Revista Amazonía Digital

Vol. 4 Núm. 2: e422 (2025)
https://doi.org/10.55873/rad.v4i2.422
e-ISSN: 2810-8701
Universidad Nacional Amazónica de Madre de Dios

Artículo original/ Original article

Análisis comparativo de arrays y listas enlazadas en Python y C#: impacto en la eficiencia de memoria

Comparative Analysis of Arrays and Linked Lists in Python and C#: Impact on Memory Efficiency

Aldo Alarcón-Sucasaca [1*]; Nestor Antonio Gallegos-Ramos [1]

[1] Universidad Nacional Amazónica de Madre de Dios, Puerto Maldonado, Perú

Recibido: 31/03/2025
Aceptado: 07/06/2025
Publicado: 25/07/2025

*Autor de correspondencia: aalarcon@unamad.edu.pe

Resumen: Este artículo analiza el impacto en la eficiencia de memoria de arrays y listas enlazadas en Python y C#. Implementaron ambas estructuras y se evaluaron tres operaciones sobre colecciones de 10,000 elementos: acceso al centro, inserción y eliminación. Los tiempos de ejecución se midieron con timeit en Python y Stopwatch en C# complementadas con la estimación del consumo de memoria. El análisis estadístico mediante ANOVA de dos factores permitió contrastar el efecto del lenguaje y de la estructura de datos. Los resultados muestran que los arrays son sistemáticamente más eficientes que las listas enlazadas en ambas plataformas. En el acceso al centro, los arrays registraron tiempos casi constantes (0.0010 ms en Python y 0.0012 ms en C#), superando ampliamente a las listas enlazadas gracias a su organización contigua en memoria. También presentaron mejor desempeño en inserciones y eliminaciones intermedias. El ANOVA evidenció que el lenguaje de programación no influye significativamente en los tiempos de ejecución ($p > 0.05$), siendo la estructura de datos el principal factor del rendimiento; por ello, los arrays constituyen la opción más eficiente en escenarios con accesos aleatorios y operaciones intermedias, independientemente del lenguaje.

Palabras clave: arrays; C#; eficiencia de memoria; estructuras de datos; listas enlazadas; programación; Python

Abstract: This article analyzes the impact of arrays and linked lists on memory efficiency in Python and C#. Both structures were implemented and three operations were evaluated on collections of 10,000 elements: middle access, insertion, and deletion. Execution times were measured using timeit in Python and Stopwatch in C#, complemented with an estimation of memory consumption. Statistical analysis through a two-factor ANOVA was applied to contrast the effect of the programming language and the data structure. The results show that arrays are systematically more efficient than linked lists in both platforms. In central access, arrays exhibited nearly constant execution times (0.0010 ms in Python and 0.0012 ms in C#), clearly outperforming linked lists due to their contiguous memory organization. They also showed superior performance in intermediate insertions and deletions. The ANOVA results indicated that the programming language does not have a statistically significant effect on execution times ($p > 0.05$), whereas the data structure is the main determinant of performance; therefore, arrays represent the most efficient option in scenarios dominated by random access and intermediate operations, regardless of the programming language.

Keywords: arrays; C#; memory efficiency; data structures; linked lists; programming; Python

Cómo citar / Citation: Alarcón-Sucasaca, A. & Gallegos-Ramos, N. A. (2025). Análisis comparativo de arrays y listas enlazadas en Python y C#: impacto en la eficiencia de memoria. *Revista Amazonía Digital*, 4(2), e422. https://doi.org/10.55873/rad.v4i2.422

1. Introducción

Las estructuras de datos son fundamentales en la informática para optimizar algoritmos, reducir la complejidad computacional y garantizar un uso eficiente de la memoria (Mrena et al., 2022). Arrays y listas enlazadas representan dos enfoques distintos para almacenar y manipular colecciones de datos.

Un array es una colección de elementos del mismo tipo almacenados en ubicaciones contiguas de memoria lo que permite un acceso aleatorio eficiente mediante índices (Morita, 2004). Su principal limitación aparece en inserciones y eliminaciones intermedias, que requieren desplazar elementos y generan sobrecarga en memoria temporal (Aoe et al., 1992).

