PROYECTO DE

"ANÁLISIS COMPARATIVO DE EFICIENCIA ENTRE LOS ALGORITMOS SHELLSORT Y COUNTING SORT"

INTEGRANTES:

Pedro Pablo Ninaja Medina

pninjam@unjbg.edu.pe

Universidad Nacional jorge Basadre Grohman

ORCID:https://orcid.org/0009-0009-3285-598X

Jairo Brayan Zegarra Gutierrez

jzegarrag@unjbg.edu.pe

Universidad Nacional jorge Basadre Grohman

ORCID:https://orcid.org/0009-0009-3285-598X

Arturo Willy Montalico Llica

Amontalicol@unjbg.edu.pe

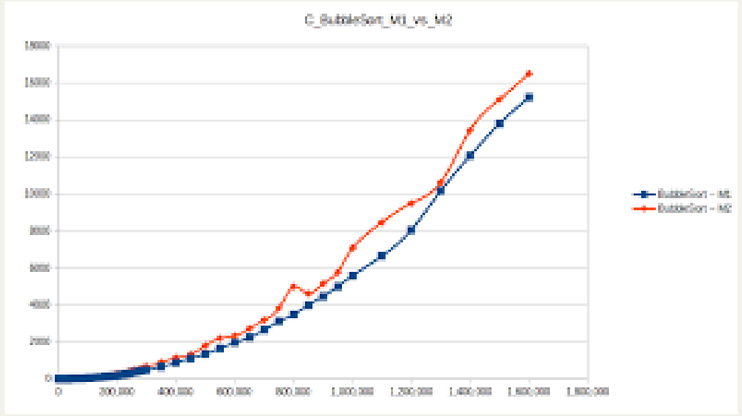
Universidad Nacional jorge Basadre Grohman

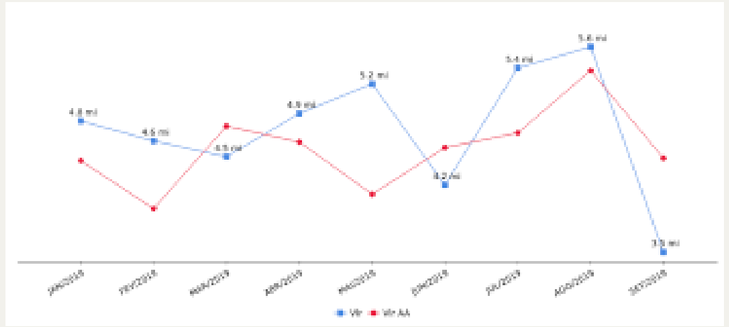
ORCID: https://orcid.org/0009-0006-9380-1799

INTRODUCCIÓN

El ordenamiento de datos es una operación esencial en computación, ya que mejora el acceso y procesamiento de la información. La eficiencia de cada algoritmo varía según el tipo y tamaño de los datos.

Este trabajo presenta un análisis comparativo entre Shellsort y Counting Sort, dos algoritmos con enfoques diferentes: uno basado en comparaciones y otro no comparativo. Se busca comprender mejor la notación Big O y se incluyen pruebas en C++ para observar su rendimiento en la práctica.





METODOS

Shellsort

Algoritmo comparativo derivado de Insertion Sort. Mejora el desempeño mediante comparaciones de elementos separados por un intervalo (gap) que se reduce progresivamente hasta 1. Se utilizó la secuencia de incrementos de Ciura, conocida por ofrecer un balance eficiente entre tiempo y número de comparaciones.

Counting Sort

Algoritmo no comparativo que organiza los datos contando las ocurrencias de cada valor. Se implementó la versión estable, que conserva el orden relativo de elementos iguales mediante un arreglo auxiliar acumulativo

COMPLEJIDAD TEÓRICA

La comparación se fundamentó en los modelos de complejidad temporal y espacial descritos por la notación asintótica Big O.

