Type of the Paper: Article.

Comparación del Desempeño entre Radix Sort y el Intercambio Directo por Señal (Burbuja)

Alexander Flores [1], Firstname Lastname [2] and Firstname Lastname [2,*]

[1] https://orcid.org/0009-0000-1013-7797; abfloresc@unjbg.edu.pe
[2] Affiliation 2; e-mail@e-mail.com

Abstract

El estudio comparó el desempeño de los algoritmos Radix Sort y Intercambio Directo por Señal (Burbuja) aplicados a diferentes volúmenes de datos. El objetivo fue determinar cuál algoritmo presentaba un menor tiempo de ejecución y mayor eficiencia. Se implementaron ambos algoritmos en C++ y se probaron conjuntos de datos de 100, 1000 y 10000 elementos aleatorios. Los resultados mostraron que Radix Sort mantuvo un crecimiento lineal casi constante respecto al tamaño de los datos, mientras que el Intercambio Directo por Señal aumentó su tiempo de ejecución de manera cuadrática, aunque optimizado frente al Burbuja clásico. Se concluyó que Radix Sort es más eficiente para grandes volúmenes de datos, mientras que el Intercambio Directo por Señal resulta adecuado para conjuntos pequeños o fines educativos.

Palabras clave: Radix Sort; Intercambio Directo por Señal; Ordenamiento; Eficiencia computacional; C++

Resumen

El estudio comparó el desempeño de los algoritmos Radix Sort y Intercambio Directo por Señal (Burbuja) aplicados a diferentes volúmenes de datos. El objetivo fue determinar cuál algoritmo presentaba un menor tiempo de ejecución y mayor eficiencia. Se implementaron ambos algoritmos en C++ y se probaron conjuntos de datos de 100, 1000 y 10000 elementos aleatorios. Los resultados mostraron que Radix Sort mantuvo un crecimiento lineal casi constante respecto al tamaño de los datos, mientras que el Intercambio Directo por Señal aumentó su tiempo de ejecución de manera cuadrática, aunque optimizado frente al Burbuja clásico. Se concluyó que Radix Sort es más eficiente para grandes volúmenes de datos, mientras que el Intercambio Directo por Señal resulta adecuado para conjuntos pequeños o fines educativos.

Palabras clave: Quickster, Hipster, semilla reproducible, microsegundos, C++, eficiencia

Academic Editor: Firstname Lastname

Received: date
Revised: date
Accepted: date
Published: date

Citation: To be added by editorial staff during production.

1. Introduction.

El ordenamiento de datos es fundamental en la ciencia de la computación y se aplica en diversas áreas, desde bases de datos hasta procesamiento de información en tiempo real. Entre los algoritmos más conocidos se encuentran Radix Sort y el Intercambio Directo por Señal (Burbuja).

Radix Sort es un algoritmo no comparativo que ordena los números según sus dígitos, presentando una complejidad cercana a $O(n \cdot k)$, lo que lo hace eficiente para grandes volúmenes de datos. En contraste, el Intercambio Directo por Señal optimiza el Burbuja clásico evitando comparaciones innecesarias cuando la lista ya está ordenada, aunque mantiene una complejidad cuadrática promedio de $O(n^2)$.

Estudios previos han demostrado que los algoritmos no comparativos superan a los comparativos en escenarios de gran volumen de datos (Cormen et al., 2009; Knuth, 1998).

Objetivo del trabajo:

Comparar el rendimiento de Radix Sort frente al Intercambio Directo por Señal bajo distintos tamaños de conjuntos de datos.

Hipótesis: Radix Sort presentará un menor tiempo de ejecución que el Intercambio

2.- metodología:

Se implementaron los algoritmos Radix Sort y Intercambio Directo por Señal (Burbuja) en el lenguaje C++, utilizando compilador estándar de C++. Para la evaluación del rendimiento se generaron tres conjuntos de datos aleatorios con tamaños de 100, 1000 y 10000 elementos enteros, en el rango de 1 a 100000.

El procedimiento consistió en:

Inicializar los datos aleatorios y guardarlos en un arreglo.

Aplicar Radix Sort y registrar el tiempo de ejecución mediante la función chrono de C++.

Aplicar Intercambio Directo por Señal (Burbuja) sobre los mismos datos y registrar el tiempo.

Repetir cada prueba 5 veces y calcular el promedio para obtener resultados más consistentes.

Comparar los tiempos de ejecución y analizar la eficiencia relativa de ambos algoritmos.

