**Índice**

Tabla de contenido

[Introducción 1](#_Toc211722627)

[Objetivos 2](#_Toc211722628)

[Problemáticas 2](#_Toc211722629)

[Kanade’s 2](#_Toc211722630)

[Ordenamiento 3](#_Toc211722631)

[ALGORITMOS 3](#_Toc211722632)

[Algoritmo de Kanade's: Subarreglo Máximo 3](#_Toc211722633)

[Algoritmos de Ordenamiento 7](#_Toc211722634)

[Comparaciones entre algoritmos 10](#_Toc211722635)

[Algoritmos Cuadráticos (O(n^2)) 10](#_Toc211722636)

[Algoritmos O(n \log n): 15](#_Toc211722637)

[Conclusiones 17](#_Toc211722638)

# Introducción

Los algoritmos son una disciplina esencial que busca crear soluciones computacionales eficientes intentando usar los menos recursos posibles e intentando que la complejidad de los algoritmos sea lineal O(n). En el presente trabajo se trata de algoritmos sumamente importantes y usados como el algoritmo del máximo subarray o conocido como Kanade’s en sus tres versiones donde cada una reduce su complejidad hasta tener una versión lineal los algoritmos.

También incluye métodos de ordenamiento como Burbuja,Selección,Insercion,Mezcla y Quicksort comparándolos entre ellos la eficiencia midiéndose principalmente por el **tiempo de ejecución** y el **uso de memoria**, formalizado mediante la **Notación de O**, que describe el comportamiento asintótico de un algoritmo a medida que el tamaño de los datos de entrada (n) crece en este caso hasta 5 mil.

Teniendo como objetivo el trabajo de determinar las mejores prácticas algorítmicas para optimizar la solución de estos problemas o problemas que enfrentemos en el futuro como ingenieros.

# Objetivos

El propósito de este reporte es analizar y comparar la eficiencia, complejidad y aplicabilidad de algoritmos fundamentales en el manejo de datos, específicamente:

Analizar y contrastar las tres versiones del algoritmo de Kanade's (con 3, 2, y 1 bucle 'for') para la solución del problema del subarreglo máximo, evaluando su complejidad temporal mediante la notación O().Con lo que determinamos la mejor versión del algortmo.

Identificar y comparar la complejidad en el peor caso de los algoritmos de ordenamiento básicos como Burbuja, Selección, Inserción y algunos más avanzados como Mezcla, Quicksort.

# Problemáticas

## Kanade’s

La problemática central abordada es la ineficiencia de los algoritmos para el uso de menos recursos y reducción de costos computacional que resulta de un diseño algorítmico deficiente o de la mala implementación de estos según el caso o la cantidad de datos que tenemos de entrada. Específicamente, se busca:

Problema de Ineficiencia en Subarreglo Máximo: En esta versión tiene un enfoque de fuerza bruta para el problema del subarreglo máximo puede resultar en una complejidad O(n^3) o O(n^2), volviéndose impráctico para grandes conjuntos de datos y solo útil en cantidades pequeñas.

La versión más eficiente en este caso es la aplicación del Algoritmo de Kanade's O(n) demuestra la optimización de un problema típicamente cuadrático o cúbico a una solución lineal.

## Ordenamiento

El problema de Ineficiencia en Ordenamiento de datos es muy común por el tiempo y el costo computacional para algoritmos de ordenamiento con complejidad O(n^2) como Burbuja o Selección en grandes volúmenes de datos conduce a tiempos de espera inaceptables si fuera solo mil datos tendrían que ser en el peor de los casos un millón de iteraciones.

Por lo que la implementación y análisis de algoritmos de complejidad O(n \log n) como Mezcla o Quicksort debe de ser muy importante en el uso de grandes cantidades de datos ofreciendo un rendimiento superior, esencial en sistemas de bases de datos y análisis masivo de información.

Usando en ambos casos como herramienta el análisis de complejidad temporal mediante la Notación O() como la técnica principal de cómputo para evaluar la eficiencia del algoritmo según una entrada(n).

# ALGORITMOS

## Algoritmo de Kanade's: Subarreglo Máximo

En los seudocódigos vistos en la clase buscando más información se encontró que el problema se le conoce como Kanade´ en el cual nos pide la suma máxima de valores consecutivos en una sucesión de números enteros positivos como negativos se nos propuso realizar la suma de todos los posibles subarreglos lo cual se puede hacer, pero es muy ineficiente y difícil de implementar, después una versión aun del mismo nivel de complejidad, pero sin acomodar los datos hasta una versión lineal.