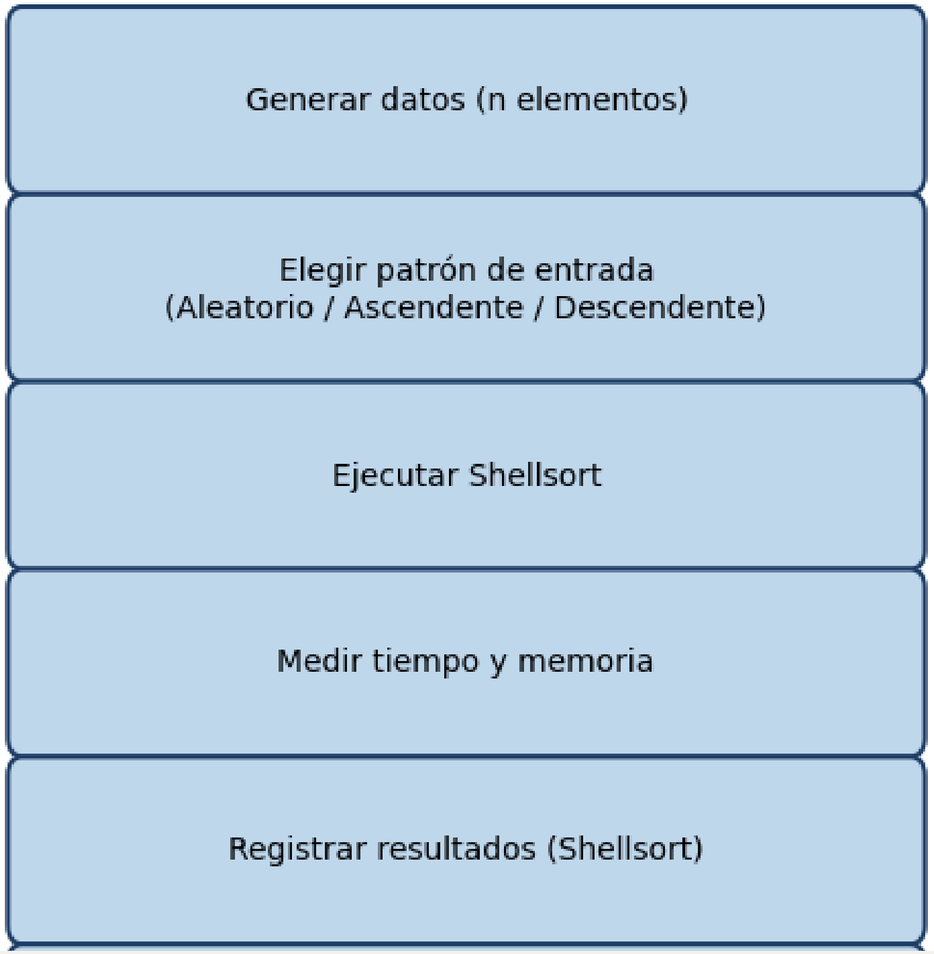
Teóricamente, Counting Sort debería superar a Shellsort en tiempo cuando el rango de datos k es comparable o menor que n, mientras que Shellsort debería ser más eficiente en espacio y aplicable a tipos de datos no enteros.

