# ANÁLISIS COMPARATIVO DEL RENDIMIENTO DE TIEMPO Y COSTO COMPUTACIONAL

## QUICKSORT VS HEAPSORT

Dany Ronaldo Muriel Herencia 2024-119048
Anette Quispe
Jair Mendoza

# Índice

# INTRODUCCIÓN

## LOS ALGORITMOS DE ORDENAMIENTO SON FUNDAMENTALES PARA EVALUAR LA EFICIENCIA COMPUTACIONAL.

- Entre los más estudiados destacan QuickSort (Hoare, 1962) y HeapSort (Williams, 1964).
- Ambos poseen complejidad teórica O(*n log n*), pero su comportamiento real varía según el hardware y la distribución de los datos.

## JUSTIFICACIÓN

1

La eficiencia de un algoritmo no depende solo de su complejidad teórica, sino también de cómo interactúa con el hardware y los datos.

2

QuickSort y HeapSort, aunque similares en teoría, muestran diferencias reales de rendimiento según el tipo de entrada y las condiciones del entorno.

3

Analizar ambos métodos permite identificar cuál resulta más eficiente o estable en distintos escenarios, aportando criterios para optimizaciones prácticas en ingeniería de software.

## HIPÓTESIS

1

QuickSort mostrará mayor velocidad en conjuntos de datos medianos, gracias a su menor costo de operaciones.

2

HeapSort ofrecerá mayor estabilidad y consistencia en situaciones de alta carga o con datos duplicados.

3

Ningún método es superior en todos los casos: su desempeño depende del contexto computacional.

# METODOLOGÍA GENERAL

| | |
|---|---|
| LENGUAJE | C++ |
| COMPILADOR | MINGW G++ |

- Se compararon QuickSort (Quickster) y HeapSort (Hipster) bajo condiciones controladas.
- Los datos se generaron mediante la función rand() con semilla reproducible (srand(42)), garantizando la misma entrada para ambos algoritmos.
- Se analizaron tres tamaños de arreglos:
  - Pequeño: 1 000 elementos
  - Mediano: 10 000 elementos
  - Grande: 100 000 elementos

# PROCEDIMIENTO EXPERIMENTAL

# RESULTADOS DE QUICKSORT (N=1000)

| Patrón de entrada | Tiempo promedio (ms) | Desv. estándar (ms) |
|---|---|---|
| Aleatorio uniforme | **67** | 254 |
| Ordenado ascendente | 33 | 183 |
| Ordenado descendente | **0** | 0 |
| Casi ordenado (5%) | **0** | 0 |
| Duplicados (10–20%) | 33 | 183 |

- El mejor rendimiento se observó en los patrones ordenado descendente y casi ordenado (5%), con tiempo 0.000 ms.
- El caso más lento fue el aleatorio uniforme (0.067 ms), con mayor variabilidad.
- El pivote central utilizado permite a QuickSort manejar bien los datos parcialmente ordenados.
- Las diferencias son pequeñas por el tamaño reducido del conjunto (N = 1000).

# RESULTADOS HEAPSORT (N=1 000)

| Patrón de entrada | Tiempo promedio (ms) | Desv. estándar (ms) |
|---|---|---|
| Aleatorio uniforme | **200** | 407 |
| Ordenado ascendente | 133 | 345 |
| Ordenado descendente | **100** | 305 |
| Casi ordenado (5%) | 133 | 345 |
| Duplicados (10–20%) | 133 | 345 |

- HeapSort mostró tiempos ligeramente mayores, pero más consistentes en todos los patrones.
- El mejor caso fue el ordenado descendente (0.100 ms).
- El peor caso fue el aleatorio uniforme (0.200 ms).
- Su rendimiento se mantiene estable, independiente del orden inicial de los datos.
- Confirma la naturaleza teórica de HeapSort: O(n log n) estable.

# COMPARATIVA (N=10 000)

| Algoritmo | Mejor caso (ms) | Peor caso (ms) | Media general | Comporta miento |
|---|---|---|---|---|
| **QuickSort** | **133** | **700** | Rápido | Alta variabilidad |
| **HeapSort** | **1.567** | **2.067** | Más lento | Estable y predecible |

