Universidad del Valle de Guatemala
Algoritmos y Estructuras de Datos
Daniel Hernández
Iván Roblero

Hoja de Trabajo 3 – Sistemas de Ordenamiento

Profiler Usado

Para este proyecto se utilizó System.nanoTime(), que es el método estándar de Java para medir tiempos de ejecución con precisión de nanosegundos.

Implementación:

```
long inicio = System.nanoTime();
algoritmo.sort(datos);
long fin = System.nanoTime();
long tiempoTranscurrido = fin - inicio;
```

El tiempo se convierte de nanosegundos a milisegundos dividiendo entre 1,000,000.

Metodología de Profiling

Tamaños de datos probados:

-10, 50, 100, 500, 1,000, 2,000, 3,000 elementos

Condiciones evaluadas:

- Datos desordenados (caso promedio)
- Datos ordenados (mejor caso)

Algoritmos evaluados:

- Merge Sort
- Quick Sort
- Radix Sort
- Heap Sort
- Gnome Sort

Universidad del Valle de Guatemala
Algoritmos y Estructuras de Datos
Daniel Hernández
Iván Roblero

Proceso:

Generación de datos: Números aleatorios entre 0 y 10,000

Guardado en archivos: Cada tamaño en un archivo separado

Medición:

- Primera ejecución con datos desordenados
- Segunda ejecución con datos ordenados (usando Arrays.sort())

Registro: Todos los tiempos guardados en CSV

Resultados Obtenidos

1.1 Gnome Sort

Tamaño	Desordenado (ms)	Ordenado (ms)
10	1.005	0.001
50	0.032	0.001
100	0.129	0.002
500	Desordenado (ms) 2.731	Ordenado (ms) 0.025
1000	1.290	0.003
2000	2.497	0.007
3000	5.922	0.005

Análisis:

Comportamiento $O(n^2)$ en

caso promedio

Excelente en datos ya ordenados: O(n)

Diferencia dramática entre ordenado y desordenado

No recomendable para grandes volúmenes

1.2 Merge Sort

Tamaño	Desordenado (ms)	Ordenado (ms)
10	1.228	0.005
50	0.028	0.023
100	0.060	0.058
500	0.125	0.080
1000	0.157	0.074
2000	0.431	0.307
3000	0.603	0.399

Análisis:

Comportamiento consistente O(n log n)

Muy predecible y confiable

Uno de los mejores para datasets grandes

1.3 Quick Sort

Universidad del Valle de Guatemala
Algoritmos y Estructuras de Datos
Daniel Hernández
Iván Roblero

Tamaño	Desordenado (ms)	Ordenado (ms)
10	1.732	0.005
50	0.017	0.047
100	0.063	0.177
500	0.085	1.241
1000	0.080	2.098
2000	0.136	20.480
3000	0.201	3.760

Análisis:

- Problema crítico: Peor rendimiento con datos ordenados
- Caso desordenado: Excelente O(nlogn)
- Caso ordenado: Degradación a $O(n^2)$
- Esto ocurre por la elección del pivote (último elemento)
- Solución: Usar pivote aleatorio o mediana de tres

1.4Radix Sort

Tamaño	Desordenado (ms)	Ordenado (ms)
10	1.541	0.006
50	0.017	0.023
100	0.033	0.030
500	0.124	0.136

Universidad del Valle de Guatemala
Algoritmos y Estructuras de Datos
Daniel Hernández
Iván Roblero

1000	0.239	0.260
2000	0.500	0.462
3000	0.473	0.345

Análisis:

Rendimiento muy consistente entre ordenado/desordenado

Complejidad O(d*n) donde d = dígitos

Excelente para números de rango conocido

No depende de comparaciones

Universidad del Valle de Guatemala
Algoritmos y Estructuras de Datos
Daniel Hernández
Iván Roblero