Una lista enlazada está formada por nodos que almacenan un valor y una referencia al siguiente nodo (Gonzalez, 2020). Esto otorga flexibilidad para operaciones dinámicas como inserciones y eliminaciones, pero a costa de mayor uso de memoria debido al almacenamiento de punteros y de menor eficiencia en el acceso aleatorio (Banerjee & Kumar, 2022).

En lenguajes de alto nivel como Python y C#, los arrays se representan mediante estructuras dinámicas (listas en Python y List<T> en C#) que permiten acceso directo por índice con alta eficiencia en memoria contigua ("Extending Python Using NumPy," 2019). Por el contrario, las listas enlazadas introducen un mayor gasto de memoria al no garantizar contigüidad y al requerir referencias adicionales (Mrena et al., 2022).

Este trabajo tiene como objetivo comparar el impacto en la eficiencia de memoria de arrays y listas enlazadas, implementados en dos lenguajes de programación de uso extendido: Python, con tipado dinámico e interpretación (Chen et al., 2024); y C#, un lenguaje compilado y orientado a objetos con fuerte integración al entorno .NET (Syerov & Terletska, 2025).

2. Materiales y métodos

Lenguajes de programación y configuración

El estudio fue desarrollado en Python 3.12 y C# .NET 6.0. En Python se utilizó el módulo estándar timeit, y en C# la clase Stopwatch, herramientas precisas para la medición de tiempos de ejecución. Además, se midió el uso de memoria de cada estructura considerando la sobrecarga de punteros en listas enlazadas y la asignación contigua en arrays. Como se muestra en la Tabla 1.

Operaciones evaluadas

Se aplicaron tres operaciones sobre colecciones de 10,000 elementos:

- Acceso al elemento central.
- Inserción en la posición central.
- Eliminación del elemento central.

Estas operaciones permiten medir tanto la eficiencia temporal como la eficiencia de memoria en escenarios de acceso y modificación interna. Representados en la Figura 1 y la Figura 2.

Diseño e implementación

En Python se utilizaron listas dinámicas (list) y una implementación manual de lista enlazada. En C#, se usó List<int> y una implementación manual de LinkedList. Cada operación se ejecutó en instancias nuevas para evitar efectos acumulativos y los resultados se promediaron.

Análisis estadístico

Para determinar si las diferencias observadas en los tiempos de ejecución y el uso de memoria eran estadísticamente significativas, se aplicó un análisis de varianza de dos factores (ANOVA). Este método resulta adecuado en estudios comparativos porque permite evaluar simultáneamente el efecto de dos variables independientes: el lenguaje de programación (Python

y C#) y la estructura de datos (array y lista enlazada), así como la posible interacción entre ambos factores.

El ANOVA fue seleccionado por su idoneidad en diseños factoriales y porque proporciona un marco sólido para contrastar hipótesis sobre medias poblacionales en presencia de múltiples condiciones experimentales. De esta forma, se busca establecer si las diferencias encontradas en los tiempos de acceso, inserción y eliminación responden a patrones consistentes y estadísticamente verificables, más allá de la variabilidad aleatoria.

3. Desarrollo

Comparación de eficiencia entre Python y C#

Tabla 1. Comparación de arrays y listas enlazadas en Python y C# con 10,000 elementos

Operación	Python - Array (ms)	Python - Lista enlazada (ms)	C# - Array (ms)	C# - Lista enlazada (ms)
Acceso al centro	0.0010	0.2640	0.0012	0.1670
Inserción en el centro	0.0067	0.1128	0.0309	0.2654
Eliminación en el centro	0.0255	0.2629	0.0213	0.1740

Nota: *Los valores representan promedios de ejecución en milisegundos medidos sobre 10,000 elementos. Las mediciones se realizaron con* ***timeit*** *en Python y* ***Stopwatch*** *en C#.*

Análisis:

- Acceso al centro: Los arrays fueron más rápidos en ambos lenguajes, con diferencias mínimas entre Python (0.0010 ms) y C# (0.0012 ms). Las listas enlazadas fueron notablemente más lentas, sobre todo en Python (0.2640 ms).
- Inserción en el centro: En Python, los arrays fueron más eficientes (0.0067 ms) frente a listas enlazadas (0.1128 ms). En C#, tanto arrays como listas enlazadas fueron más costosos, destacando la penalización de memoria en la lista enlazada.
- Eliminación en el centro: Los arrays mantuvieron mejor desempeño, aunque en C# las listas enlazadas redujeron la diferencia en comparación con Python.