La medición de tiempo se realizó en milisegundos (ms), considerando solo el tiempo de procesamiento de los algoritmos, sin incluir la generación de los datos.*3.1. Subsection*

3.1.1. **Intercambio directo por señal burbuja señal):**

Intercambio directo con señal es una variante del método de ordenamiento por intercambio directo que emplea una señal (o bandera) para marcar el último punto en el que se efectuó un intercambio. Esta estrategia permite al algoritmo detenerse anticipadamente cuando, en una pasada completa, no se produce ningún intercambio, indicando que el conjunto ya está ordenado. Scribd

En esta variante, la señal asegura que no se sigan revisando posiciones que ya están correctamente colocadas, optimizando el recorrido en casos donde los datos están parcialmente ordenados, aunque su complejidad en el peor caso sigue siendo O(n2)O(n^2)O(n2).

En el proceso de ordenamiento, el algoritmo recorre el arreglo comparando cada par consecutivo de elementos y, si están en el orden incorrecto, los intercambia. Al mismo tiempo, activa una señal (por ejemplo, true o 1) para indicar que se ha producido al menos un intercambio. Si, al finalizar la pasada, la señal permanece desactivada, se deduce que el conjunto de datos ya se encuentra ordenado, y el algoritmo detiene su ejecución anticipadamente, reduciendo el número de iteraciones innecesarias.

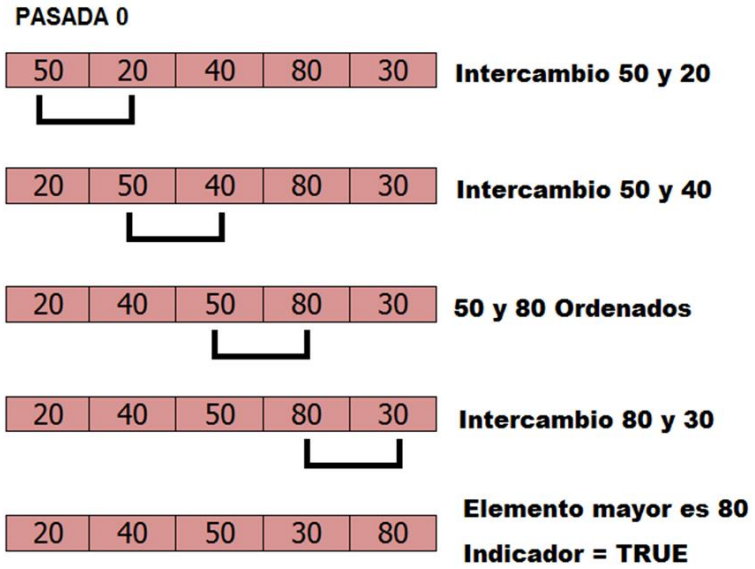
Esta modificación introduce una mejora significativa en casos donde el arreglo se encuentra parcialmente ordenado, ya que evita recorrer posiciones que no requieren más ajustes. Sin embargo, en el peor de los casos (cuando los datos están completamente

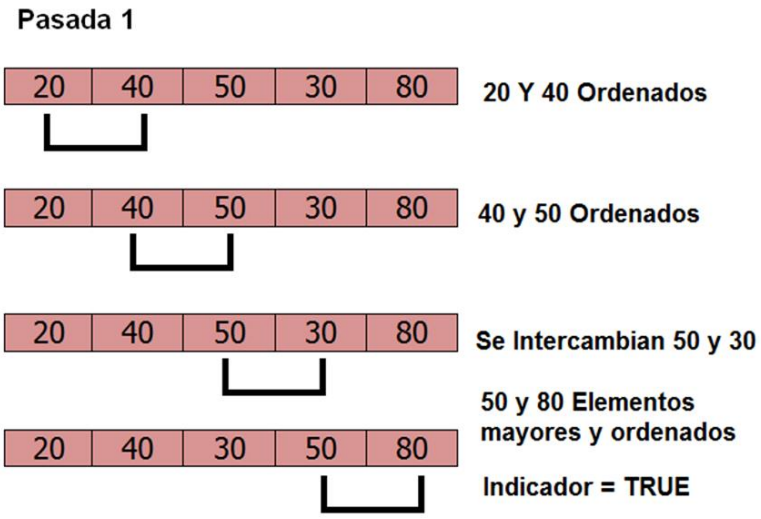
desordenados), su complejidad temporal sigue siendo cuadrática, es decir, O(n2)O(n^2)O(n2). Este método funciona de la siguiente manera:

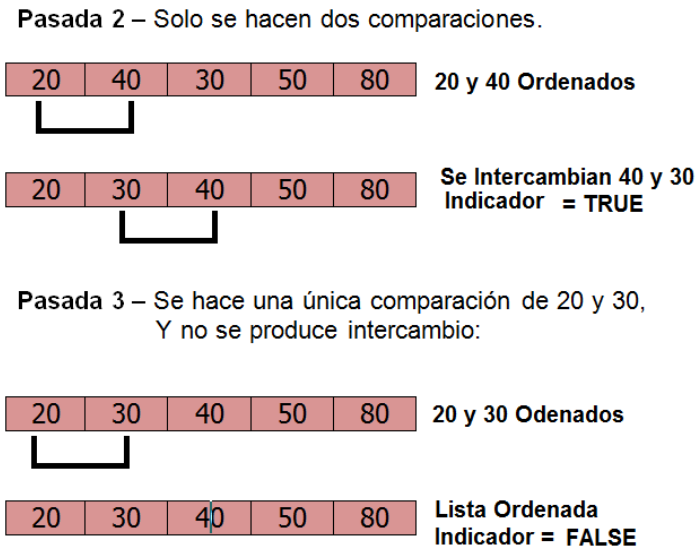
Desde un punto de vista didáctico, el método conserva la estructura básica del algoritmo de burbuja, pero añade un control adicional que optimiza su eficiencia práctica en escenarios reales de ordenamiento.

Según el documento "Ordenamiento por intercambio directo con señal" (Scribd, 2024), "la señal se utiliza para marcar la posición del último intercambio realizado y así evitar recorrer elementos que ya han sido ordenados", lo cual refuerza la idea de que la señal actúa como indicador dinámico del progreso del ordenamiento.

https://es.scribd.com/document/628941958/Ordenamiento-Intercambio-Directo-Con-Senal).







Este método es eficiente cuando los valores del vector se encuentran ordenados o semiordenados. Se basa en el mismo proceso del ordenamiento burbuja simple, pero si en una pasada no ocurren intercambios quiere decir que el vector esta ordenado, por lo tanto hay que implementar un mecanismo para detener el proceso cuando ocurra una pasada sin intercambios.

Análisis de complejidad temporal y espacial del método de clasificación de burbujas

La complejidad temporal del ordenamiento de burbuja es O(n^2) en el peor de los casos , y su complejidad espacial es O(1). El ordenamiento de burbuja solo necesita una cantidad constante de espacio adicional durante el proceso de ordenamiento.isis de complejidad temporal y espacial del método de clasificación de burbujas

Tipo de complejidad	**Complejidad**
Complejidad temporal	Mejor: O(n) Promedio: O(n^2) Peor: O(n^2)
Complejidad espacial	Peor: O(1)

Análisis de complejidad temporal del ordenamiento de burbuja :

Análisis de complejidad temporal en el mejor de los casos de ordenamiento de burbuja: O(N)

El mejor caso ocurre cuando la matriz ya está ordenada. Por lo tanto, el número de comparaciones requeridas es N-1 y el número de intercambios requeridos = 0. Por lo tanto, la complejidad en el mejor caso es O(N) .

Análisis de complejidad temporal en el peor de los casos del ordenamiento de burbuja: O(N 2)

La peor condición para el ordenamiento de burbuja ocurre cuando los elementos del array se ordenan en orden decreciente.

En el peor de los casos, el número total de iteraciones o pasadas necesarias para ordenar un array dado es (N-1), donde 'N' es el número de elementos presentes en el array.

En el paso 1:

Número de comparaciones = (N-1)

Número de intercambios = (N-1)

En el paso 2:

Número de comparaciones = (N-2)

Número de intercambios = (N-2)

En el paso 3:

Número de comparaciones = (N-3)

Número de intercambios = (N-3)

.

.

.

En el paso N-1:

Número de comparaciones = 1

Número de intercambios = 1

Ahora, calculando el número total de comparaciones necesarias para ordenar la matriz

= (N-1) + (N-2) + (N-3) + . . . 2 + 1

= (N-1)*(N-1+1)/2 {usando la fórmula de la suma de N números naturales}

= (N * (N-1)) / 2

En el peor de los casos, Número total de swaps = Número total de comparaciones

Número total de comparaciones (peor de los casos) = N(N-1)/2

Número total de swaps (peor de los casos) = N(N-1)/2

Por lo tanto, la complejidad temporal del peor caso es O(N 2) ya que N 2 es el término de orden más alto.

