PROYECTO

DE ORDENAMIENTO Y

BIG DATA

ESCUELA POLITÉCNICA NACIONAL

Estructura de Datos y Algoritmos I

Ing. Boris Astudillo

Integrantes:

ABIGAIL BECERRA

DAVID JACOME

EDISON ORDOÑEZ

INTRODUCCIÓN

- El proyecto analiza un archivo con 1 millón de registros de películas.

- Se aplicarán métodos estadísticos y computacionales para analizar y ordenar los atributos del dataset.

OBJETIVO

- Desarrollar un experimento computacional sobre el dataset de 1M de registros.
- Implementar y medir el rendimiento de dos algoritmos de ordenamiento.
- Aplicar técnicas de multiprocesamiento en Python.
- Comparar la eficiencia del procesamiento en diferentes equipos (hardware).

MARCO TEÓRICO

BASE DE DATOS UTILIZADA

- **Dataset:** Información de cintas cinematográficas (+1 millón de datos).
- **Atributos clave:** ID, Título, Calificación, Fecha de Estreno, Presupuesto, Popularidad.
- **Algoritmos a comparar:** MergeSort y QuickSort.

FÓRMULA UTILIZADA

Propósito: Crear un "Valor Final" simulado para usarlo como criterio de ordenamiento.

Componentes de la simulación:

- Variación: Incremento aleatorio (5% a 15%) al presupuesto.
- Impuesto: Carga o descuento aleatorio (5%, 10% o 15%).

Fórmula:

Valor final = Budget x (1 + Variación) x (1 - Impuesto)

ALGORITMOS DE ORDENAMIENTO

Merge Sort:

- Divide recursivamente la lista hasta tener elementos únicos.
- Luego, combina (merge) las sublistas de forma ordenada.

Quick Sort:

- Reorganiza los elementos: menores a la izquierda del pivote y mayores a la derecha.
- Repite el proceso en las sublistas.

METODOLOGÍA

Se realizaron en 3 equipos computacionales distintos (Abigail, David, Edison).

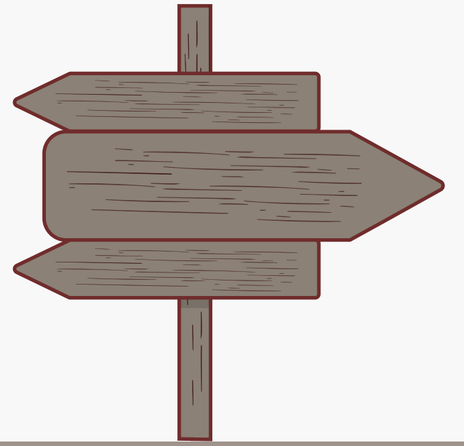
Especificaciones	Abigail	David	Edison
Procesador	Intel(R) Core i7-1265U (12ª generación)	Intel(R) Core i7-1355U (13ª generación)	Intel(R) Pentium(R) CPU
Frecuencia del Procesador	1.80 GHz	1.70GHz	2.16 GHz
Núcleos del Procesador	10	10	4
RAM	16	16	8

EJECUCIÓN EN PYTHON

Pasos principales:

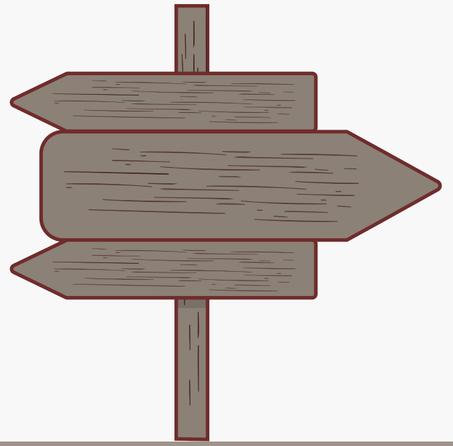
- Importar librerías: Pandas, Numpy, Multiprocessing.

```
import pandas as pd
import numpy as np
import time
import warnings
from multiprocessing import Pool, cpu_count

warnings.filterwarnings("ignore", category=FutureWarning)
```