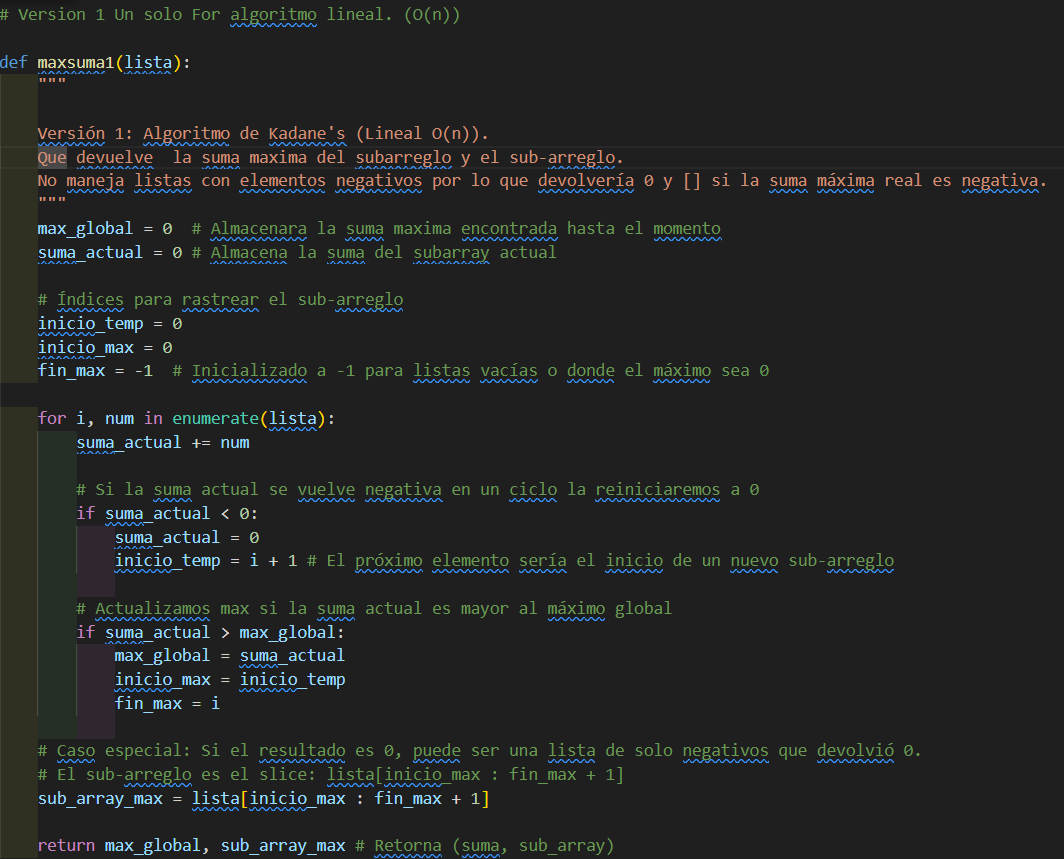
El algoritmo de Kanade's está diseñado para resolver el problema del subarreglo máximo que es encontrar un subarreglo contiguo dentro de un arreglo unidimensional de números (que pueden ser positivos o negativos) cuya suma sea la mayor.

En los seudocódigos proporcionados podemos notar que la complejidad de implementación, así como el costo de cómputo de cada uno cambia de O(n^3) hasta realizarse a O(n ) como se muestra en la tabla y después ya implementado en código:

|  |  |  |  |
| --- | --- | --- | --- |
| Versión | Bucles (for) | Complejidad O () | ¿Cuál es Mejor? |
| Fuerza Bruta 1 | 3 for | O(n^3) | Muy Ineficiente |
| Fuerza Bruta 2 | 2 for | O(n^2) | Ineficiente |
| Kanade's Óptimo | **1 for** | **O(n)** | **El Mejor** |

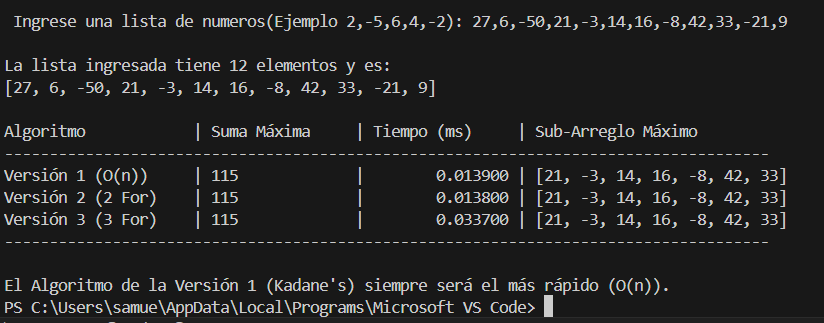
**Codigo:**

En la imagen solo mostramos la implementación del seudocódigo de la versión de un for pero con una pequeña modificación con el uso de variables como índices para no solo mostrar la suma máxima si no también el subarreglo.

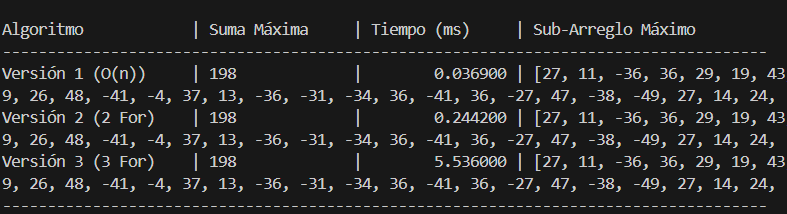
****

En el archivo Max\_Sub\_Array.py podemos encontrar la implementación de las 3 versiones, así como ejemplo de usando la lista 27,6,-50,21,-3,14,16,-8,42,33,-21,9 el resultado de la suma es 115 y el subarreglo máximo es 21,-3,14,16,-8,42,33.

En el segundo archivo Menu\_ Max\_Sub\_Array.py podemos llamar los 3 métodos al mismo tiempo con una lista que podemos introducir y este nos mostrara el tiempo que tarda cada método en mostrar el subarray con lo que podemos demostrar cual versión de las 3 es mejor según su tiempo de ejecución. Usando la misma lista del problema notamos que la diferencia es pequeña en los tiempos.



Podemos observar que la versión 1con un solo for y la versión 2 sin ordenarlos datos, con poca cantidad de datos (12 en este caso) ambas funciones funcionan igual o mejor la versión 2 para tan pocos datos, por lo que ingresamos una lista con mayor cantidad de datos podemos ver una gran diferencia.



