Algoritmos de ordenamiento

Este proyecto trata sobre los diferentes algoritmos de ordenamientos que hay para la programación los cuales son: Bubble Sort, Insertion Sort, Selection Sort, Merge Sort, Quick Sort, y Heap Sort. Nosotros vamos a escoger 4 de estos y vamos a explicar cómo funcionan y cuales son más eficientes a la hora de ordenar. Los algoritmos que escogimos son Bubble Sort, Insertion Sort, Selection Sort, Merge Sort.

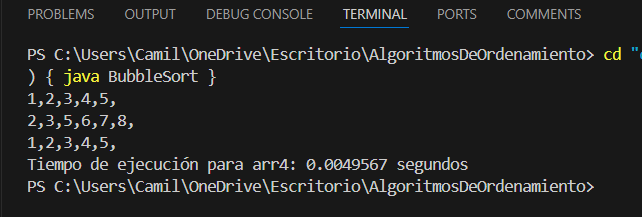
Bubble Sort

Este es uno de los algoritmos más sencillos y fácil de implementar que hay de ordenamiento, pero ser el más sencillo tiene una desventaja y es que además es uno de los menos eficientes que hay al ser O(n). Por ende, escogimos este, para mostrar su sencillez a la hora de programarlo, pero su increíble demora en los tiempos de ordenamiento.

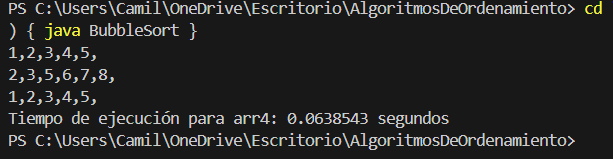
El funcionamiento de Bubble Sort se basa en dos ciclos principales. El primero de estos ciclos es el ciclo **i**, que recorre todo el arreglo, mientras que el segundo ciclo, denominado **j**, es el que compara los elementos adyacentes del arreglo y realiza los intercambios cuando es necesario para organizar los números. En cada iteración del ciclo **j**, el algoritmo compara el elemento en la posición j con el siguiente elemento en la posición j+1.

Si el elemento en la posición j es mayor que el de la posición j+1, entonces se realiza un intercambio entre estos dos elementos, colocando al número mayor en la posición siguiente. Este proceso se repite de manera continua a lo largo del arreglo, y en cada pasada del ciclo **i.**

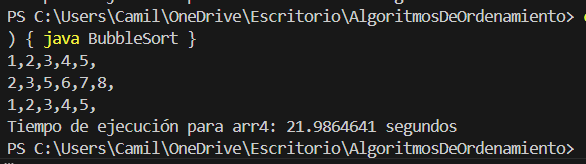
Pruebas de bubble sort

Prueba con 1000 elementos:

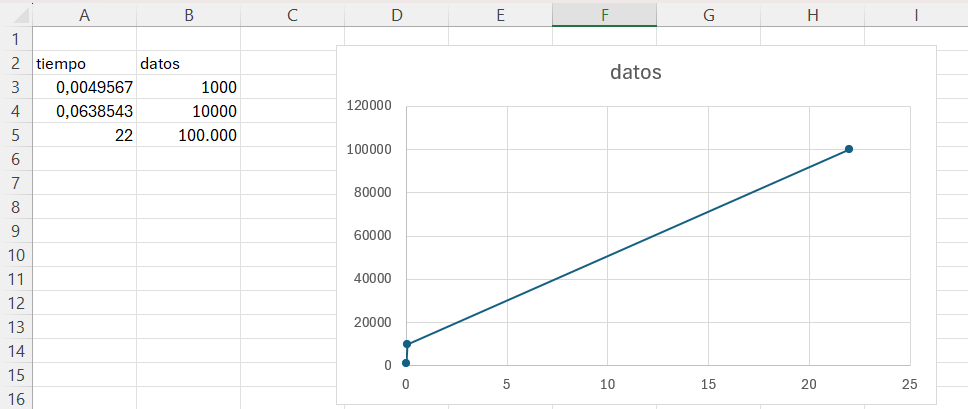
Prueba con 10000 elementos:



Prueba con 100000 elementos:



Grafica de Bubble sort



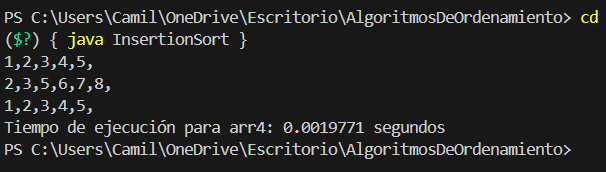
Insertion sort

Insertion Sort también es uno de los algoritmos de ordenamiento más sencillos de entender y fácil de implementar que hay. Aunque a pesar de su simplicidad, presenta también una desventaja en la eficiencia, ya que su complejidad es O(n²), pero aun así siendo muchísimo más eficiente que el algoritmo bubble sort. Por lo que es muy intuitivo y rápido para listas pequeñas o casi ordenadas, su rendimiento se ve gravemente afectado cuando se trabaja con listas grandes o desordenadas. Este lo escogimos también por su sencilles y además por su memoria eficiente.