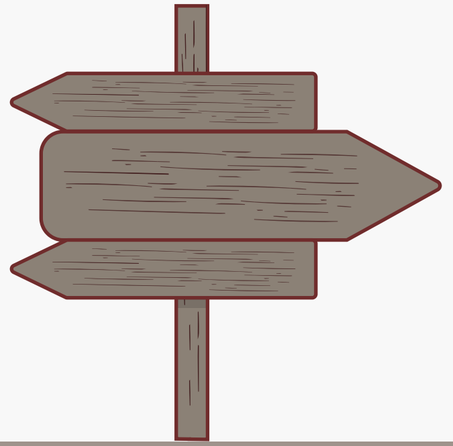
- Lectura del archivo .csv en el proyecto.

```
ruta = "BasedeDatosMoviesEDA.csv"

for enc in ["utf-8-sig", "cp1252", "latin-1", "ISO-8859-1"]:
    try:
        df = pd.read_csv(
            ruta,
            engine="python",
            sep=None,
            on_bad_lines="skip",
            encoding=enc
        )
        print("Cargó con encoding:", enc)
        break
    except UnicodeDecodeError as e:
        print("Falló con", enc, "->", e)

# Mostrar información del DataFrame
print(df.shape)
display(df.head())
print(df.info())
```



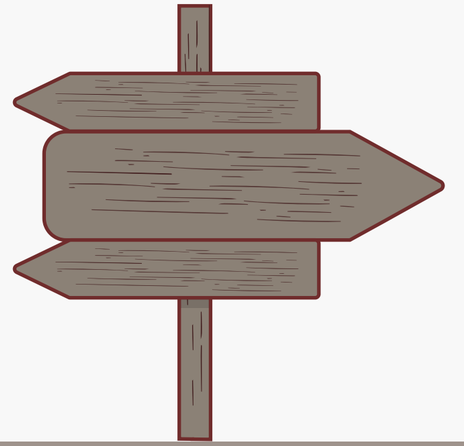
- Fórmulas que podrían usarse para la database.

```python
import numpy as np

# Se simula una variación del 5% a 15%
df["variacion"] = (df["budget"] * (1 + np.random.uniform(0.05, 0.15, len(df)))).round(2)

# Se simula un impuesto o descuento
df["impuesto"] = np.random.choice([0.05, 0.10, 0.15], size=len(df)).round(2)

# Se calcula un valor final
df["valor_final"] = (df["variacion"] * (1 - df["impuesto"])).round(2)

# Ordenar por valor_final de mayor a menor
df = df.sort_values(by="valor_final", ascending=False)

# Imprimir el DataFrame resultante
print(df[["title", "budget", "variacion", "impuesto", "valor_final"]].head(10))
```



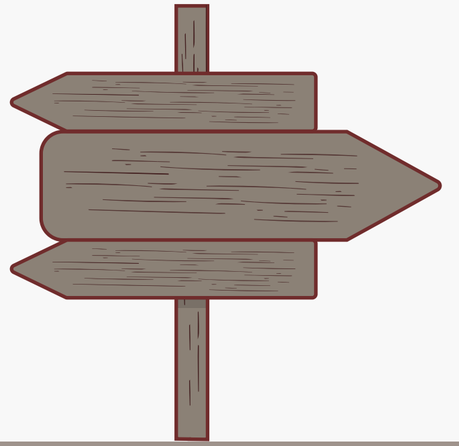
- Método del Algoritmo de Ordenamiento de MergeSort.

```python
def merge_sort(arr):
    if len(arr) <= 1:
        return arr
    mid = len(arr) // 2
    left = merge_sort(arr[:mid])
    right = merge_sort(arr[mid:])
    return merge(left, right)

def merge(left, right):
    result = []
    i = j = 0
    while i < len(left) and j < len(right):
        if left[i][1] <= right[j][1]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    result.extend(left[i:])
    result.extend(right[j:])
    return result
```