En el ejemplo anterior se uso una lista de 100 elementos del 50 al -50 con lo que la version uno fue la mas eficiente .0369 segundos la version 2(0.244 segundos) casi diez veces más lento que la version 1 y la version 3 (5.53 segundos) cerca de 180 mas lento que la version 1.

Concluyendo que siempre es mejor buscar la forma lineal de un algoritmo si es posible o si conocemos su forma lineal de implementación con lo que reducimos el tiempo y recursos ya que una computadora más potente no significa que sea más eficiente nuestro codigo.

**Uso del algoritmo**

Se encontró que el uso más común del algoritmo de Kanade's se encuentra en el análisis de series de tiempo financieras y la Optimización de Transacciones Bursátiles para la automatización de compra o venta de acciones o divisas.

Donde una persona o institución tiene una lista de precios de una acción durante n días. El objetivo no es simplemente encontrar la compra y venta más barata y cara, sino encontrar el periodo contiguo de compra y venta que maximice la ganancia neta.

En lugar de usar los precios de las acciones (P), se crea una nueva lista (D) que registra la diferencia (cambio) de precio entre el día actual y el día anterior. Un valor positivo en D representa una ganancia diaria, y uno negativo, una pérdida.

D[i] = P[i] - P[i-1]

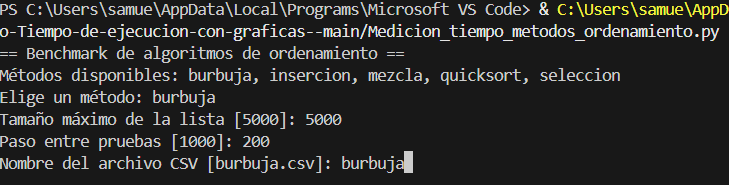
Al aplicar el algoritmo de Kanade's a la lista de diferencias (D), el resultado (la suma máxima del subarreglo) indica el beneficio neto más alto que se pudo obtener de forma contigua.

## Algoritmos de Ordenamiento

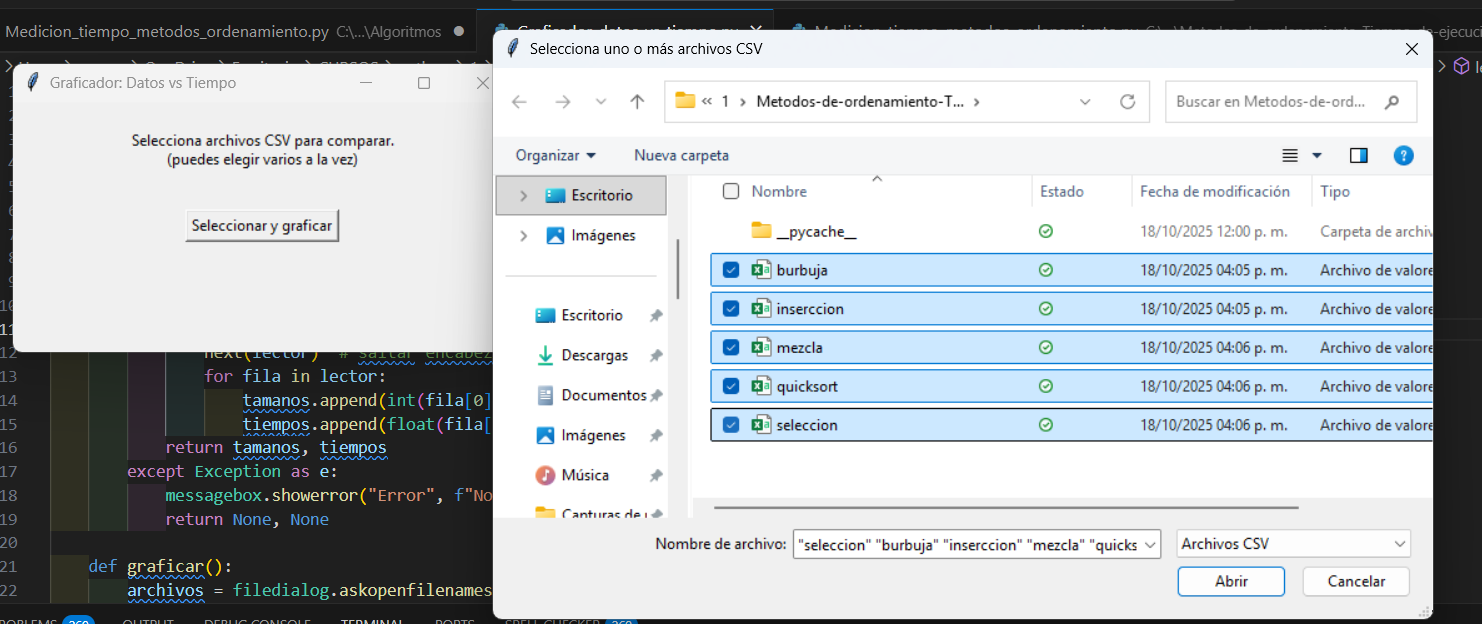
El problema de **Ordenamiento** (Sorting) es posiblemente de los más comunes en análisis de datos, la eficiencia de un algoritmo de ordenamiento es crucial, ya que el ordenamiento es la base para búsquedas rápidas, procesamiento de bases de datos y análisis de información. En este trabajo se implementamos 5 algoritmos para organizar los elementos de una lista o arreglo en un orden específico (numérico o alfabético).