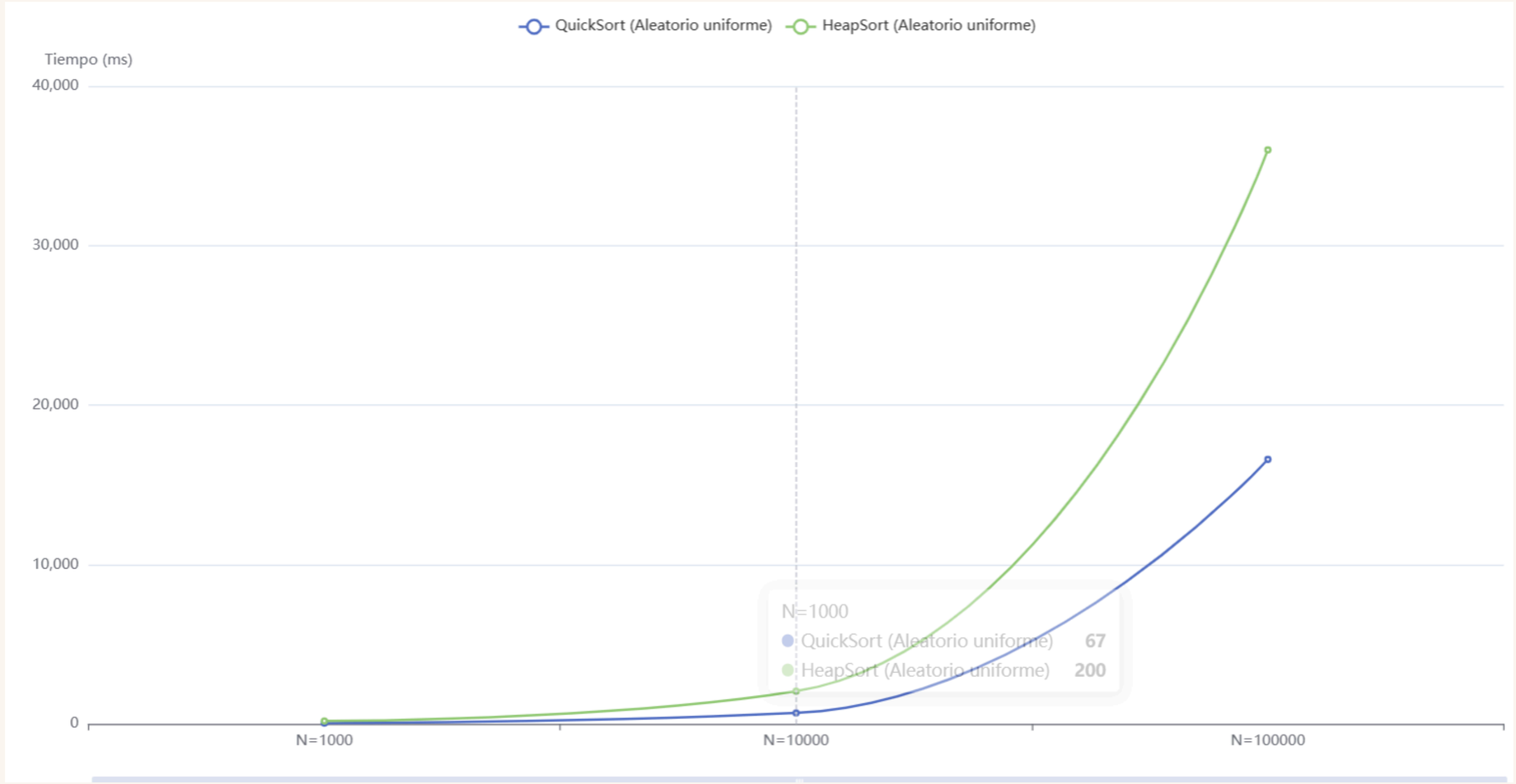
- QuickSort mejora el tiempo promedio, pero su desempeño fluctúa según el patrón de entrada.
- HeapSort mantiene rendimiento constante, sin depender del orden inicial de los datos.
- En datos grandes (N = 10 000):
  - QuickSort sigue siendo más veloz,
  - HeapSort es más robusto y predecible.
- Las diferencias confirman las ventajas teóricas de cada método.