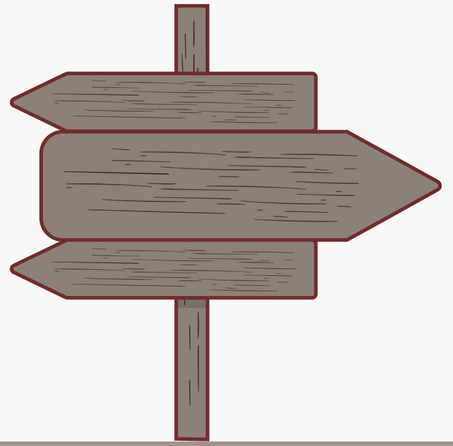
- Método del Algoritmo de Ordenamiento de QuickSort.

```
def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2][1]
    left = [x for x in arr if x[1] < pivot]
    middle = [x for x in arr if x[1] == pivot]
    right = [x for x in arr if x[1] > pivot]
    return quick_sort(left) + middle + quick_sort(right)
```



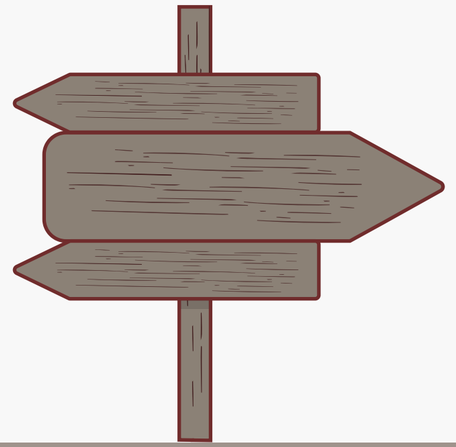
- Método para ejecutar los algoritmos.

```python
def ordenar_dataframe(df, algoritmo):
    """Ordena un DataFrame según 'valor_final' usando el algoritmo dado."""
    arr = list(zip(df.index, df["valor_final"].values))
    inicio = time.perf_counter()
    if algoritmo == "mergesort":
        ordenado = merge_sort(arr)
    elif algoritmo == "quicksort":
        ordenado = quick_sort(arr)
    else:
        raise ValueError("Algoritmo no reconocido. Usa 'mergesort' o 'quicksort'.")
    fin = time.perf_counter()

    indices_ordenados = [i for i, _ in ordenado]
    df_ordenado = df.loc[indices_ordenados].reset_index(drop=True)
    return df_ordenado, fin - inicio
```



- Métodos para iniciar el multiprocesamiento, nuevamente estarán los algoritmos, pero ya integrados al multiprocesamiento.

```python
from concurrent.futures import ThreadPoolExecutor, as_completed

def worker_mergesort(chunk):
    arr = list(zip(chunk.index, chunk["valor_final"].values))
    ordenado = merge_sort(arr)
    indices_ordenados = [i for i, _ in ordenado]
    return chunk.loc[indices_ordenados]

def worker_quicksort(chunk):
    arr = list(zip(chunk.index, chunk["valor_final"].values))
    ordenado = quick_sort(arr)
    indices_ordenados = [i for i, _ in ordenado]
    return chunk.loc[indices_ordenados]

def ejecutar_multiproceso(df, algoritmo):
    num_procesos = min(4, cpu_count())
    # repartir todas las filas, sin perder sobrantes
    chunks = [c for c in np.array_split(df, num_procesos) if len(c) > 0]

    print(f"\n Iniciando multiprocesamiento ({algoritmo}) con {len(chunks)} hilos...")
    t0 = time.perf_counter()

    if algoritmo == "mergesort":
        worker_func = worker_mergesort
    elif algoritmo == "quicksort":
        worker_func = worker_quicksort
    else:
        raise ValueError("Algoritmo inválido.")

    with ThreadPoolExecutor(max_workers=len(chunks)) as executor:
        futures = [executor.submit(worker_func, chunk) for chunk in chunks]
        partes = [future.result() for future in as_completed(futures)]

    # concatenar todas las partes; reset_index si quieres índice secuencial
    df_final = pd.concat(partes).reset_index(drop=True)
    t1 = time.perf_counter()

    tiempo_total = t1 - t0
    print(f" {algoritmo.capitalize()} paralelo completado en {tiempo_total:.3f} segundos.")
    return df_final, tiempo_total
```