Después de implementar los algoritmos los compararemos cada uno contra los demás algoritmos entre estos hay dos categorías principales según su complejidad en el peor caso como cuadrática O(n^2) como Burbuja y logarítmica lineal O(n \log n) como Quicksort.

En el archivo metodos\_ordenamiento.py están los métodos de ordenamiento el programa pide al usuario que método quiere utilizar después el tamaño quiere la lista tiene como límite máximo de 5mil elemento que se puede modificar, luego los pasos en que se cuenta el tiempo y por último en nombre del CSV.



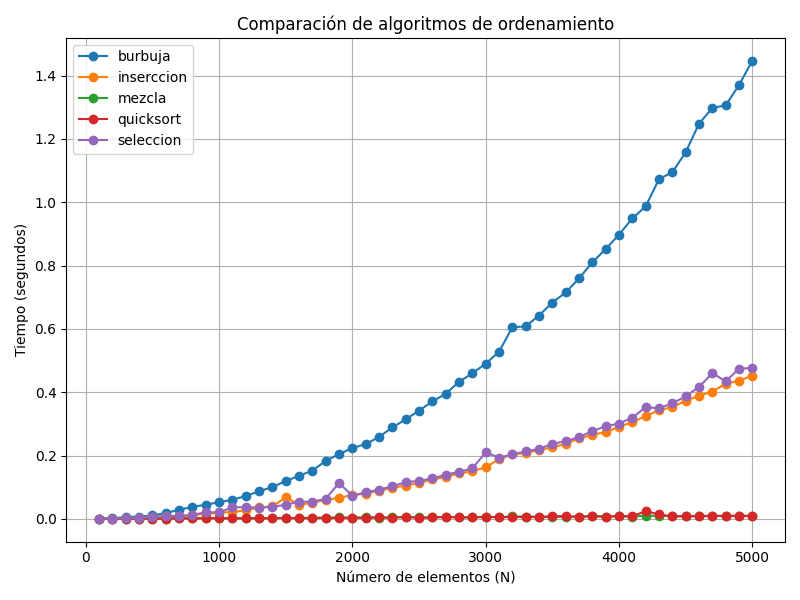
Para la prueba de su funcionamiento en Medicion\_tiempo\_metodos\_ordenamiento.py esta contiene las funciones para generar un CSV donde almacena el tiempo y datos que el usuario introdujo y en Graficador\_datos\_vs\_tiempo.py abre una ventana donde el usuario debe de seleccionar los CSV que quiere comparar.



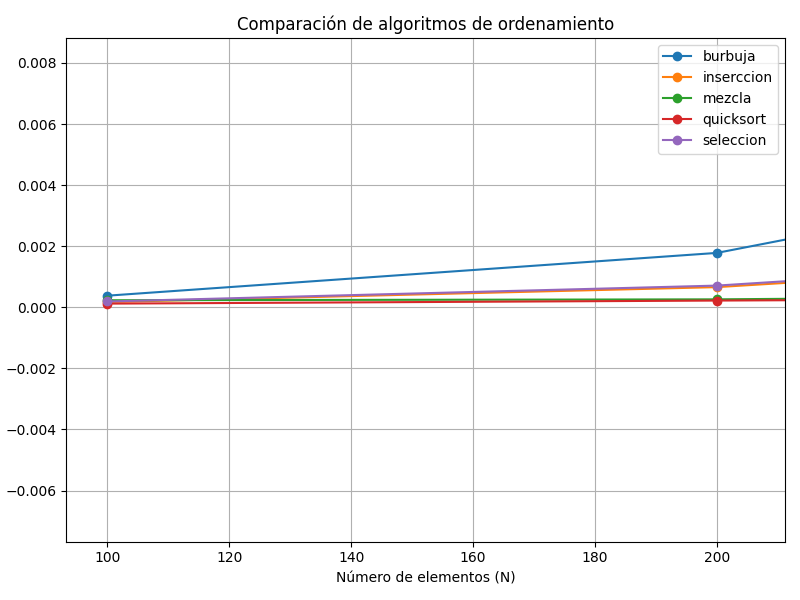
Después de esas pruebas determinamos que el método más rápido es Quicksort como se muestra en la tabla y en las futuras imágenes que demuestra como entre mas datos se ingresan el tiempo se dispara exponencialmente en Burbuja e Inserción O(n^2) y las más cercanas a lo lineal son mezcla y Quicksort O(n \log n).

**Complejidad Asintótica de los Algoritmos de Ordenamiento**

|  |  |  |  |
| --- | --- | --- | --- |
| **Versión** | **Tipo de Algoritmo** | **Complejidad $O()$ (Peor Caso)** | **¿Cuál es Mejor?** |
| Burbuja (Bubble Sort) | Comparación Simple | O(n^2) | Muy Lento |
| Selección (Selection Sort) | Comparación Simple | O(n^2) | Lento |
| Inserción (Insertion Sort) | Comparación Simple | O(n^2) | Eficiente para datos casi ordenados |
| Mezcla (Merge Sort) | Divide y Vencerás | O(n \log n) | El Mejor en el Peor Caso |
| **Quicksort (Rápido)** | **Divide y Vencerás** | **O(n^2) (Peor caso), O(n \log n) (Promedio)** | **El Más Rápido en Promedio** |

****

Observamos en la gráfica que Merge Sort y Quicksort mantienen un crecimiento mucho más moderado y son significativamente más rápidos garantiza la complejidad O(n \log n) incluso en el peor caso, el **Quicksort** suele ser marginalmente más rápido en el **caso promedio** debido a una menor sobrecarga de operaciones internas.