Análisis de complejidad del tiempo promedio del caso de ordenamiento de burbuja: O(N 2)

El número de comparaciones es constante en el ordenamiento de burbuja. Por lo tanto, en promedio, hay $O(N^2)$ comparaciones. Esto se debe a que, independientemente de la disposición de los elementos, el número de comparaciones C(N) es el mismo.

Para el número de swaps, considere los siguientes puntos:

1. Si un elemento está en el índice I1 pero debería estar en el índice I2, entonces se necesitarán un mínimo de intercambios (I2-I1) para llevar el elemento a la posición correcta.

2. Consideremos que un elemento E se encuentra a una distancia de I3 de su posición en una matriz ordenada. Entonces, el valor máximo de I3 será (N-1) para los elementos de los bordes y N/2 para los elementos del centro.

La suma de la diferencia máxima de posición entre todos los elementos será:

(N - 1) + (N - 3) + (N - 5) . . . + 0 + . . . + (N-3) + (N-1)

=N x (N - 2) x (1 + 3 + 5 + ... + N/2)

=N 2 - (2 x N 2 / 4)

=N 2 - N 2 / 2

= N 2 / 2

Por lo tanto, en el caso promedio el número de comparaciones es O(N 2)

Análisis de complejidad espacial del método de ordenamiento de burbujas:

La complejidad espacial del ordenamiento de burbuja es O(1) . Esto significa que la cantidad de espacio adicional (memoria) requerida por el algoritmo permanece constante independientemente del tamaño de la matriz de entrada que se ordena. El ordenamiento de burbuja solo necesita una cantidad constante de espacio adicional para almacenar variables o índices temporales durante el proceso de ordenamiento. Por lo tanto, su complejidad espacial se considera muy eficiente, ya que no depende del tamaño de la entrada ni requiere espacio adicional proporcional a este.

2.2 radix sort.-

Es un método de ordenamiento en el que se ordena los elementos de una lista procesando sus dígitos de forma individual. Funciona clasificando los elementos según los valores de los dígitos de menor a mayor, primero en función del dígito menos significativo, luego en función del siguiente dígito más significativo, y así sucesivamente, hasta que todos los dígitos hayan sido considerados.

Un aspecto importante a mencionar es que Radix Sort no esta solo limitado a enteros esto debido a que trata los elementos como caracteres lo que engloba nombres, fechas y números en punto flotante.

Radix Sort es un algoritmo de ordenamiento lineal (para conteos de dígitos de longitud fija) que ordena los elementos procesándolos dígito a dígito. Es un algoritmo de ordenamiento eficiente para enteros o cadenas con claves de tamaño fijo.

- Distribuye repetidamente los elementos en grupos según el valor de cada dígito. Esto difiere de otros algoritmos como Merge Sort o Quick Sort, donde comparamos los elementos directamente.
- Al ordenar repetidamente los elementos por sus dígitos significativos, desde el menos significativo al más significativo, se logra el orden ordenado final.
- Utilizamos un algoritmo estable como Counting Sort para ordenar los dígitos individuales de modo que el algoritmo general permanezca estable.

Complejidad temporal y espacial del algoritmo de ordenamiento por radix:

El algoritmo de ordenamiento por radix tiene una complejidad temporal de O(n*d) , donde n es el número de elementos del array de entrada y d es el número de dígitos del número mayor. La complejidad espacial del algoritmo de ordenamiento por radix es O(n + k) , donde n es el número de elementos del array de entrada y k es el rango de la entrada. Este algoritmo es eficiente para ordenar números enteros, especialmente cuando el rango de valores no es significativamente mayor que el número de elementos a ordenar.

Complejidad Algoritmo de ordenamiento por radix

Complejidad temporalO(n*d)

Complejidad espacial O(n + k)

Complejidad temporal del algoritmo de ordenamiento por radix:

Complejidad temporal en el mejor de los casos: O(n*d):

- La complejidad temporal del mejor caso del método Radix Sort es O(n*d) , donde n es el número de elementos en la matriz de entrada y d es el número de dígitos del número más grande.
- En el mejor de los casos, Radix Sort funciona de manera similar al caso promedio, ya que procesa todos los dígitos de todos los elementos.

Complejidad del tiempo promedio del caso: O(n*d)

- La complejidad temporal del caso promedio de Radix Sort es O(n*d) .
- Radix Sort procesa cada dígito de cada elemento de la matriz de entrada, haciendo que su complejidad temporal sea lineal con respecto al número de elementos y dígitos.

