

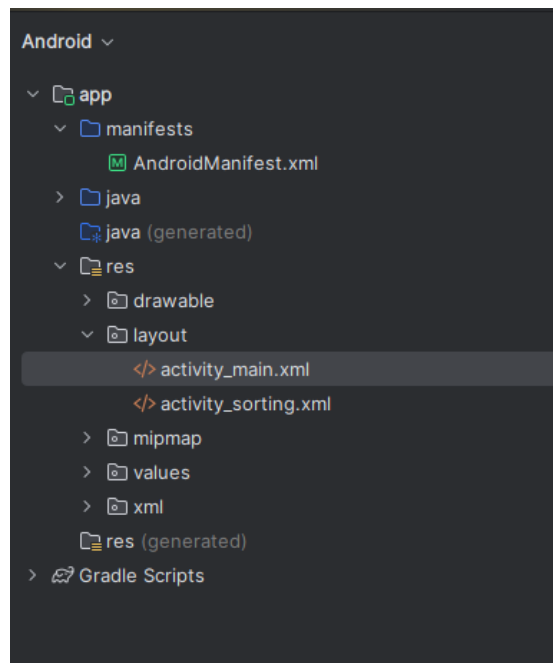
Objetivo

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  AndroidManifest.xml

1  package com.carlos.arreglosorterjava;
2
3  import androidx.appcompat.app.AppCompatActivity;
4
5  import android.os.Bundle;
6  import android.text.TextUtils;
7  import android.util.Log;
8  import android.widget.Button;
9  import android.widget.EditText;
10 import android.widget.TextView;
11
12 import java.util.Arrays;
13
14 public class SortingActivity extends AppCompatActivity {
15
16     //usage
17     private static final String TAG = "SortingActivity";
18
19     @Override
20     protected void onCreate(Bundle savedInstanceState) {
21         super.onCreate(savedInstanceState);
22         // Asegúrate de que tu layout se llame activity_sorting.xml
23         setContentView(R.layout.activity_sorting);
24
25         EditText inputNumbers = findViewById(R.id.inputNumbers);
26         Button btnSort = findViewById(R.id.btnSort);
```

```
</> activity_main.xml  © SortingActivity.java  ×  © MainActivity.java  </> activity_sorting.xml  AndroidManifest.xml

14 public class SortingActivity extends AppCompatActivity {
19     protected void onCreate(Bundle savedInstanceState) {
20
21         Button btnSort = findViewById(R.id.btnSort);
22         TextView txtResult = findViewById(R.id.txtResult);
23
24         btnSort.setOnClickListener( View v -> {
25             String raw = inputNumbers.getText().toString().trim();
26
27             if (TextUtils.isEmpty(raw)) {
28                 txtResult.setText("Por favor, escribe números separados por comas.");
29                 return;
30             }
31
32             try {
33                 // Admite: comas, espacios, tabs y saltos de línea como separadores
34                 String[] parts = raw.split(regex: "[,\\s]+");
35                 int[] numbers = new int[parts.length];
36
37                 for (int i = 0; i < parts.length; i++) {
38                     numbers[i] = Integer.parseInt(parts[i].trim());
39                 }
40
41                 Arrays.sort(numbers);
42
43                 // Mostrar un máximo de 50 para no saturar la UI
44                 int toShow = Math.min(50, numbers.length);
```

```

48         int toShow = Math.min(50, numbers.length);
49         int[] preview = Arrays.copyOf(numbers, toShow);
50
51         String suffix = (numbers.length > toShow)
52             ? "... (total " + numbers.length + " números)"
53             : "";
54         txtResult.setText("Ordenado: " + Arrays.toString(preview) + suffix);
55
56     } catch (NumberFormatException e) {
57         txtResult.setText("Error: asegúrate de usar solo números enteros separados por comas/espacios.");
58     }
59 }
60
61 // Si quieres probar el benchmark, descomenta:
62 // benchmarkOneMillion();
63
64
65 /**
66  * Crea un arreglo de 1,000,000 enteros aleatorios, lo ordena y reporta el tiempo en Logcat.
67  * Útil para la parte de "medir tiempo de inicio a fin" de tu actividad.
68  */
69 no usages
private void benchmarkOneMillion() {

```

```

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

```

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:

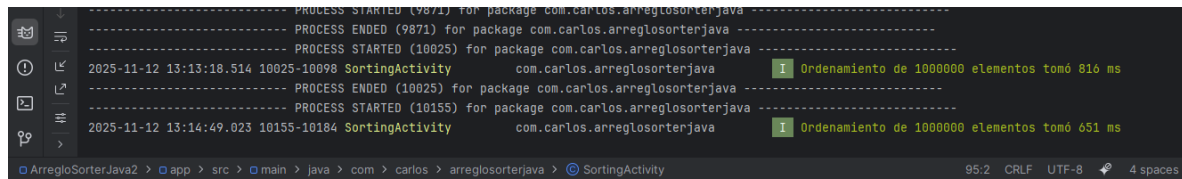
```

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:



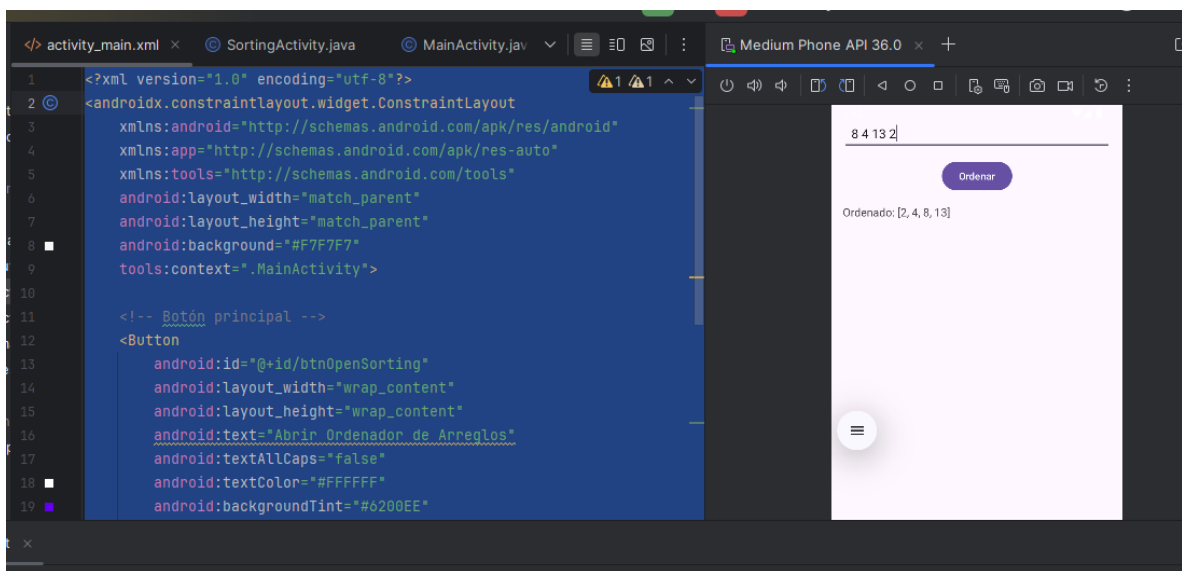
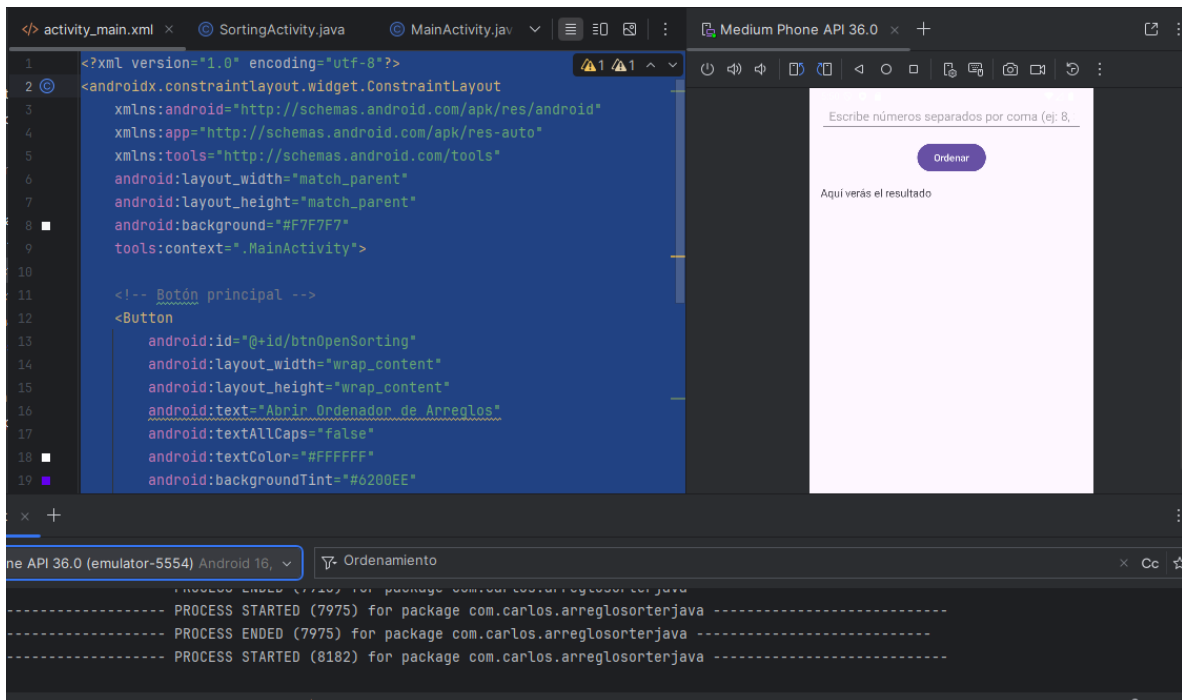
```

----- PROCESS STARTED (9871) for package com.carlos.arreglosorterjava -----
----- PROCESS ENDED (9871) for package com.carlos.arreglosorterjava -----
----- PROCESS STARTED (10025) for package com.carlos.arreglosorterjava -----
2025-11-12 13:13:18.514 10025-10098 SortingActivity com.carlos.arreglosorterjava I Ordenamiento de 1000000 elementos tomó 816 ms
----- PROCESS ENDED (10025) for package com.carlos.arreglosorterjava -----
----- PROCESS STARTED (10155) for package com.carlos.arreglosorterjava -----
2025-11-12 13:14:49.023 10155-10184 SortingActivity com.carlos.arreglosorterjava I Ordenamiento de 1000000 elementos tomó 651 ms

```

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