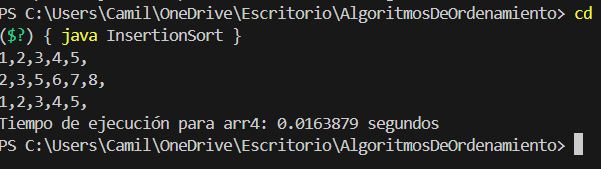
El funcionamiento de Insertion Sort se basa en un solo ciclo que recorre todo el arreglo, comenzando con el segundo elemento suponiendo que el primero ya está ordenado. En cada iteración, se toma el elemento en la posición actual y se compara con los elementos previos en la lista (ya ordenada). Si el elemento actual es más pequeño que el de la posición anterior, se realiza un desplazamiento de los elementos más grandes hacia la derecha para hacer espacio. Este proceso se repite hasta que se encuentra la posición correcta para el elemento actual, insertándolo en la sublista ordenada.

Pruebas de Insertion sort

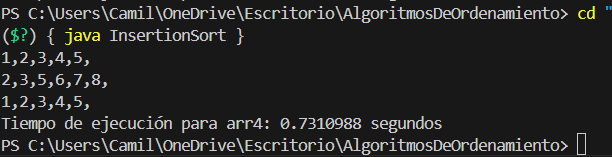
Prueba con 1000 elementos



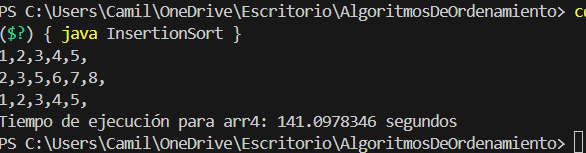
Prueba con 10000 elementos



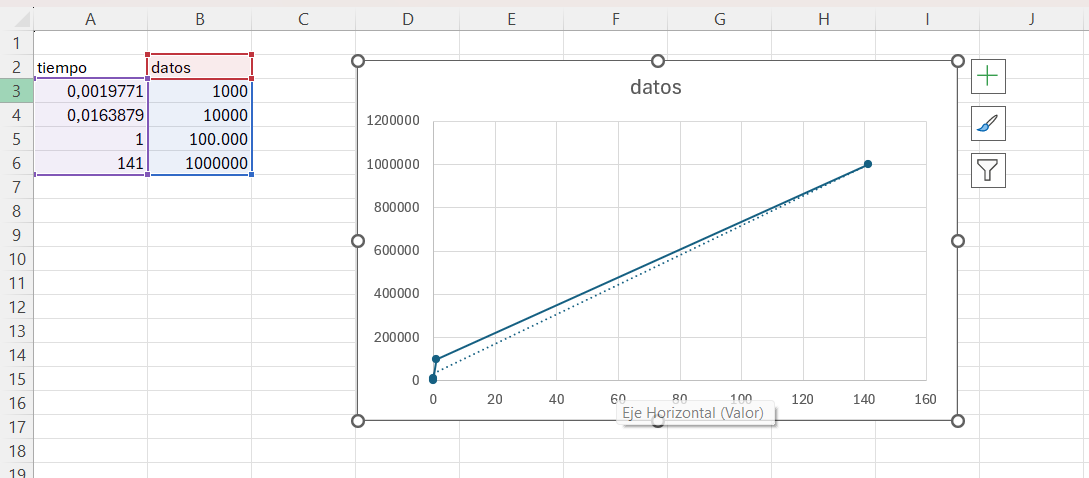
Prueba con 100000 elementos



Prueba con 1000000 elementos



Grafica de insertion sort



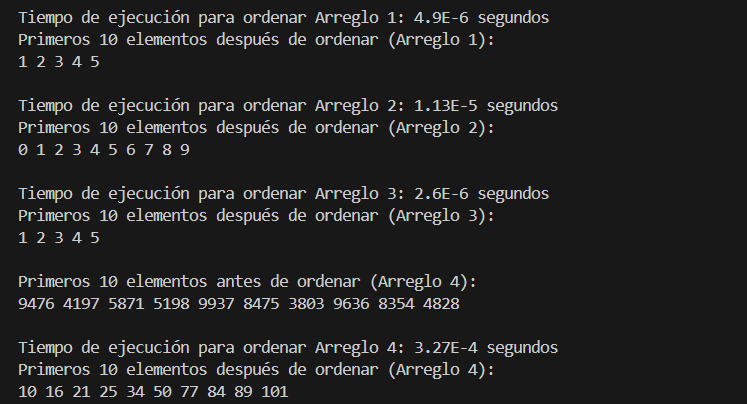
Merge sort