# COMPARATIVA (N=100 000)

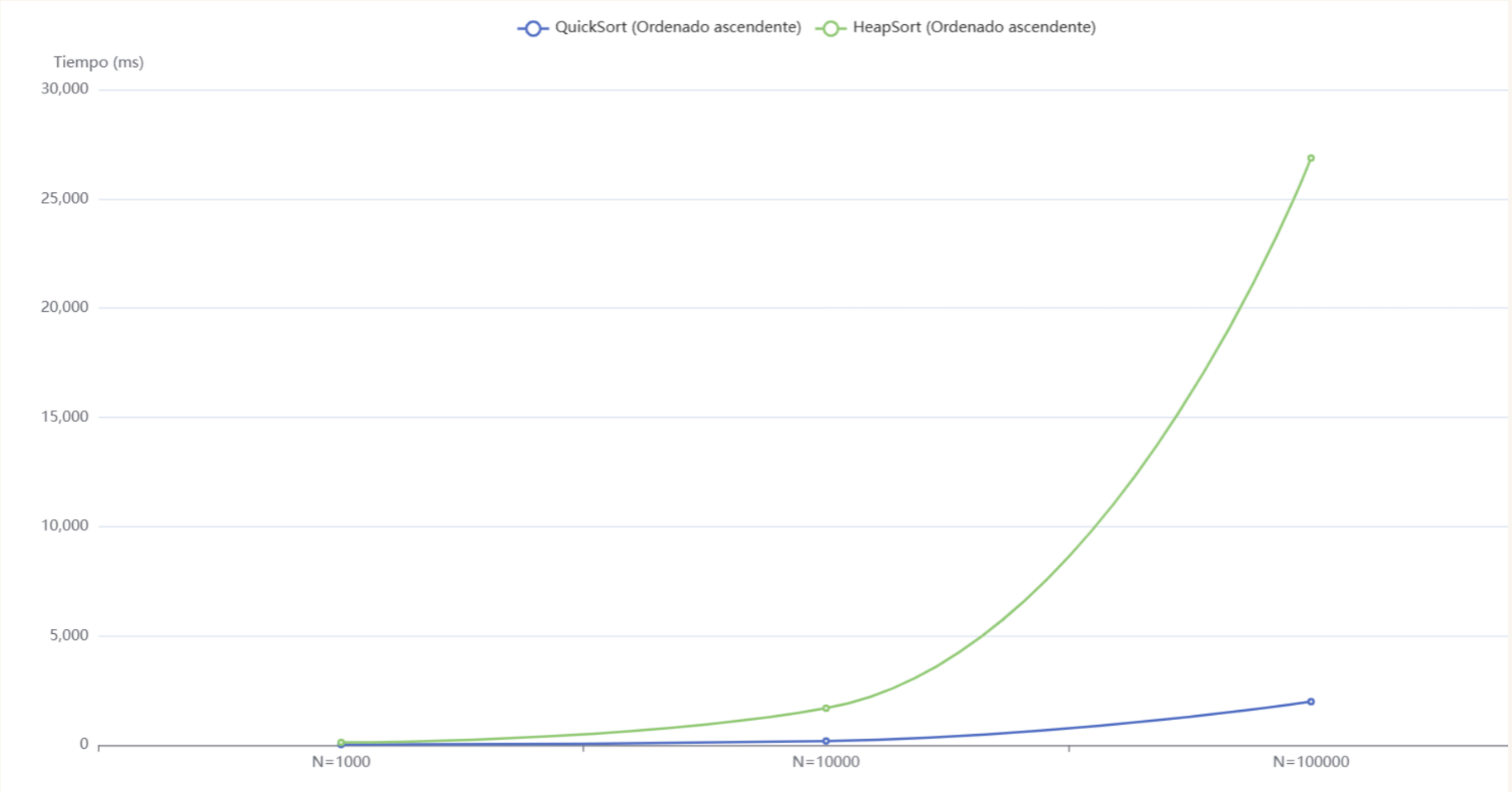
| Algoritmo | Aleatorio | Ascendente | Descendente | Casi ordenado (5%) | Duplicados (10–20) |
|---|---|---|---|---|---|
| **QuickSort (100 000)** | 13.667 | 7.333 | 6.333 | 6.333 | 6.667 |
| **HeapSort (100 000)** | 26.667 | 19.667 | 19.000 | 19.333 | 19.667 |

- Con 100 000 elementos, las diferencias de rendimiento se amplían notablemente.
- QuickSort mantiene tiempos promedio 3 a 4 veces menores que HeapSort.
- Sin embargo, HeapSort conserva un comportamiento uniforme, sin importar el patrón de entrada.
- La variación en QuickSort se debe a la recursión y la elección del pivote, aunque sigue siendo el más eficiente en promedio.
- Los resultados confirman la ventaja práctica de QuickSort para volúmenes grandes, siempre que los datos no provoquen particiones desbalanceadas.

