De la teoría a la práctica – Merge Sort, Benchmarks y Aplicaciones

Departamento de Física.

Corodinadora: C Loyola
Profesores C Femenías / F Bugini / D Basantes

Primer Semestre 2025

Universidad Andrés Bello
Departamento de Física y Astronomía

Universidad Andrés Bello®

python powered

Resumen – Semana 14, Sesión 2

Recap rápido

Recursividad & Merge Sort

Benchmark y visualización

Caso de aplicación con **pandas**

Trabajo en sala

Cierre

Recap rápido

Recap rápido $\in$ Lo aprendido en la Sesión 1

- **Bubble Sort**: claro pero ineficiente $\mathcal{O}(n^2)$.
- **Binary Search**: súper-rápida, pero sólo en listas ordenadas.
- `%%timeit`: herramienta para medir micro-rendimiento en Colab.

Pregunta relámpago: ¿cuál es la complejidad de buscar linealmente en una lista?

Recursividad & Merge Sort

- Una función que se llama a sí misma para resolver un problema más pequeño.
- Necesita: *caso base* + *llamada recursiva*.
- Ejemplo clásico: factorial, Fibonacci... ¡y algoritmos de ordenamiento!

Recursividad & Merge Sort $\in$ Merge Sort en Python ($\mathcal{O}(n \log n)$)

```python
def merge_sort(lst):
    if len(lst) <= 1:
        return lst                      # caso base
    mid = len(lst) // 2
    left  = merge_sort(lst[:mid])       # divide
    right = merge_sort(lst[mid:])
    return merge(left, right)           # conquista

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

- Dos fases: división recursiva y mezcla ordenada.
- Estable (preserva orden relativas de iguales) y predecible.

Benchmark y visualización

Benchmark y visualización ∈ Midiendo rendimientos

```python
import numpy as np, timeit, pandas as pd

# --- función de benchmark ----------------------------------------
def timing(alg, n, rep=3):
    base = np.random.randint(0, 10_000, n).tolist()
    return timeit.timeit(lambda: alg(base.copy()), number=rep) / rep

# --- medir -------------------------------------------------------
Ns   = [2**k for k in range(8, 15)]           # 256 … 16384
algs = {"bubble": bubble_sort,
        "merge":  merge_sort,
        "numpy":  lambda x: np.sort(x).tolist()}

df = pd.DataFrame({name: [timing(f, n) for n in Ns]
                   for name, f in algs.items()},
                  index=Ns)

print(df)
```

- Función `timing` promedia `rep` ejecuciones.
- Guardamos resultados en un `DataFrame` ⟶ ideal para graficar.

Benchmark y visualización ∈ Gráfica bubble vs merge vs np.sort

```python
import matplotlib.pyplot as plt

df.plot(marker="o")
plt.loglog()                          # ejes log-log
plt.xlabel("Tamaño de la lista (n)")
plt.ylabel("Tiempo [s]")
plt.title("Escalamiento temporal de algoritmos de
↪  ordenamiento")
plt.legend(title="Algoritmo")
plt.show()
```

- La pendiente 2 de bubble confirma su $\mathcal{O}(n^2)$.
- Merge y `np.sort` (Timsort) se alinean con $n \log n$.

Caso de aplicación con pandas

Caso de aplicación con pandas ∈ Mini-dataset astronómico

```python
import pandas as pd, numpy as np
np.random.seed(0)
df = pd.DataFrame({
    "nombre":  [f"Star-{i}" for i in range(5000)],
    "mag":     np.random.normal(8, 1.2, 5000).round(2),
    "dist_pc": np.random.exponential(20, 5000).round(1)
})
```

- Cada grupo puede descargar un catálogo real (`exoplanet.eu`) o usar este simulado.
- Tareas guiadas:
  1. Ordenar por magnitud aparente y hallar los 10 más brillantes.
  2. Usar `binary_search` para ubicar rápidamente una estrella con magnitud ≈ 9.3 en la lista ordenada.

Trabajo en sala

Trabajo en sala $\in$ Actividad práctica grupal

Objetivo

Comparar empíricamente $\mathcal{O}(n^2)$ y $\mathcal{O}(n \log n)$ y reflexionar sobre cuándo vale la pena optimizar.

Pasos sugeridos

1. Implementar `merge_sort` (o `quick_sort`) desde cero.
2. Medir tiempos con `%%timeit` para $n = 2^{10}, 2^{12}, 2^{14}$.
3. Graficar tiempo vs. n y anotar pendiente aproximada.
4. Responder en el notebook: ¿Para qué tamaños de n bubble sort sigue siendo aceptable?

Aprevechemos de ejercitar y aclarar dudas. Hay Tarea la próxima semana.

Cierre

Cierre $\in$ Conclusiones

- Recursividad introduce *divide & conquer*: potencia evidente en merge sort.
- Visualizar tiempos ayuda a internalizar las escalas de complejidad.
- La próxima semana veremos cómo *perfilar* código y optimizar "cuellos de botella" (**Semana 15**).

¡Nos vemos en el próximo laboratorio!