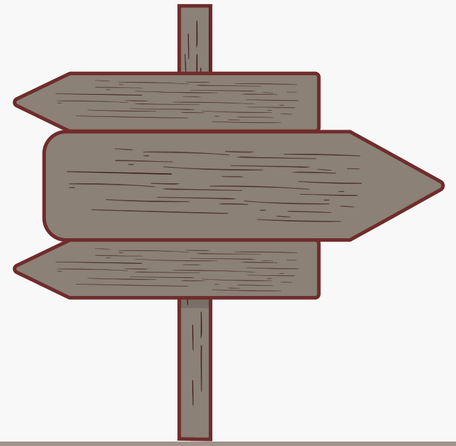


- Aplicación del multiprocesamiento y verificación de la duración de este en cada uno de los algoritmos.

```
print("\n--- ORDENAMIENTO CON MULTIPROCESAMIENTO ---")

df_merge_multi, tiempo_merge_multi = ejecutar_multiproceso(df, "mergesort")
print(f" MergeSort paralelo completado en {tiempo_merge_multi:.3f} segundos")

df_quick_multi, tiempo_quick_multi = ejecutar_multiproceso(df, "quicksort")
print(f" QuickSort paralelo completado en {tiempo_quick_multi:.3f} segundos")
```

- Comparación de rendimientos entre procesamiento mononúcleo y multiprocesamiento.

```
print("-----------------------------------------------")
print("\n--- COMPARACIÓN DE RENDIMIENTO ---")
print(f"MergeSort (sin multiproceso): {tiempo_merge:.3f}s")
print(f"MergeSort (con multiproceso): {tiempo_merge_multi:.3f}s")
print(f"Mejora: {((tiempo_merge - tiempo_merge_multi) / tiempo_merge * 100):.2f}%")

print(f"\nQuickSort (sin multiproceso): {tiempo_quick:.3f}s")
print(f"QuickSort (con multiproceso): {tiempo_quick_multi:.3f}s")
print(f"Mejora: {((tiempo_quick - tiempo_quick_multi) / tiempo_quick * 100):.2f}%")
```

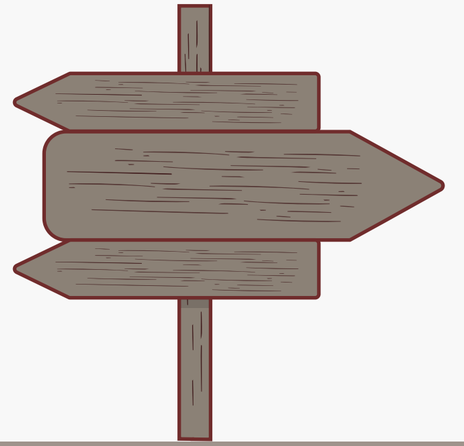


EJECUCIÓN EN JAVA

Pasos principales:

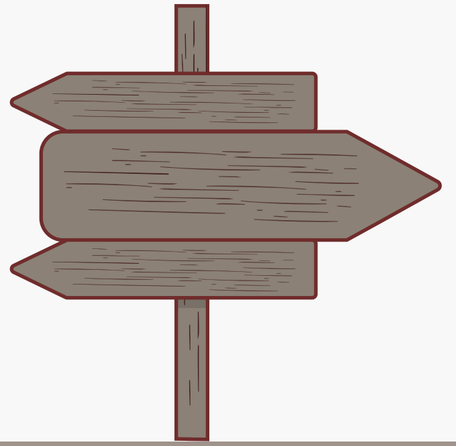
- Importación de librerias necesarias del proyecto.

```
import java.io.*;    ProyectoBimestral.java is a non-pr
import java.util.*;
import java.nio.file.*;
import java.util.concurrent.ThreadLocalRandom;
```