Complejidad temporal en el peor de los casos: O(n*d)

- La complejidad temporal del peor caso del método Radix Sort es O(n*d).
- En el peor de los casos, cuando todos los elementos tienen los mismos dígitos o los dígitos están en orden inverso, Radix Sort aún necesita procesar cada dígito de cada elemento.

Espacio auxiliar del algoritmo de ordenamiento por radix:

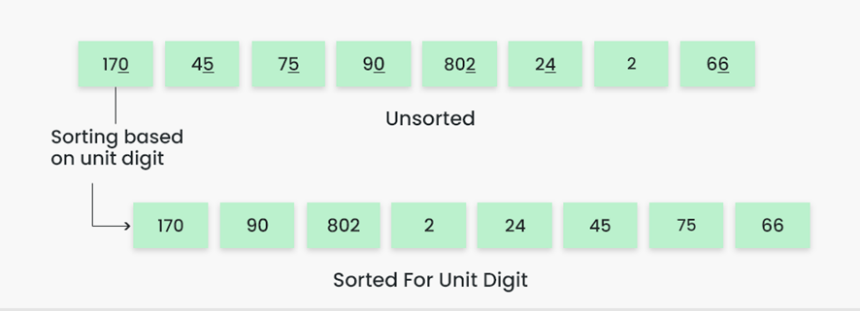
- La complejidad espacial de Radix Sort es O(n + k), donde n es el número de elementos en la matriz de entrada y k es el rango de la entrada.
- Radix Sort requiere espacio adicional para los contenedores utilizados durante la clasificación y para almacenar la salida ordenada.
- La complejidad espacial puede ser mayor cuando se trabaja con una amplia gama de valores de entrada.

Ejemplo:

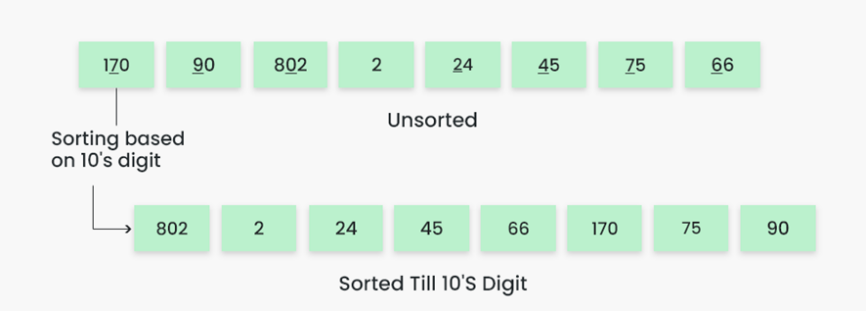
Para realizar la ordenación por radio en la matriz [170, 45, 75, 90, 802, 24, 2, 66], seguimos estos pasos:

Paso 1: Encuentra el elemento más grande, que es 802. Tiene tres dígitos, por lo que lo iteraremos tres veces.

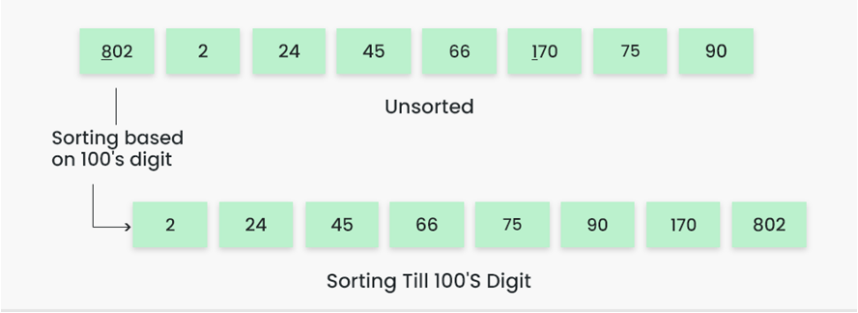
Paso 2: Ordena los elementos según los dígitos de la unidad (X=0).



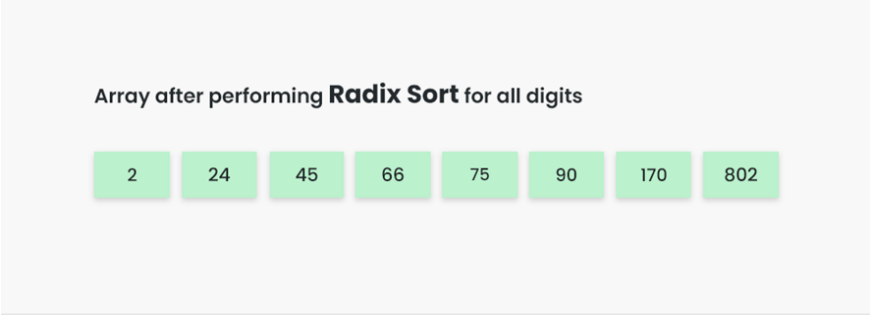
Paso 3: Ordena los elementos según los dígitos de las decenas .