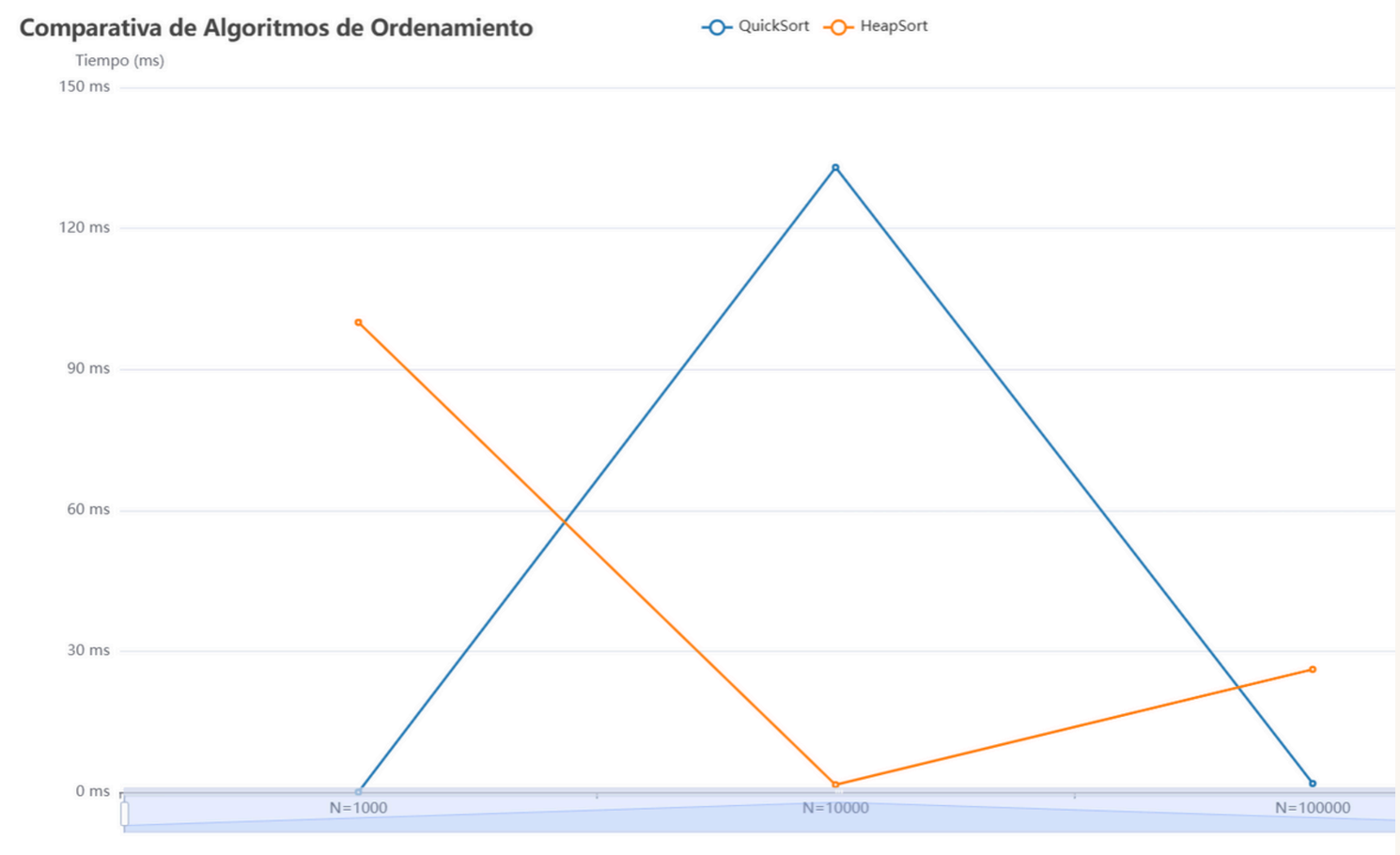
# Aleatorio uniforme



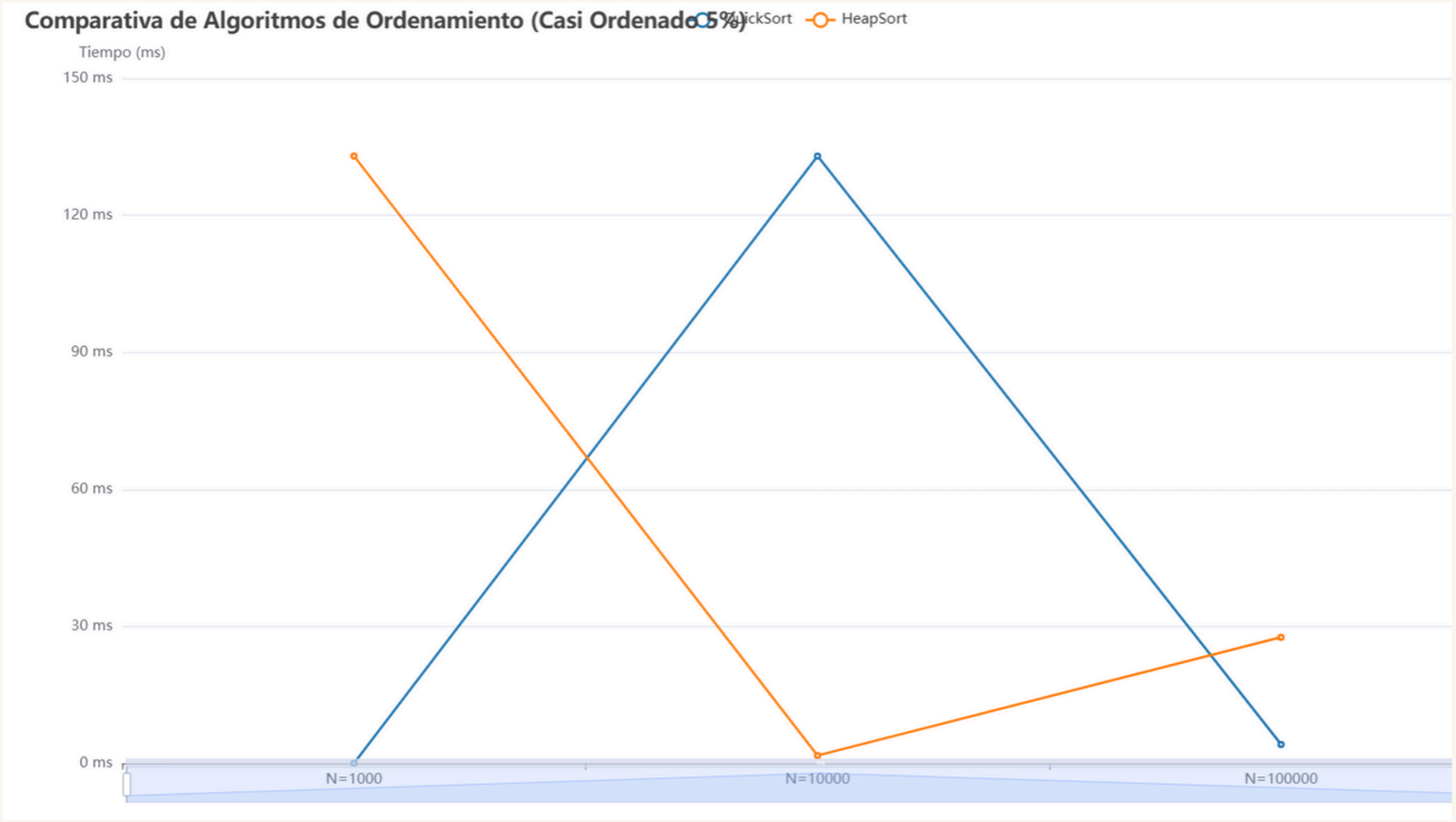
# Ordenado ascendente



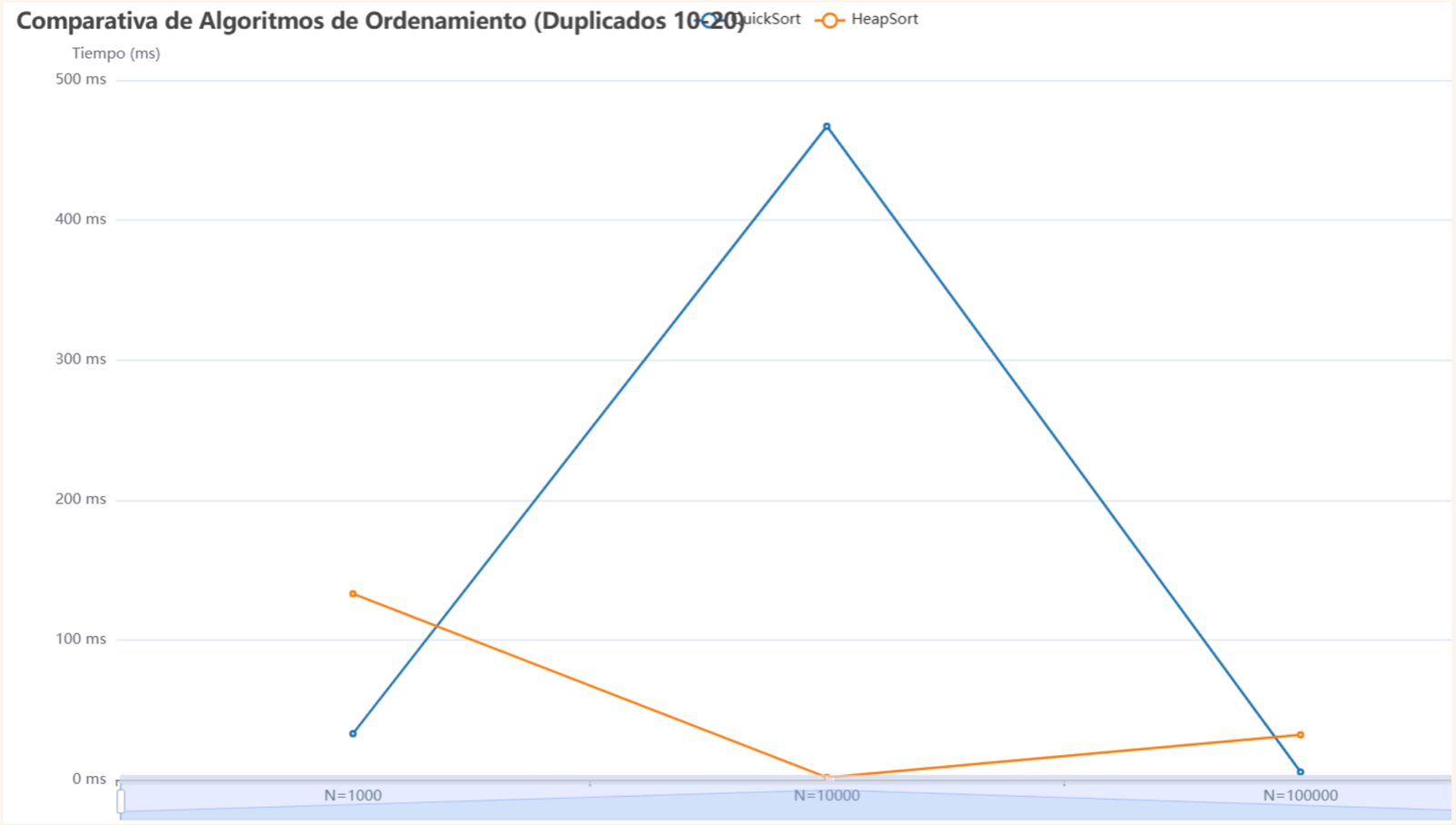
# Ordenado descendente



# Casi ordenado 5%



# Duplicados 10-20



# DISCUSIÓN Y CONCLUSIONES

- Los resultados obtenidos confirman el comportamiento teórico de ambos algoritmos:
- QuickSort es más rápido en casos promedio y con datos parcialmente ordenados.
- HeapSort mantiene rendimiento estable independientemente del patrón de entrada.