Merge sort es un algoritmo de ordenamiento basado en la técnica divide y vencerás, lo que lo convierte en uno de los algoritmos más eficientes y populares para ordenar grandes volúmenes de datos, este posee una complejidad de O(n log n), lo que lo hace más eficiente en listas grandes y desordenadas

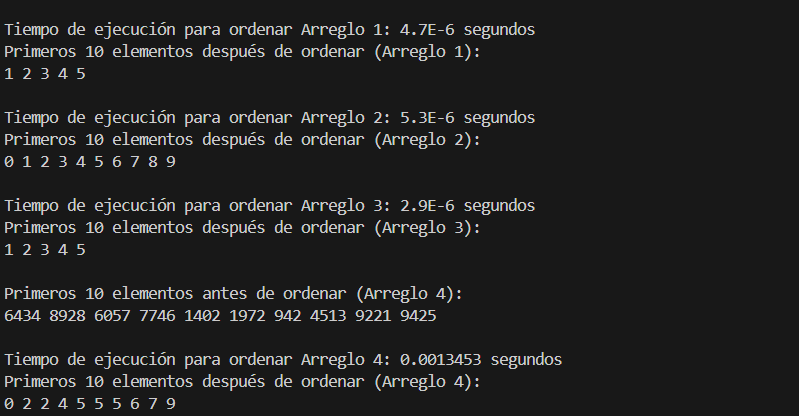
Este algoritmo funciona dividiendo recursivamente el arreglo original en dos mitades más pequeñas, hasta que cada subarreglo contiene un solo elemento, posteriormente estos subarreglos se combinan de manera ordenada para formar el arreglo final

El primer paso consiste en dividir el arreglo en dos mitades aproximadamente iguales, esto se repite recursivamente en cada mitad hasta que cada sub arreglo solo contenga un elemento, una vez todos los subarreglos solo tengan un elemento comienza la combinación. Durante este proceso se comparan los elementos de los subarreglos de manera ordenada y se combinan para formar un arreglo más grande y ordenado, este proceso garantiza que los elementos estén ordenados desde el más pequeño al más grande y esta fase continua hasta que los subarreglos se combinen de nuevo en un arreglo completamente ordenado.

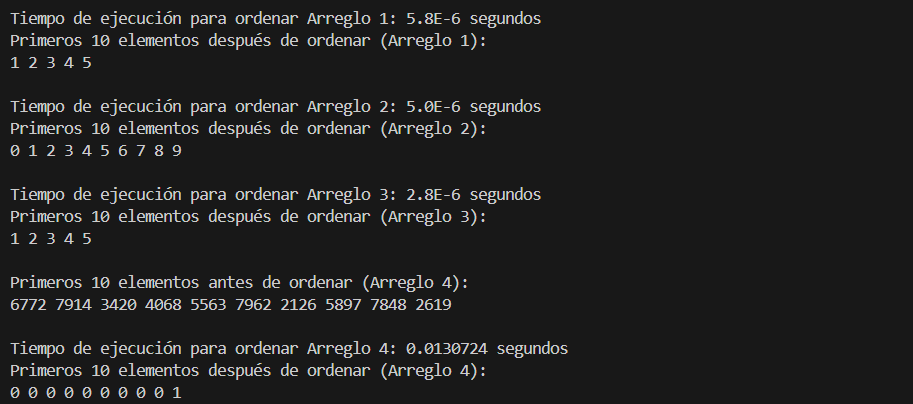
Prueba con 1000 elementos:



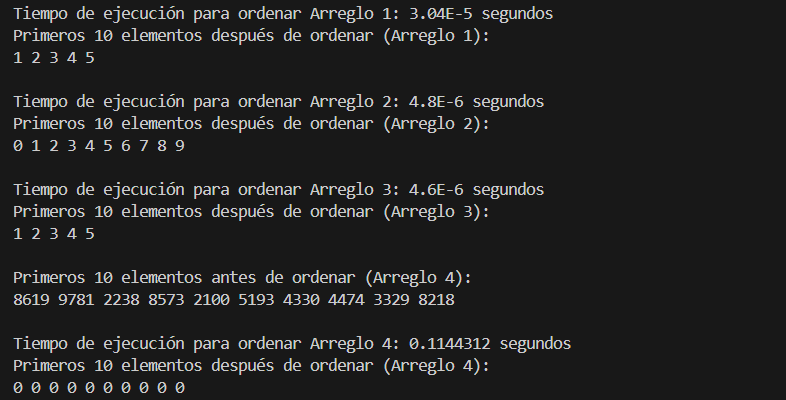
Prueba con 10000 elementos:



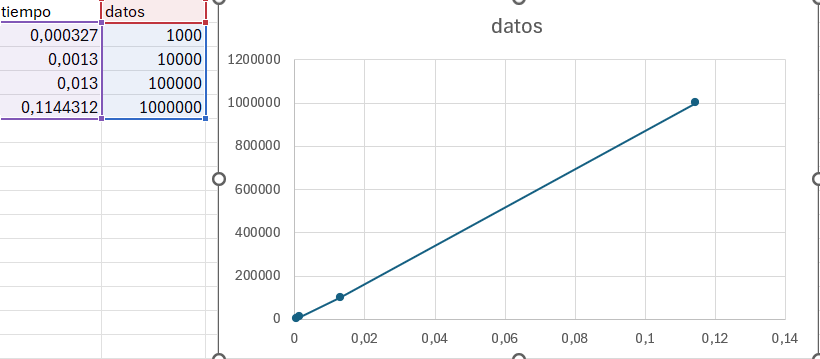
Prueba con 100000 elementos:



Prueba con 1000000 elementos:



Grafica de merge sort

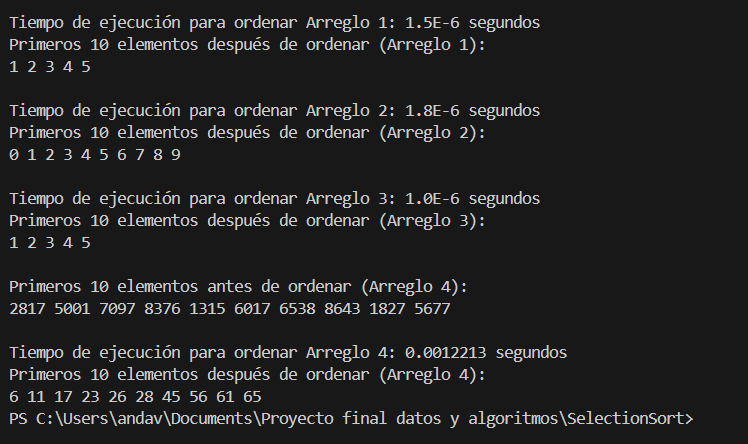


Selection sort

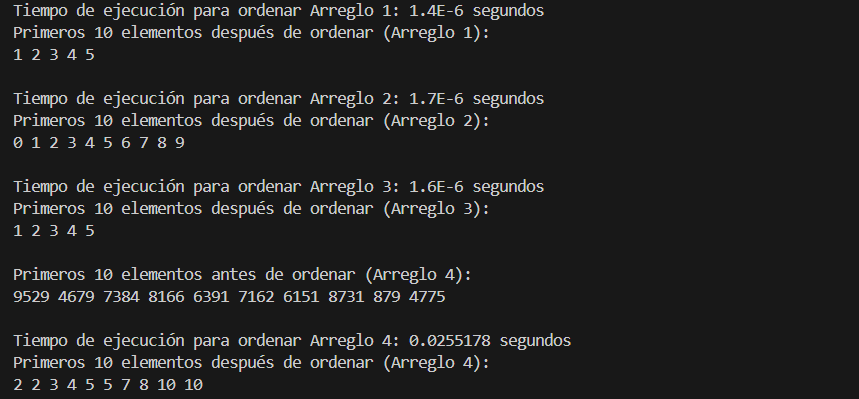
Selection Sort es un algoritmo simple y fácil de implementar que se basa en la selección repetida del valor mínimo y su colocación en la posición correcta. Sin embargo, su O(n²) de complejidad de tiempo lo hace ineficiente para listas grandes. Aunque es adecuado para listas pequeñas o casos en los que la simplicidad es una prioridad, para listas grandes se prefieren otros algoritmos más eficientes. Además de que cuenta con varias desventajas, como su inestabilidad ya que si hay elementos iguales en el arreglo su orden relativo podría cambiar tras la ordenación, lo que genera un problema si se necesita mantener la relación entre elementos con el mismo valor, o el hecho de que no es adaptativo ya que Selection Sort no tiene una mejora de rendimiento si el arreglo ya se encuentra parcialmente ordenada. Pero a pesar de estas limitaciones, su facilidad de implementación y eficiencia en espacio lo hacen una opción atractiva en escenarios donde el rendimiento no es una preocupación crítica.