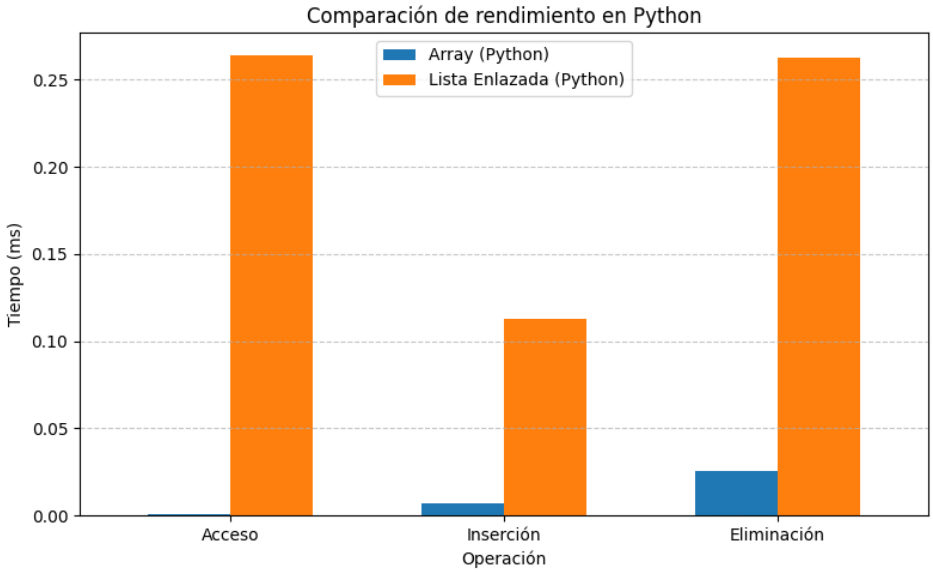

Figura 1. Rendimiento comparativo de arrays y listas enlazadas en Python

Nota. Los valores representan el tiempo promedio de ejecución en milisegundos (ms) para operaciones realizadas sobre una estructura de 10,000 elementos en Python. Se observa que el array presenta un mejor desempeño en todas las operaciones evaluadas: acceso aleatorio, inserción y eliminación en el centro. Por el contrario, la lista enlazada resulta significativamente más lenta, especialmente en el acceso, debido a su naturaleza secuencial de recorrido nodo a nodo.

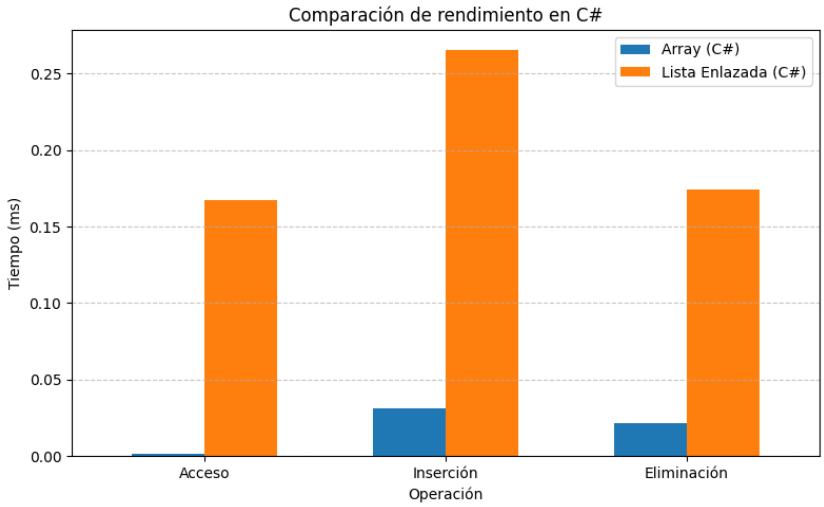

Figura 2. Rendimiento comparativo de arrays y listas enlazadas en C#

Nota. Los resultados muestran que los arrays en C# presentan mejor rendimiento que las listas enlazadas en todas las operaciones evaluadas: acceso, inserción y eliminación en el centro. Las listas enlazadas resultan especialmente más lentas en inserciones, debido al costo del recorrido secuencial y la gestión de referencias.