En la imagen se observa que a partir del dato 120 se nota la diferencia de los métodos y cuando debemos de usarlos si son miles de datos mezcla y Quicksort, muy pocos datos aun que es rara vez usaremos menos de 100 y por fácil implementación burbuja e inserccion.

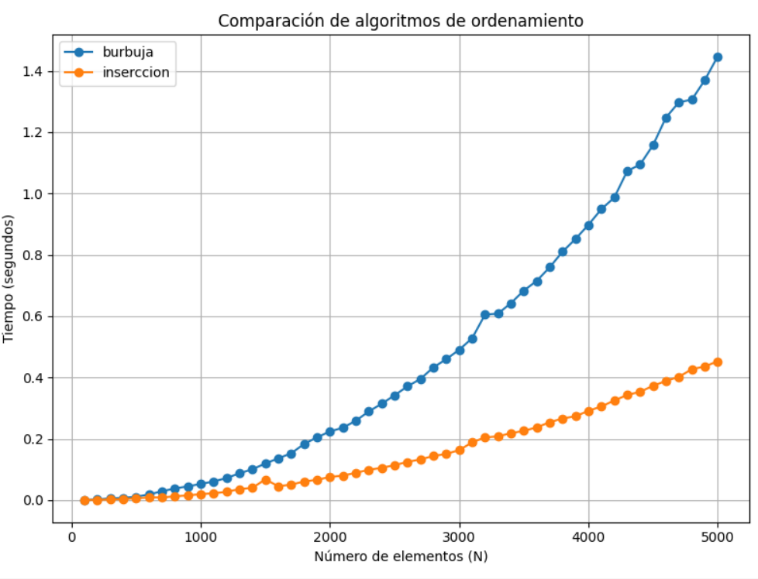
En conclusión, para cualquier aplicación real con grandes volúmenes de datos, la elección debe recaer en un algoritmo O(n \log n).

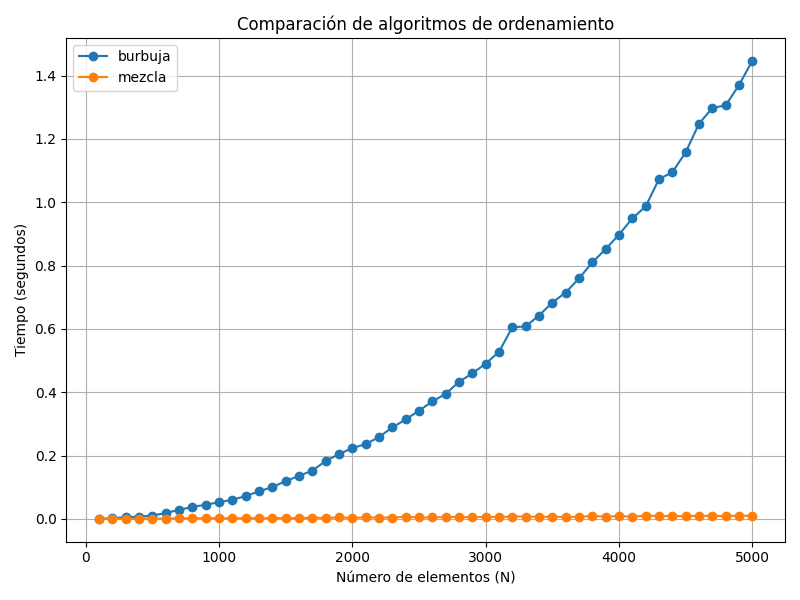
# Comparaciones entre algoritmos

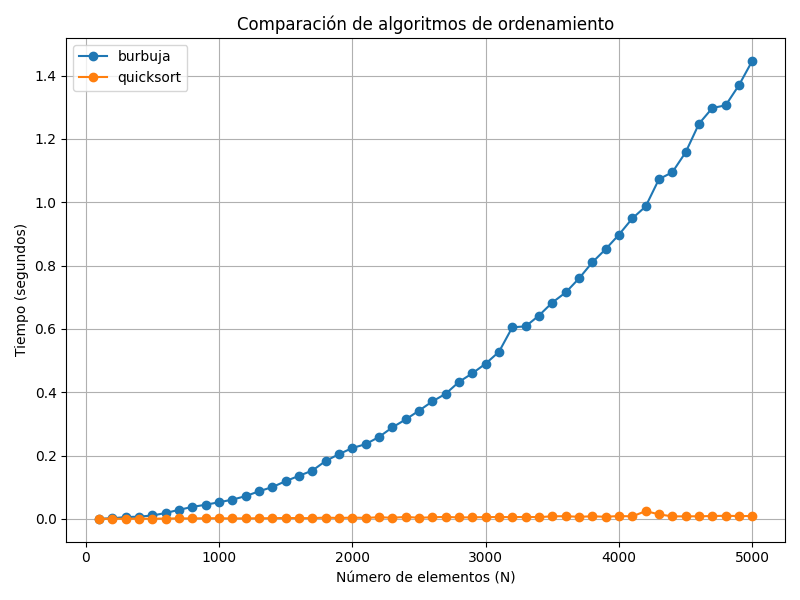
Algoritmos Cuadráticos (O(n^2))