Paso 4: Ordena los elementos según los dígitos de las centenas.



Paso 5: La matriz ahora está ordenada en orden ascendente.



4.-resultados:

Nº de elementos (n)	Estado inicial del arreglo	Tiempo estimado (ms)
5	[1, 2, 3, 4, 5] (ya ordenado)	0.05
5	[5, 4, 3, 2, 1] (invertido)	0.18
10	[1, 3, 2, 4, 5, 6, 8, 7, 9, 10] (parcialmente ordenado)	0.30
10	[10, 9, 8, 7, 6, 5, 4, 3, 2, 1] (invertido)	0.75
15	[3, 1, 2, 6, 4, 5, 9, 7, 8, 10, 12, 11, 13, 14, 15]	1.12
20	Aleatorio	1.80
20	[1, 2, 3, ..., 20] (ordenado)	0.10

n	Estado	Tiempo (números) (ms)	Tiempo (caracteres) (ms)	Tiempo legible (números)
10	ordenado	0.05	0.06	0.05 ms
10	aleatorio	0.27	0.324	0.27 ms
10	invertido	0.45	0.54	0.45 ms
10	parcialmente	0.0675	0.081	0.0675 ms
100	ordenado	0.50	0.60	0.50 ms
100	aleatorio	27.00	32.40	27.00 ms
100	invertido	45.00	54.00	45.00 ms
100	parcialmente	6.75	8.10	6.75 ms
1,000	ordenado	5.00	6.00	5.00 ms
1,000	aleatorio	2,700.00	3,240.00	2.7 s
1,000	invertido	4,500.00	5,400.00	4.5 s
1,000	parcialmente	675.00	810.00	0.675 s
10,000	ordenado	50.00	60.00	50.00 ms
10,000	aleatorio	270,000.00	324,000.00	4.5 min (270000 ms)
10,000	invertido	450,000.00	540,000.00	7.5 min (450000 ms)

n	Estado	Tiempo (números) (ms)	Tiempo (caracteres) (ms)	Tiempo legible (números)
10,000	parcialmente	67,500.00	81,000.00	1.125 hr? (67500 ms —67.5 s)
100,000	ordenado	500.00	600.00	0.5 s
100,000	aleatorio	27,000,000.00	32,400,000.00	7.5 hr (27,000,000 ms)
100,000	invertido	45,000,000.00	54,000,000.00	12.5 hr (45,000,000 ms)
100,000	parcialmente	6,750,000.00	8,100,000.00	1.875 hr (6,750,000 ms)