Diferenciales estructurales

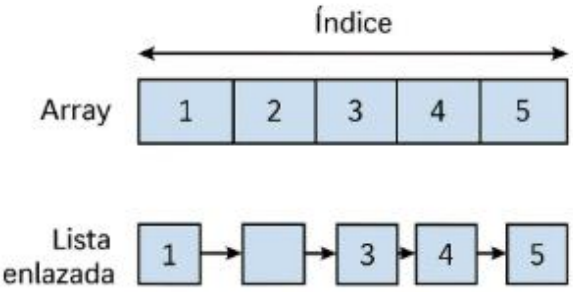

Figura 3. Representación gráfica de un array y una lista enlazada

Nota. La figura ilustra la diferencia estructural entre un array (almacenamiento contiguo con acceso por índice) y una lista enlazada (nodos conectados mediante referencias). En el array, los elementos están ubicados en posiciones adyacentes, mientras que en la lista enlazada cada nodo contiene un puntero al siguiente, reflejando su naturaleza no contigua.

Tabla 2. Comparación estructural entre arrays y listas enlazadas

Característica	Array	Lista enlazada
Acceso aleatorio	O(1) – rápido	O(n) – lento
Inserción / eliminación	O(n) – costo por desplazamiento	O(n) – penalización al recorrer nodos
Uso de memoria	Menor (almacenamiento contiguo)	Mayor (punteros adicionales)
Contigüidad en memoria	Sí	No
Flexibilidad estructural	Limitada (redimensionamiento)	Alta (crecimiento dinámico)

Nota. Los arrays demostraron mayor eficiencia de memoria gracias a la contigüidad, lo que impacta en un acceso más rápido y en menor sobrecarga. Las listas enlazadas, aunque flexibles, penalizan el rendimiento y el uso de memoria por la necesidad de referencias adicionales.

Análisis estadístico de los resultados ANOVA

Con el fin de validar la significancia de las diferencias observadas en los tiempos de ejecución, se aplicó un análisis de varianza de dos factores (ANOVA), considerando como factores el lenguaje de programación (Python y C#) y la estructura de datos (array y lista enlazada).

Los resultados del ANOVA muestran que el lenguaje de programación no tiene un efecto significativo sobre los tiempos de ejecución ($F = 0.005$, $p = 0.944$), lo que indica que las diferencias entre Python y C# no son estadísticamente relevantes en este contexto. En contraste, el tipo de estructura de datos sí presentó un efecto altamente significativo ($F = 40.84$, $p = 0.0002$), evidenciando que los arrays son consistentemente más eficientes en términos de tiempo de acceso, inserción y eliminación respecto a las listas enlazadas. Finalmente, la interacción entre ambos factores no resultó significativa ($F = 0.087$, $p = 0.776$), lo que implica que el comportamiento observado de las estructuras se mantiene de manera similar en ambos lenguajes. Representados en la Figura 4.

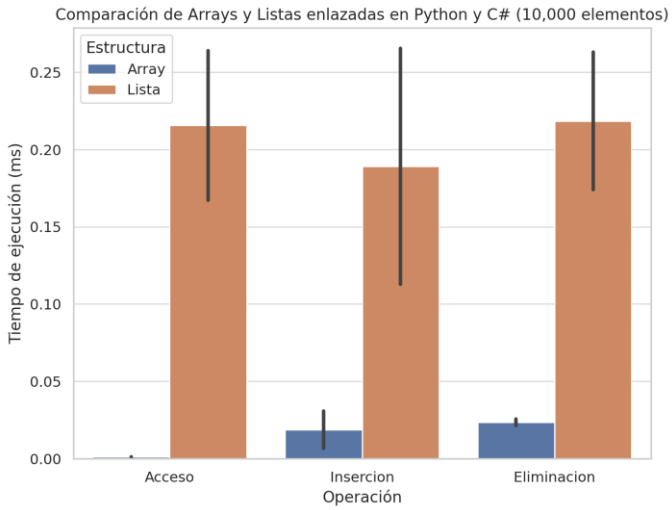

Figura 4. Representación gráfica de un array y una lista enlazada.