- Lectura del archivo .csv en el proyecto.

```java
//lector de archivo
public static double[] readBudgets(String path, int minSize) throws Exception {
    BufferedReader br = Files.newBufferedReader(Paths.get(path), java.nio.charset.StandardCharsets.ISO_8859_1);
    String header = br.readLine();
    boolean hasHeader = header != null && header.toLowerCase().contains(s: "budget");
    int budgetCol = -1;
    List<Double> vals = new ArrayList<>();
    if (hasHeader) {
        String[] cols = header.split(regex: "[,;]");
        for (int i = 0; i < cols.length; i++) if (cols[i].trim().toLowerCase().equals(anObject: "budget")) budgetCol = i;
    } else {
        budgetCol = 4;
        br = Files.newBufferedReader(Paths.get(path));
    }
    String line;
    while ((line = br.readLine()) != null) {
        String[] parts = line.split(regex: "[,;]");
        if (budgetCol >= parts.length) continue;
        try {
            double b = Double.parseDouble(parts[budgetCol].replaceAll(regex: "[^0-9.\\-]", replacement: ""));
            vals.add(b);
        } catch (Exception e) { /* saltar */ }
        if (vals.size() >= minSize) break;
    }
    // Aplicar valores random
    while (vals.size() < minSize) vals.add( ThreadLocalRandom.current().nextDouble(origin: 0, bound: 1e7) );
    double[] arr = new double[vals.size()];
    for (int i = 0; i < vals.size(); i++) arr[i] = vals.get(i);
    return arr;
}
```

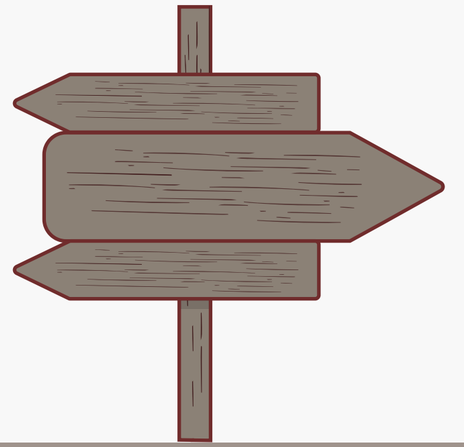


- Fórmulas que podrían usarse para la database.

```java
// Aplicar fórmula: variacion = budget * (1 + r) , impuesto in {0.05,0.10,0.15}, valor_final = variacion*(1-impuesto)
double[] valorFinal = new double[budgets.length];
Random rnd = new Random(seed: 12345);
double[] impuestos = {0.05,0.10,0.15};
for (int i=0;i<budgets.length;i++){
    double r = 0.05 + rnd.nextDouble()*(0.10); // [0.05,0.15)
    double variacion = budgets[i] * (1.0 + r);
    double imp = impuestos[rnd.nextInt(impuestos.length)];
    valorFinal[i] = variacion * (1.0 - imp);
}
```

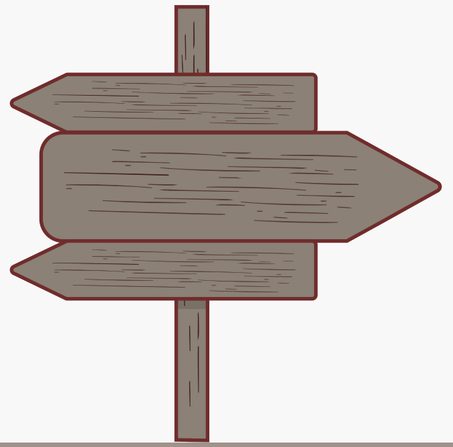
- Método del Algoritmo de Ordenamiento de MergeSort.

```java
// MergeSort
public static void mergeSort(double[] a, double[] aux, int lo, int hi) {
    if (hi - lo <= 0) return;
    int mid = (lo + hi) >>> 1;
    mergeSort(a, aux, lo, mid);
    mergeSort(a, aux, mid+1, hi);
    int i = lo, j = mid+1, k = lo;
    while (i <= mid || j <= hi) {
        if (j > hi || (i <= mid && a[i] <= a[j])) aux[k++] = a[i++];
        else aux[k++] = a[j++];
    }
    System.arraycopy(aux, lo, a, lo, hi - lo + 1);
}
```