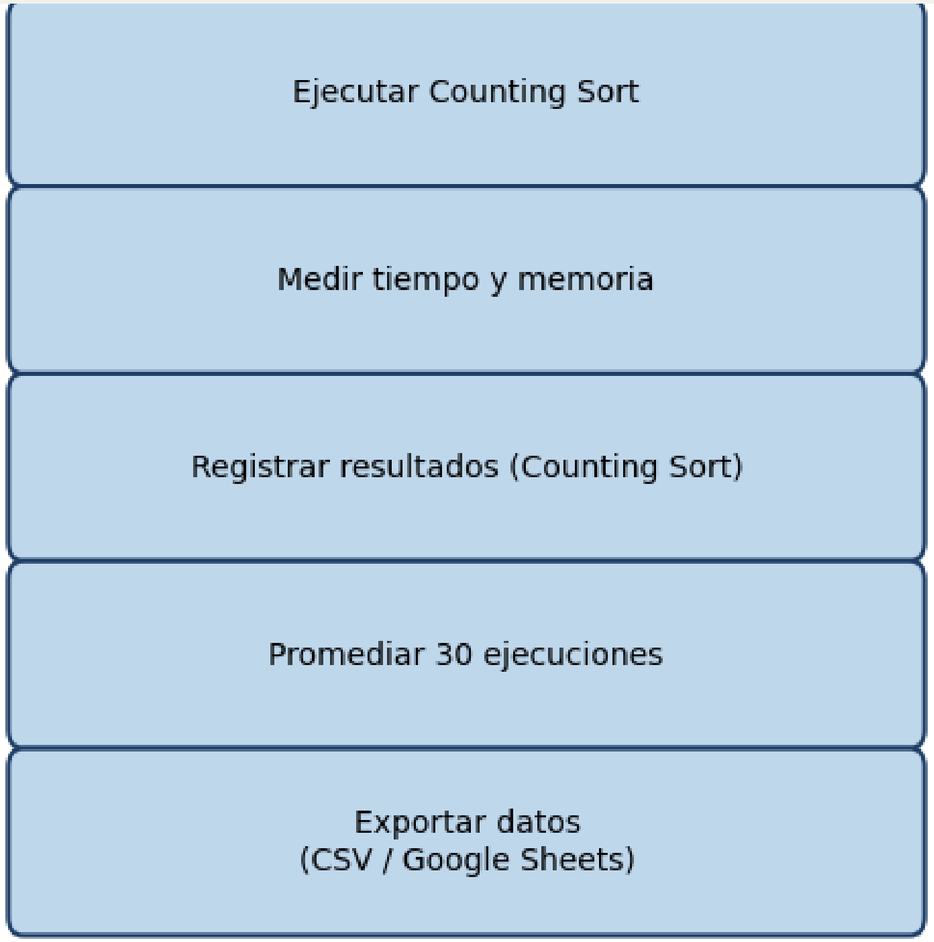
Algoritmo	Mejor caso	Promedio	Peor caso	Espacio
Shellsort	O(n log n)	O(n^(3/2))	$O(n^2)$	O(1)
Counting Sort	O(n + k)	O(n + k)	O(n + k)	O(n + k)

ENTORNO DE EJECUCIÓN

Procesador	Intel Core i5-1135G7 (4 núcleos, 2.4 GHz)
Memoria RAM	8 GB DDR4
Sistema operativo	Windows 11 Pro 64 bits
Lenguaje	C++
Editor	Visual Studio Code 1.91
Flags de compilación	optimización estándar -O2

MÉTRICAS Y PROCEDIMIENTOS DE MEDICIÓN:





ESTADÍSTICA DESCRIPTIVA DEL TIEMPO DE EJECUCIÓN

Tamaño (n)	Algoritmo	Media (ms)	Desv. Est. (σ)	CV (%)
1	Shellsort	185	9	486
1	Counting Sort	92	5	543
5	Shellsort	821	33	402
5	Counting Sort	357	16	448
10	Shellsort	1.734	81	467
10	Counting Sort	716	32	447
20	Shellsort	3.789	172	454
20	Counting Sort	1.412	61	432
50	Shellsort	9.547	428	448
50	Counting Sort	3.465	162	468

TENDENCIA DEL TIEMPO DE EJECUCIÓN:

Tamaño (n)	Shellsort (ms)	Counting Sort (ms)
1	185	92
5	821	357
10	1.734	716
20	3.789	1.412
50	9.547	3.465

USO DE MEMORIA:

Tamaño (n)	Shellsort (KB)	Counting Sort (KB)
1	102	248
5	124	317
10	150	399
20	196	545
50	279	732

EFICIENCIA RELATIVA

Tamaño (n)	Eficiencia (%)
1	2.011
5	2.299
10	2.422
20	2.684
50	2.756

RESUMEN GENERAL DE RESULTADOS:

Métrica	Shellsort	Counting Sort	Algoritmo más eficiente
Complejidad teórica	O(n^(3/2))	O(n + k)	Counting Sort
Tiempo promedio (ms)	3.255	1.248	Counting Sort
Desviación estándar (σ)	185	79	Counting Sort
Memoria (KB)	1.702	4.482	Shellsort
Estabilidad del CV (%)	451	467	Igual
Aplicabilidad	Datos generales	Rangos acotados	Depende del caso

CONCLUSION:

El análisis comparativo permite establecer que:

1. **Counting Sort es temporalmente más eficiente con datos numéricos de rango limitado.**

1. **Shellsort ofrece rendimiento equilibrado y menor uso de memoria.**

1. **No existe algoritmo universalmente superior; la selección debe considerar características de los datos y restricciones del sistema.**

Comprender la naturaleza del problema es esencial para escoger el algoritmo adecuado.

GRACIAS