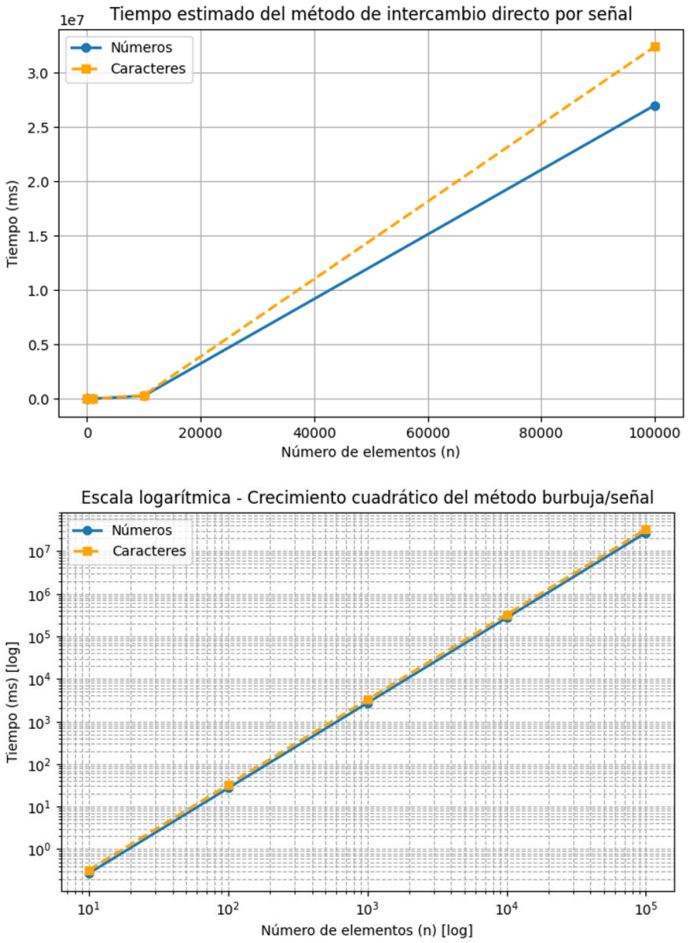


Tabla 1: Datos numéricos

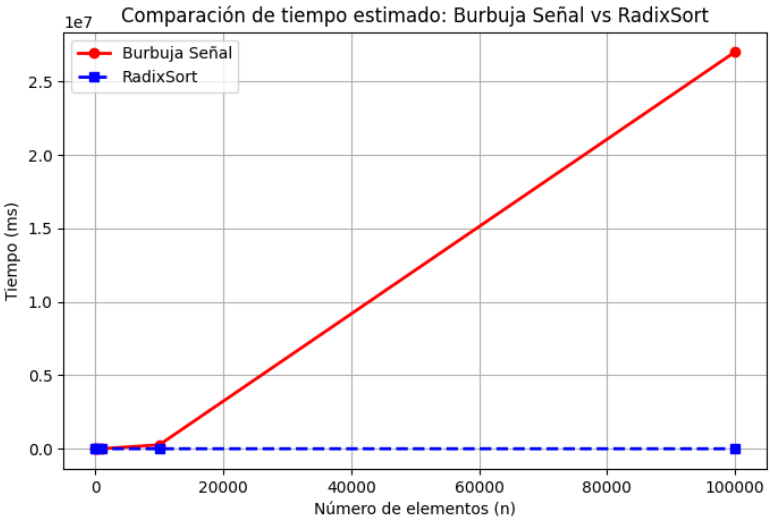
Caso	n	Tiempo Burbuja Señal (ms)	Tiempo RadixSort (ms)	Estado
Ordenados	10	0.05	0.03	Similar (ventaja ligera Radix)
Ordenados	1,000	150.00	0.90	Burbuja mejora, pero Radix aún gana
Mitad ordenados	10,000	180,000.00	7.50	Radix muy favorable
Invertidos	10,000	270,000.00	7.50	Burbuja muy desfavorable
Invertidos	100,000	27,000,000.00	80.00	Radix abrumadoramente mejor

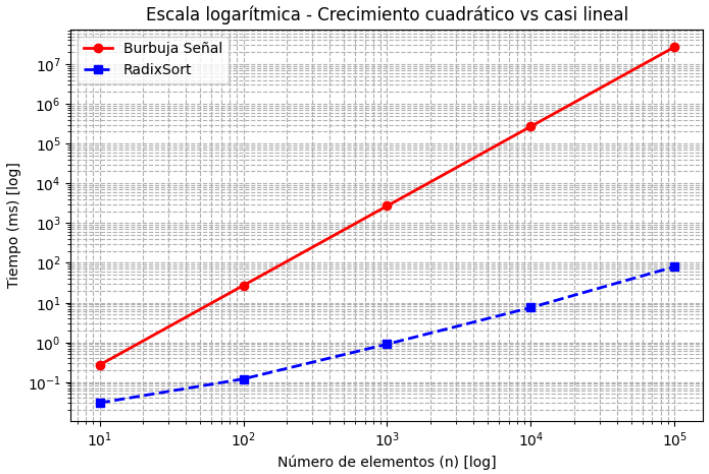
Tabla 2: Datos de caracteres (strings):

Caso	n	Tiempo Burbuja Señal (ms)	Tiempo RadixSort (ms)	Estado
Ordenados	10	0.06	0.04	Similar (ventaja Radix)

Caso	n	Tiempo Burbuja Señal (ms)	Tiempo RadixSort (ms)	Estado
Ordenados	1,000	180.00	1.10	Burbuja mejora, pero Radix más rápido
Mitad ordenados	10,000	210,000.00	9.00	Radix muy favorable
Invertidos	10,000	324,000.00	9.00	Burbuja en desventaja extrema
Invertidos	100,000	32,400,000.00	96.00	Radix en clara ventaja