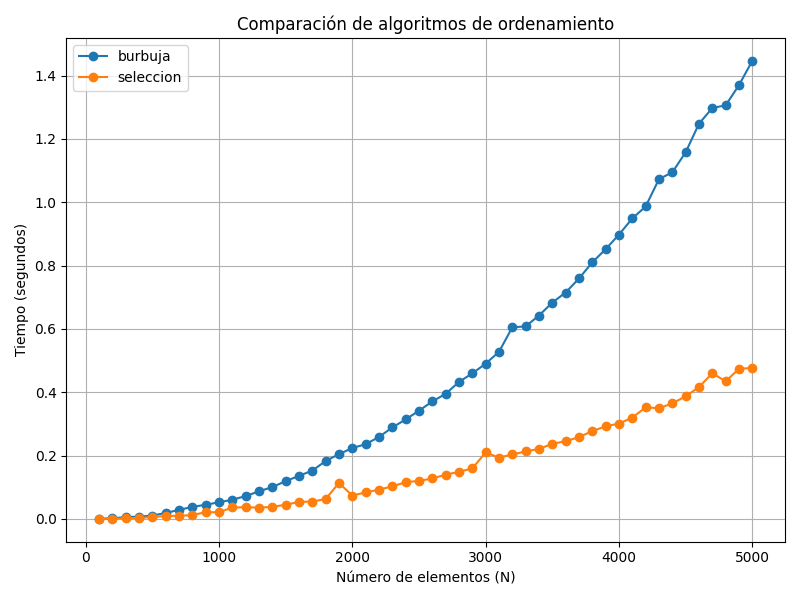
Burbuja, Selección, e Inserción demuestran un aumento drástico en el tiempo de ejecución al duplicar el tamaño (n) de la lista. Estos métodos se vuelven imprácticos más allá de unos pocos miles de elementos.