- Las diferencias se explican por la forma en que interactúan con la memoria y el tipo de datos:
- QuickSort aprovecha la localidad espacial,
- HeapSort realiza más accesos no contiguos, lo que incrementa el tiempo.
- Coincide con lo reportado por Sedgewick (1978) y LaMarca & Ladner (1999), reforzando la validez experimental del estudio.

- QuickSort: mayor velocidad promedio, ideal para conjuntos medianos o datos parcialmente ordenados.
- HeapSort: mayor estabilidad, recomendable en entornos donde la consistencia es más importante que la rapidez.
- La elección del algoritmo depende del contexto y del hardware disponible.
- La eficiencia no solo se define por la complejidad teórica, sino también por la interacción entre algoritmo, memoria y datos.

ESTA INVESTIGACIÓN NO SOLO CONTRIBUYE A LA COMPRENSIÓN DE LA EFICIENCIA DE QUICKSORT Y HEAPSORT, SINO QUE TAMBIÉN DESTACA LA COMPLEJIDAD DEL RENDIMIENTO ALGORÍTMICO EN EL CONTEXTO DEL HARDWARE Y LOS DATOS.

LOS RESULTADOS OBTENIDOS OFRECEN UN MARCO ANALÍTICO ÚTIL PARA FUTURAS INVESTIGACIONES Y DESARROLLOS EN EL CAMPO DE LA INFORMÁTICA Y EL PROCESAMIENTO DE DATOS.

# MUCHAS GRACIAS