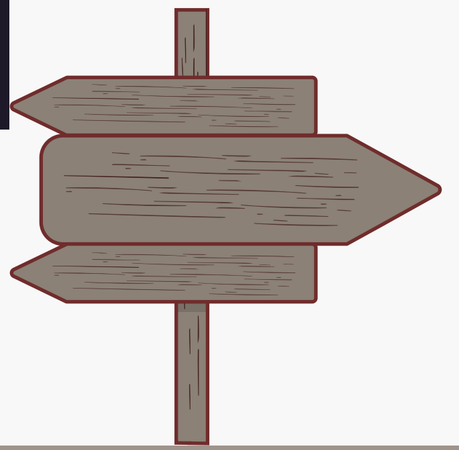
- Método del Algoritmo de Ordenamiento de QuickSort.

```java
// QuickSort
public static void quickSort(double[] a, int lo, int hi) {
    if (lo >= hi) return;
    int i = lo, j = hi;
    double pivot = a[lo + (hi - lo) / 2];
    while (i <= j) {
        while (a[i] < pivot) i++;
        while (a[j] > pivot) j--;
        if (i <= j) {
            double t = a[i]; a[i] = a[j]; a[j] = t;
            i++; j--;
        }
    }
    if (lo < j) quickSort(a, lo, j);
    if (i < hi) quickSort(a, i, hi);
}
```



- Aplicación de los métodos en la base de datos y tiempo que se tardaron en ejecutar cada algoritmo.

```java
// Tiempo Merge Sort
long t0 = System.nanoTime();
mergeSort(aForMerge, aux, lo: 0, aForMerge.length-1);
long t1 = System.nanoTime();
double mergeSec = (t1 - t0) / 1e9;
System.out.printf(format: "Mergesort: %.6f s%n", mergeSec);

// Tiempo Quick Sort
t0 = System.nanoTime();
quickSort(aForQuick, lo: 0, aForQuick.length-1);
t1 = System.nanoTime();
double quickSec = (t1 - t0) / 1e9;
System.out.printf(format: "Quicksort: %.6f s%n", quickSec);

System.out.printf(format: "Mejor ordenamiento: %s%n", (mergeSec < quickSec) ? "Mergesort" : "Quicksort");
```



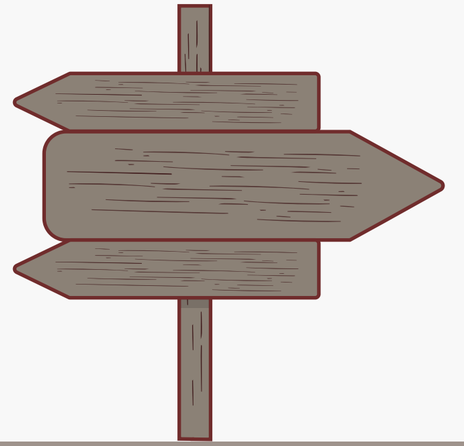
MULTIPROCESAMIENTO EN PYTHON

Librería: ThreadPoolExecutor.