**BURBUJA:**





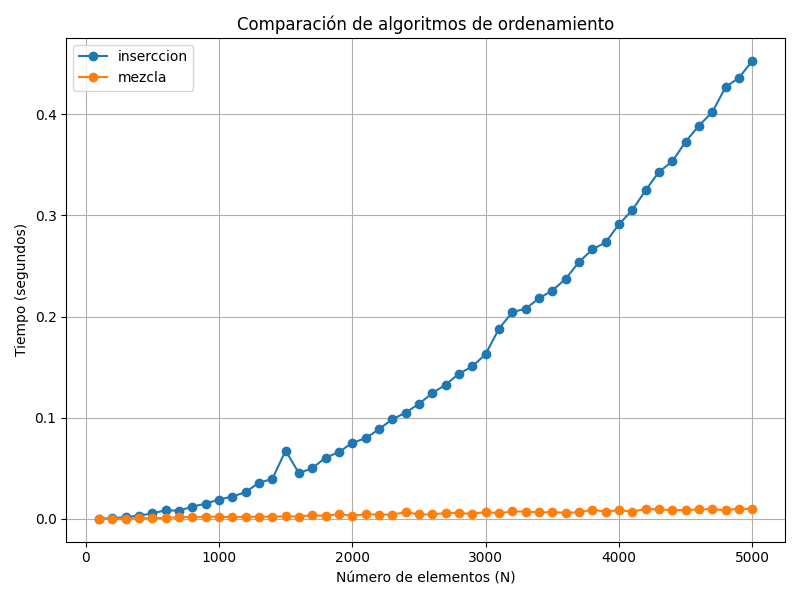


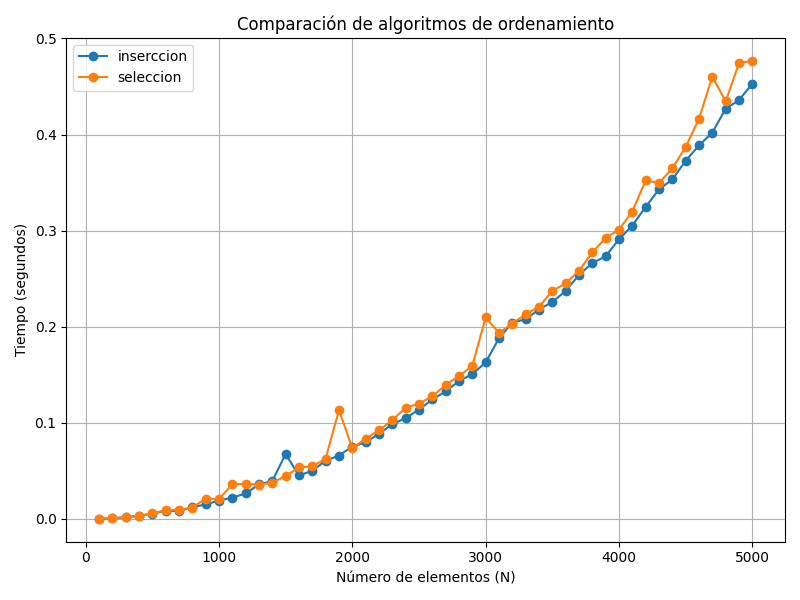


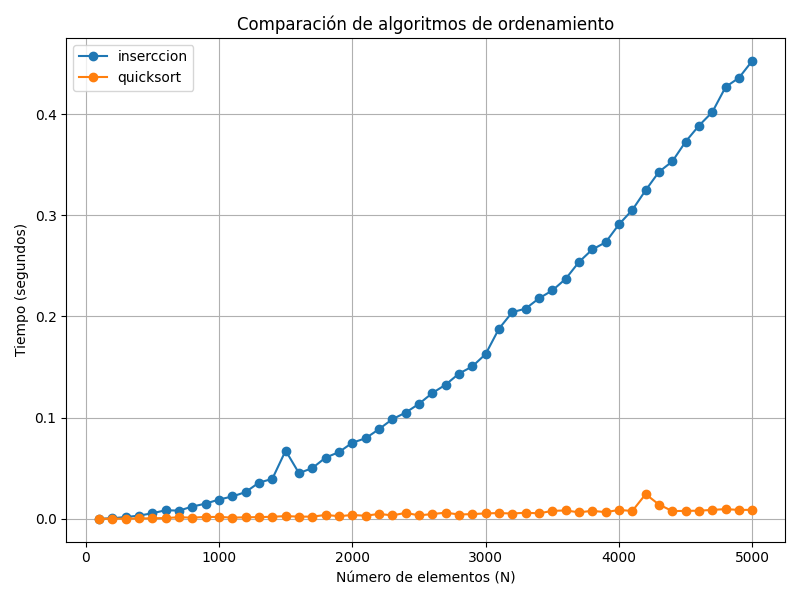
Podemos observar que burbuja es solo muy optima con pocos datos siendo igual de rapido que los otros cuando son menos de 100 datos pero al pasar los 100 datos se nota la diferencia.

**INSERCCION:**

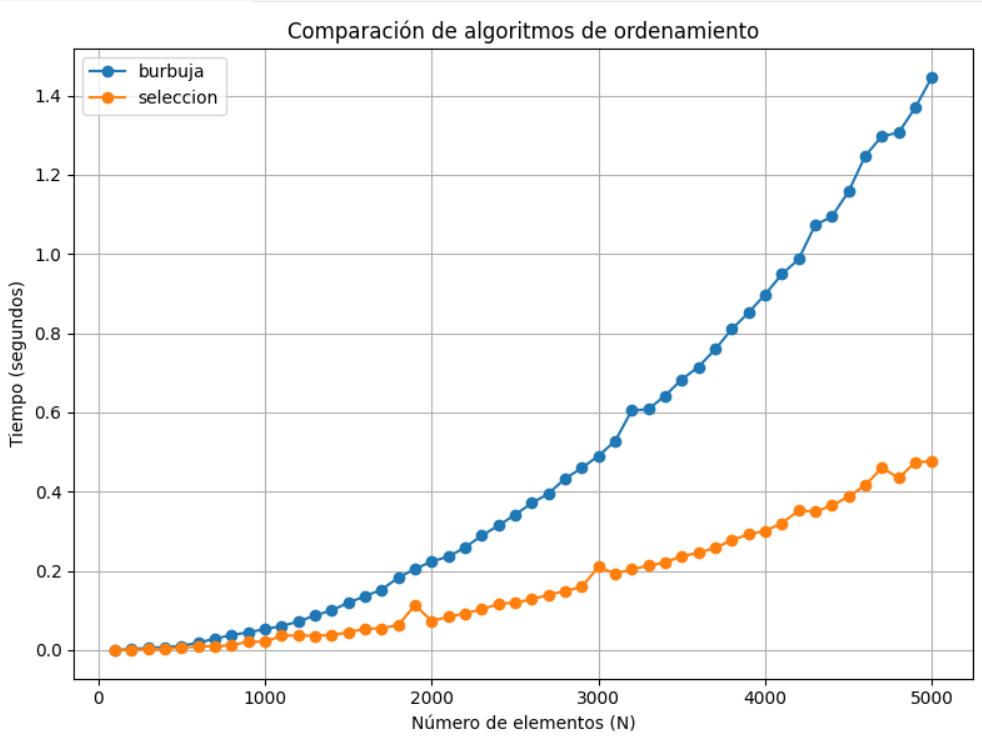
**Inserción** O(n^2) es común como subrutina en algoritmos híbridos (como Timsort) o en listas que ya están casi ordenadas, podemos notar que inserción es similar a burbuja por lo cual no es optimo con muchos datos.