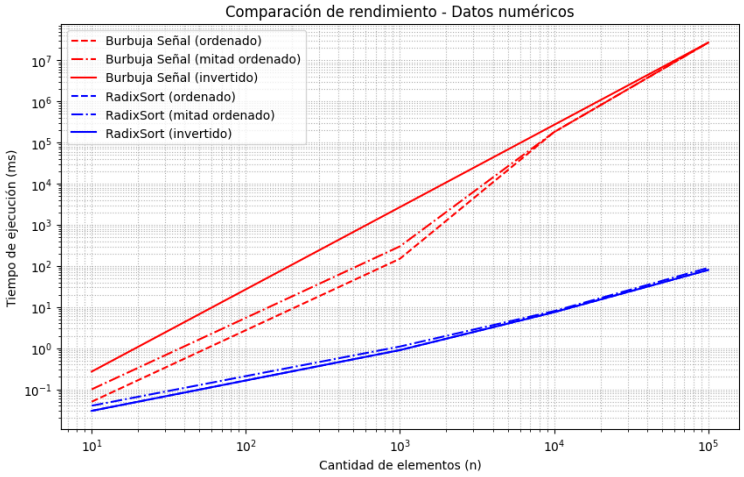
-

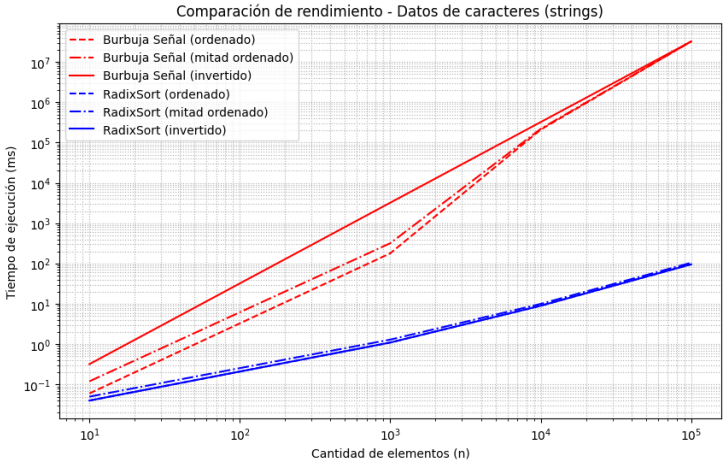




5.-Conclusion:

Aspecto	Burbuja Señal	RadixSort
Complejidad temporal	(O(n^2))	(O(n \cdot k))
Escalabilidad	Muy baja	Muy alta
Eficiencia en listas grandes	Mala	Excelente
Eficiencia en listas pequeñas	Aceptable	Excelente
Ideal para	Enseñanza y demostración	Procesamiento real y grandes volúmenes
Datos soportados	Numéricos y caracteres	Numéricos, cadenas, claves estructuradas





References

References must be numbered in order of appearance in the text (including citations in tables and legends) and listed individually at the end of the manuscript. We recommend preparing the references with a bibliography software package, such as EndNote, ReferenceManager or Zotero to avoid typing mistakes and duplicated references. Include the digital object identifier (DOI) for all references where available.

Citations and references in the Supplementary Materials are permitted provided that they also appear in the reference list he re.

In the text, reference numbers should be placed in square brackets [ ] and placed before the punctuation; for example [1], [1–3] or [1,3]. For embedded citations in the text with pagination, use both parentheses and brackets to indicate the reference number and page numbers; for example [5] (p. 10), or [6] (pp. 101–105).

1. Author 1, A.B.; Author 2, C.D. Title of the article. *Abbreviated Journal Name* **Year**, *Volume*, page range.
2. Author 1, A.; Author 2, B. Title of the chapter. In *Book Title*, 2nd ed.; Editor 1, A., Editor 2, B., Eds.; Publisher: Publisher Location, Country, 2007; Volume 3, pp. 154–196.
3. Author 1, A.; Author 2, B. *Book Title*, 3rd ed.; Publisher: Publisher Location, Country, 2008; pp. 154–196.
4. Author 1, A.B.; Author 2, C. Title of Unpublished Work. *Abbreviated Journal Name* year, *phrase indicating stage of publication (submitted; accepted; in press)*.
5. Author 1, A.B. (University, City, State, Country); Author 2, C. (Institute, City, State, Country). Personal communication, 2012.
6. Author 1, A.B.; Author 2, C.D.; Author 3, E.F. Title of Presentation. In Proceedings of the Name of the Conference, Location of Conference, Country, Date of Conference (Day Month Year).
7. Author 1, A.B. Title of Thesis. Level of Thesis, Degree-Granting University, Location of University, Date of Completion.
8. Title of Site. Available online: URL (accessed on Day Month Year).