1.5 Heap Sort

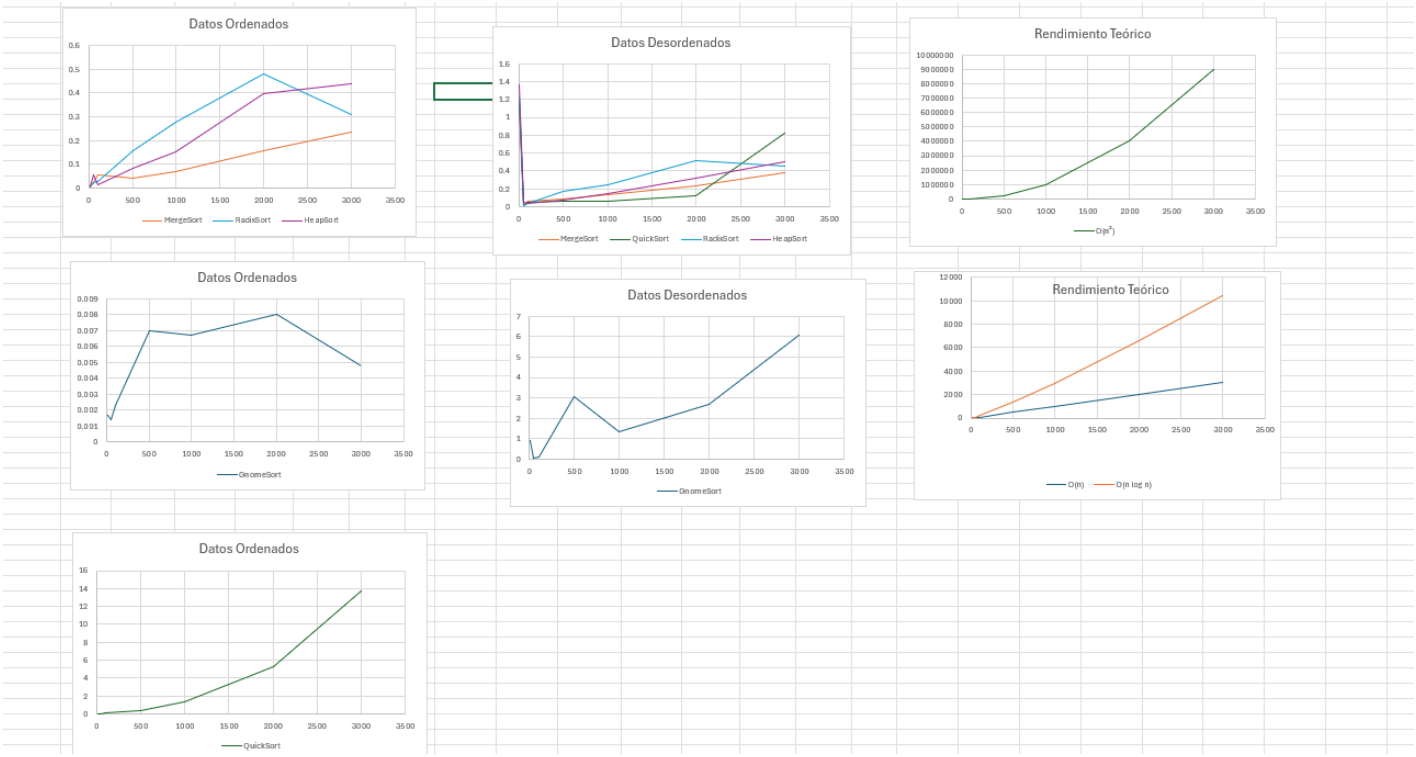
Tamaño	Desordenado (ms)	Ordenado (ms)
10	1.804	0.007
50	0.057	0.057
100	0.028	0.013
500	0.102	0.106
1000	0.150	0.193
2000	0.324	0.346
3000	0.563	0.570

Análisis:

- Comportamiento muy estable O(n log n)
- Mínima diferencia entre casos
- Bueno para memoria limitada (in-place)
- Rendimiento predecible

Universidad del Valle de Guatemala
Algoritmos y Estructuras de Datos
Daniel Hernández
Iván Roblero

Resultados en Excel:



Hicimos 3 graficas representando a los gráficos ordenados, ya que el quick sort y el gnome sort acá desbalanceaban la escala, en los gráficos de datos desordenados se logra ver que quick sort al tener una gran cantidad de datos se balancea con los demás. Pero gnome sort tarda más tiempo porque es O(n^2). A continuación se presenta la complejidad de cada uno de los algoritmos en Big O:

1. Gnome Sort: $O(n^2)$
2. Merge Sort: $O(n \log n)$
3. Quick Sort: $O(n^2)$
4. Radix Sort: $O(d \times n)$
5. Heap Sort: $O(n \log n)$