Estrategia:

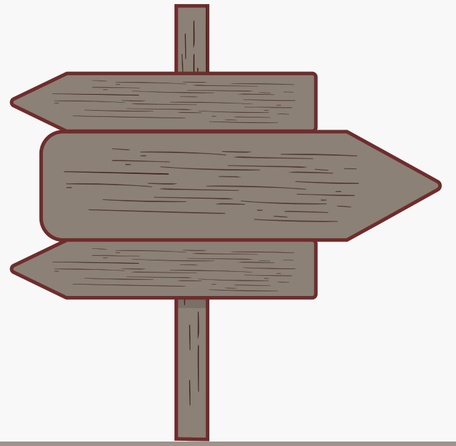
- Se divide (split) el DataFrame en "chunks" (trozos).
- Se asigna un "worker" (hilo) para ordenar cada chunk.
- Se usan min(4, cpu_count()) procesos.
- Se concatenan los resultados ordenados.

Objetivo: Comparar el tiempo mononúcleo vs. multiproceso.



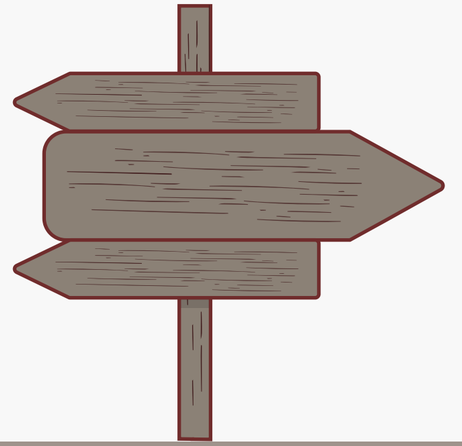
RESULTADOS SIN MULTIPROCESAMIENTO (PYTHON)

Tipo de Algoritmo	Abigail	David	Edison
MergeSort	1.336 Seg	2.872 Seg	9.993 Seg
QuickSort	0.191 Seg	0.377 Seg	1.094 Seg



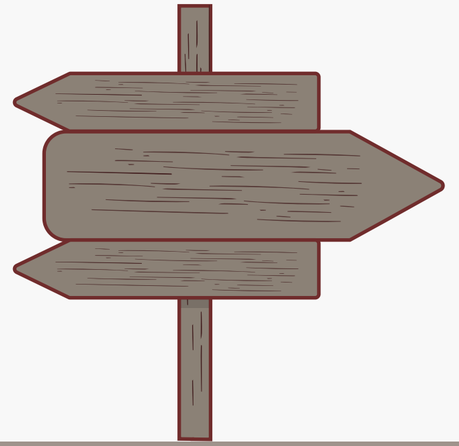
RESULTADOS CON EL MULTIPROCESAMIENTO (PYTHON)

Tipo de Algoritmo	Abigail	David	Edison
MergeSort	2.021 Seg	3.505 Seg	26.568 Seg
QuickSort	1.108 Seg	1.136 Seg	7.385 Seg



RESULTADOS CON EL MULTIPROCESAMIENTO (JAVA)

Tipo de Algoritmo	Abigail	David	Edison
MergeSort	0.118 Seg	0.0632 Seg	0.261 Seg
QuickSort	0.088 Seg	0.0605 Seg	0.166 Seg



ANÁLISIS

ANÁLISIS Y DISCUSIÓN

Observación 1 (Mononúcleo): QuickSort fue notablemente superior (1.09s) frente a MergeSort (9.99s).

Observación 2 (Multiproceso): Hubo un resultado inesperado.

- Ambos algoritmos fueron más lentos en paralelo.
- MergeSort pasó de 9.9s a 26.5s.
- QuickSort pasó de 1.0s a 7.3s.

Hipótesis: El paralelismo introdujo más overhead (costo de gestión, partición y copia) que beneficio para este problema.

CONCLUSIONES

- QuickSort ofrece un rendimiento superior en ejecución mononúcleo para este atributo.
- El multiprocesamiento no garantiza mejoras automáticas.
- En este caso, los costos de particionar y comunicar los datos superaron las ventajas de la concurrencia.
- Es crucial seleccionar el algoritmo y la estrategia de paralelización adecuados según el contexto.

MUCHAS GRACIAS