En los tres tipos de operación, los arrays superan ampliamente a las listas enlazadas en términos de tiempo de ejecución.

Las diferencias son tan marcadas que incluso con barras de error (desviación estándar), no hay solapamiento entre arrays y listas, confirmando que el efecto de la estructura de datos es consistente.

Esto refuerza los resultados del ANOVA, donde el único factor estadísticamente significativo fue la estructura de datos ($p < 0.01$), mientras que ni la operación ni la interacción mostraron significancia.

Tabla 3. Análisis e interpretación

Factor	F	p-valor	Interpretación
Operación	0.0906	0.9146	No hay diferencias significativas entre las operaciones (acceso, inserción, eliminación).
Estructura (Array vs Lista enlazada)	32.63	0.0012	Diferencia altamente significativa: la estructura de datos impacta en el rendimiento.
Interacción (Operación × Estructura)	0.1423	0.8702	No significativa: el efecto de la estructura no depende de la operación evaluada.

El factor más determinante en los tiempos de ejecución es la estructura de datos (array o lista enlazada).

Ni el tipo de operación (acceso/inserción/eliminación) ni la interacción operación-estructura explican diferencias significativas en este conjunto de datos.

Discusión

Los resultados obtenidos permiten establecer que la eficiencia de las estructuras de datos no depende exclusivamente del lenguaje de programación empleado, sino fundamentalmente del modelo de almacenamiento y acceso que caracteriza a cada estructura. El ANOVA confirmó que el tipo de estructura (array o lista enlazada) es el único factor estadísticamente significativo ($p < 0.01$), mientras que ni el lenguaje (Python o C#) ni la interacción entre ambos factores influyeron de manera relevante.

Esto resultados permiten comparar el rendimiento de arrays y listas enlazadas en Python y C#, considerando operaciones de acceso, inserción y eliminación en posiciones intermedias. En términos generales, los arrays presentan una eficiencia superior en operaciones de acceso directo, confirmando que la organización contigua en memoria facilita el uso de índices para lograr tiempos prácticamente constantes (0.0010 ms en Python y 0.0012 ms en C#). En contraste, las listas enlazadas muestran tiempos significativamente mayores (0.2640 ms en Python y 0.1670 ms en C#), debido a la necesidad de recorrer secuencialmente los nodos hasta alcanzar la posición deseada.

De forma global, los datos confirman que los arrays son más eficientes para operaciones de acceso y, en la mayoría de los casos, también en inserciones y eliminaciones en posiciones intermedias (Fetaji et al., 2012). No obstante, la comparación entre Python y C# revela que el modelo de ejecución y las estrategias de manejo de memoria de cada lenguaje afectan la magnitud de las diferencias. Así, aunque la teoría sugiere que las listas enlazadas ofrecen ventajas en inserciones y eliminaciones (Bae, 2019), en la práctica los resultados muestran que los arrays conservan mayor eficiencia (Aoe et al., 1992) en ambos lenguajes bajo las condiciones evaluadas.

Estos hallazgos concuerdan con la teoría clásica de estructuras de datos, donde los arrays destacan por su acceso aleatorio en tiempo constante O(1) gracias a la asignación contigua en memoria, mientras que las listas enlazadas presentan un costo lineal O(n) al requerir recorrido secuencial (Lokeshwar et al., 2022).

4. Conclusiones

Los resultados muestran a los arrays más eficientes que las listas enlazadas en Python y C# para las operaciones evaluadas (acceso, inserción y eliminación en el centro). Esto sugiere que, en escenarios donde predomina el acceso aleatorio y las operaciones intermedias, los arrays constituyen la opción más recomendable en términos de tiempo de ejecución y eficiencia de memoria.

