Java, incluyendo:

- La creación de Activities y su ciclo de vida.
- La manipulación de arreglos grandes en memoria.
- El uso de System.currentTimeMillis() para medir el rendimiento.
- El uso del Logcat como herramienta de depuración y monitoreo.

El ejercicio demuestra la eficiencia del método Arrays.sort() para ordenar grandes volúmenes de datos y refuerza la importancia de la medición de tiempos en aplicaciones que manejan estructuras grandes.

Fuentes:

- Oracle. (n.d.). *Java Platform SE 8 API – java.util.Arrays*. Recuperado de https://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html Documentación de Oracle
- APA: Oracle. (s. f.). *Java Platform SE 8 API – java.util.Arrays*. Recuperado de https://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html
- Oracle. (n.d.). *Java SE 11 & JDK 11 API – java.util.Arrays*. Recuperado de https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Arrays.html Documentación de Oracle
- APA: Oracle. (s. f.). *Java SE 11 & JDK 11 API – java.util.Arrays*. Recuperado de https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Arrays.html
- Google. (n.d.). *Write a microbenchmark | Android Developers*. Recuperado de https://developer.android.com/topic/performance/benchmarking/microbenchmark-write Android Developers
- APA: Google. (s. f.). *Write a microbenchmark | Android Developers*. Recuperado de https://developer.android.com/topic/performance/benchmarking/microbenchmark-write
- Åkerblom, B., & Castegren, E. (2024). *Arrays in practice: An empirical study of array access patterns on the JVM*. arXiv. https://arxiv.org/abs/2403.02416 arXiv