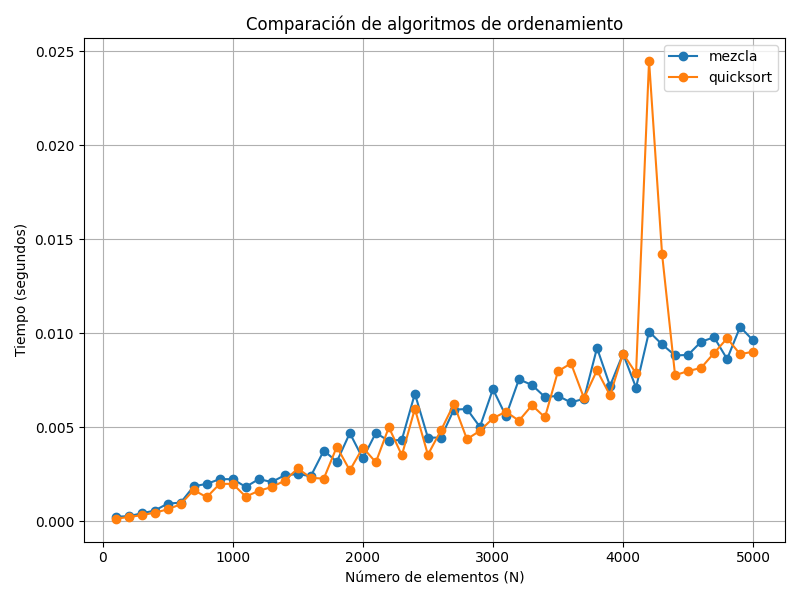
**Selección:**

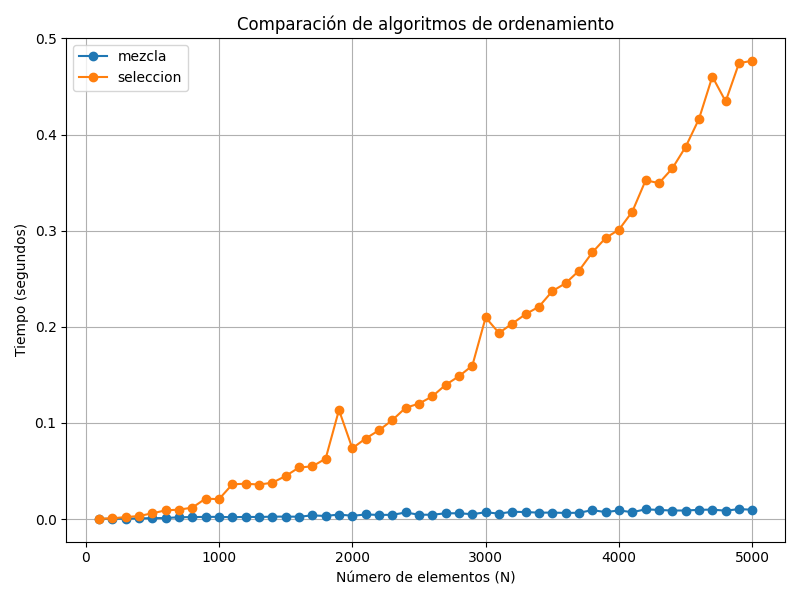
****

## Algoritmos O(n \log n):

**MEZCLA:**

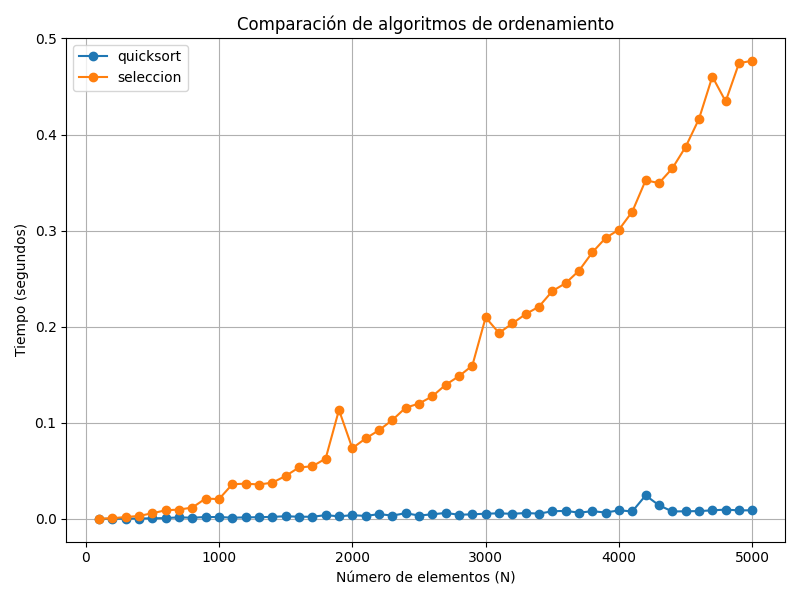
Es fundamental en el **Ordenamiento Externo** (cuando los datos no caben en la memoria principal, como en bases de datos masivas) y en cualquier sistema donde la **estabilidad** (mantener el orden relativo de elementos iguales) o un tiempo de ejecución garantizado sean vitales. Por lo que comparamos los dos más rápidos que son muy similares según las graficas en donde hay momentos que aún es más rápido que Quicksort en ciertos intervalos.





**Quicksort:**

Es el método de ordenamiento **por defecto** en muchas librerías y lenguajes (C, Java, Python) para ordenamientos internos, debido a su excelente rendimiento en promedio y su eficiencia en memoria. Por ultimo se compara el más rápido con un algoritmo cuadrático para ver la gran diferencia de la eficiencia de este algoritmo.



# Conclusiones

Como ingeniero debemos de tener la capacidad para la elección del algoritmo determina la escalabilidad usando herramientas de análisis de complejidad como el uso de O() demostró que la diferencia entre un algoritmo O(n^2) (como la versión de 2 for de Kanade's o el Bubble Sort) y uno O(n \log n) o O(n) es crítica e importante según la cantidad de datos que manejamos por lo que debemos de tener el suficiente criterio para saber que algoritmo usar dependiendo del contexto. Para el problema del subarreglo máximo, la optimización a O(n) de Kanade's con 1 for es la única solución viable para conjuntos de datos masivos.

Pero también hay que mencionar que no existe un único "mejor" algoritmo de ordenamiento para todas las situaciones. Mientras que Quicksort ofrece la mejor velocidad promedio, el Merge Sort garantiza la complejidad O(n \log n) incluso en el peor escenario, haciéndolo preferible en aplicaciones críticas que no pueden tolerar picos de rendimiento (como ocurre con el peor caso O(n^2) de Quicksort). Los algoritmos O(n^2) solo son justificables en casos educativos o en listas extremadamente pequeñas operaciones que sean una sola vez cada cierto tiempo y con pocos datos.

Podemos ver la importancia de la optimización de la evolución del algoritmo de Kanade's ilustra un principio fundamental en el diseño de algoritmos: la simplificación y el análisis de la dependencia de los bucles anidados pueden transformar una solución ineficiente (O(n^3)) en una solución lineal óptima (O(n)). El diseño eficiente, no la fuerza bruta, es la base de la programación moderna.

Por último la más importante es saber elegir el algoritmo correcto que radica en la escalabilidad y el propósito del sistema así como la cantidad y tipos de datos que manejaremos.