El algoritmo comienza recorriendo el arreglo para encontrar el valor más pequeño en la parte no ordenada, este valor más pequeño se intercambia con el primer valor de la parte no ordenada del arreglo, este proceso se repite para el siguiente elemento no ordenado, seleccionando así el valor más pequeño y colocándolo en su posición correcta y así sucesivamente, a medida que este algoritmo avanza la parte ordenada del arreglo crece mientras que la parte no ordenada se va reduciendo. La clave de este algoritmo es el intercambio entre el elemento mínimo y el primero de la parte no ordenada ya que así es como la parte ordenada en cada iteración crece un elemento más, y ya al final el algoritmo asegura que todos los elementos están en su posición correcta y el arreglo está completamente ordenado

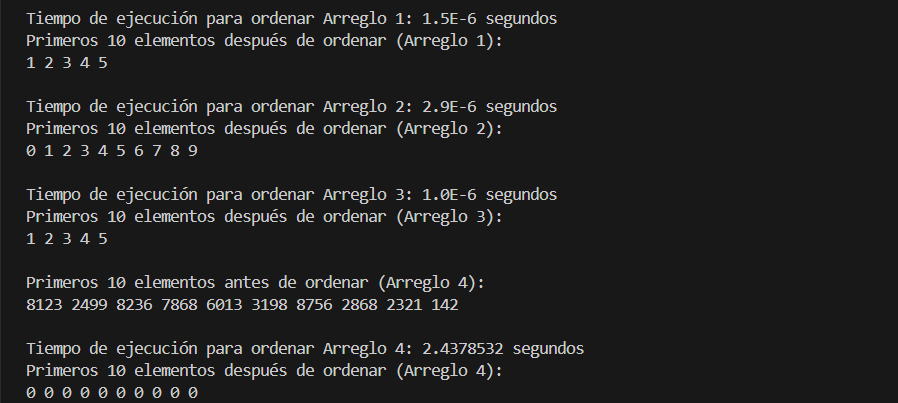
Prueba con 1000 elementos:



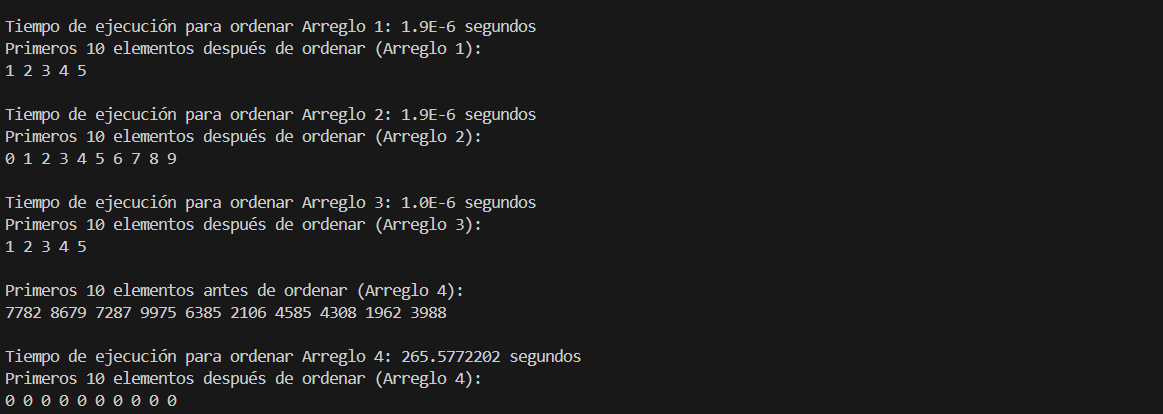
Prueba con 10000 elementos:



Prueba con 100000 elementos:



Prueba con 1000000 elementos:

(Sin contar que hubo mayor consumo de memoria RAM de casi hasta el 55% del pc)

Grafica de selection sort