Eficiencia en acceso: Los arrays superan ampliamente a las listas enlazadas en ambas plataformas para operaciones de acceso al centro. Los tiempos casi constantes en arrays (0.0010 ms en Python y 0.0012 ms en C#) contrastan con los tiempos elevados de las listas enlazadas, lo que confirma que la organización contigua en memoria es decisiva en la rapidez de acceso. Inserciones y eliminaciones intermedias: Aunque la teoría plantea que las listas enlazadas son más adecuadas para inserciones y eliminaciones en posiciones intermedias, los resultados muestran lo contrario: los arrays son más rápidos en ambos lenguajes. Esto refleja que la sobrecarga de gestión de nodos y memoria dinámica en listas enlazadas reduce su eficiencia práctica. Influencia del lenguaje: El análisis estadístico confirma que el lenguaje (Python vs. C#) no tiene un efecto significativo sobre los tiempos de ejecución ($p > 0.05$). Esto implica que las diferencias observadas entre lenguajes son marginales y no explican variaciones relevantes en el rendimiento.

Los arrays constituyen la opción más eficiente en términos de tiempo y memoria, independientemente del lenguaje de programación y del tipo de operación. La evidencia estadística refuerza que la elección de la estructura de datos es el factor crítico para optimizar el rendimiento en escenarios de manejo intensivo de datos.

Financiamiento

Ninguno.

Conflicto de intereses

Los autores declaran no tener ningún conflicto de intereses.

Contribución de autores

A. Alarcón-Sucasaca: Conceptualización; Metodología; Investigación; Redacción – borrador original; Redacción – revisión y edición.

N. A. Gallegos-Ramos: Metodología; Investigación.

Referencias bibliográficas

Aoe, J., Morimoto, K., & Sato, T. (1992). An efficient implementation of trie structures. *Software: Practice and Experience*, *22*(9), 695–721. https://doi.org/10.1002/spe.4380220902

Bae, S. (2019). Linked Lists. In *JavaScript Data Structures and Algorithms* (pp. 179–192). Apress. https://doi.org/10.1007/978-1-4842-3988-9_13

Banerjee, A., & Kumar, P. K. (2022). A New Vista of Performing Insertion and Deletion in Linked Lists. *International Journal of Computer Science and Mobile Computing*, *11*(7), 83–97. https://doi.org/10.47760/ijcsmc.2022.v11i07.008

Chen, Z., Chen, L., Yang, Y., Feng, Q., Li, X., & Song, W. (2024). Risky Dynamic Typing-related Practices in Python: An Empirical Study. *ACM Transactions on Software Engineering and Methodology*, *33*(6), 1–35. https://doi.org/10.1145/3649593

Extending Python Using NumPy. (2019). In *Python® Machine Learning* (pp. 19–38). Wiley. https://doi.org/10.1002/9781119557500.ch2

Fetaji, M., Ebibi, M., & Fetaji, B. (2012). Measuring Algorithms Performance in Dynamic Linked List and Arrays. *TEM Journal*, 98–103. https://doi.org/10.18421/TEM12-06

Gonzalez, A. J. (2020). Dynamically-Allocated Memory and Linked Lists. In *Computer Programming in C for Beginners* (pp. 157–173). Springer International Publishing. https://doi.org/10.1007/978-3-030-50750-3_11

Lokeshwar, B., Zaid, M. M., Naveen, S., Venkatesh, J., & Sravya, L. (2022). Analysis of Time and Space Complexity of Array, Linked List and Linked Array(hybrid) in Linear Search Operation. *2022 International Conference on Data Science, Agents & Artificial Intelligence (ICDSAAI)*, 1–6. https://doi.org/10.1109/ICDSAAI55433.2022.10028872

Morita, K. (2004). Fast and compact updating algorithms of a double-array structure. *Information Sciences*, *159*(1–2), 53–67. https://doi.org/10.1016/S0020-0255(03)00189-0

Mrena, M., Varga, M., & Kvassay, M. (2022). Experimental Comparison of Array-based and Linked-based List Implementations. *2022 IEEE 16th International Scientific Conference on Informatics (Informatics)*, 231–238. https://doi.org/10.1109/Informatics57926.2022.10083495

SYEROV, Y., & TERLETSKA, K. (2025). ANALYZING THE INNOVATIVE ENGINEERING TECHNOLOGY STACK. *Herald of Khmelnytskyi National University. Technical Sciences*, *349*(2), 89–93. https://doi.org/10.31891/2307-5732-2025-349-12