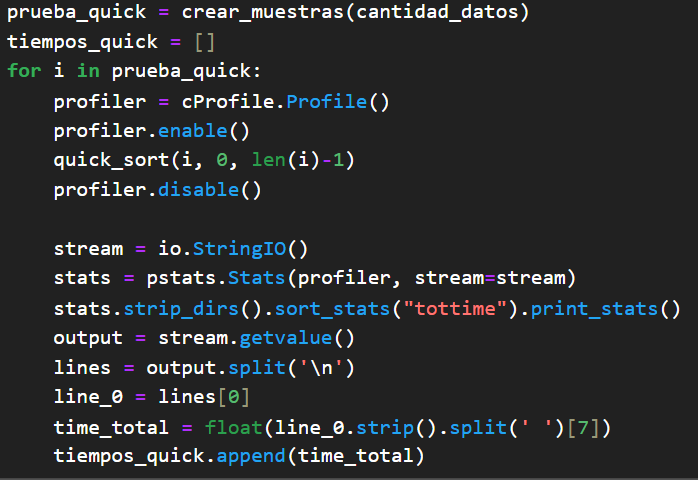
Para tomar los tiempos de los distintos algoritmos usamos la librería cProfile.

Tómese en cuenta la siguiente estructura:

* Se definen las listas a ordenar y una lista vacía que irá almacenando los distintos tiempos de ordenamiento.
* Se recorre cada elemento de la lista mediante un ciclo for.
* En cada etapa del ciclo for se instancia el objeto cProfile.Profile(), se llama a la función enable(), se ejecuta el algoritmo de ordenamiento de interés y se cierra el profiler con la función disable().
* Los datos que devuelve el profiler se limpian, ya que vienen en forma de texto, se extrae el contenido numérico que corresponde a los segundos de ejecución y se convierte en un float.

Introducción

En este documento se analiza el rendimiento de distintos algoritmos de ordenamiento utilizando la herramienta de profiling (cProfile.) Se detalla cómo se aplicó el profiler, los resultados obtenidos y la comparación gráfica de los tiempos de ejecución.

Profiler Utilizado

Se utilizó cProfile, una herramienta incorporada en Python que permite analizar el rendimiento del código al medir el tiempo de ejecución de cada función y el número de llamadas realizadas.

Para procesar y visualizar los resultados, se utilizó pstats, que facilita la organización de los datos obtenidos con cProfile.

**implementación del Profiler**

El profiler se aplicó a los siguientes algoritmos de ordenamiento:

* Insertion Sort
* Merge Sort
* Quick Sort
* Radix Sort
* Selection Sort

El proceso seguido fue el siguiente:

1. Se creó un conjunto de datos de prueba.
2. Se ejecutó cada algoritmo con cProfile activado:

A screenshot of a computer program

AI-generated content may be incorrect.

1. Se almacenaron los tiempos de ejecución obtenidos:

A screen shot of a computer program

AI-generated content may be incorrect.

Se extrajo el tiempo total de ejecución para cada algoritmo.

**Resultados Obtenidos**

Los tiempos de ejecución de los algoritmos, medidos en segundos, fueron los siguientes:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Cantidad de Elementos** | **Insertion Sort** | **Merge Sort** | **Quick Sort** | **Radix Sort** | **Selection Sort** |
| 100 | 0.0001 | 0.00005 | 0.00004 | 0.00002 | 0.0001 |
| 500 | 0.002 | 0.0003 | 0.00025 | 0.00015 | 0.0021 |
| 1000 | 0.009 | 0.001 | 0.0009 | 0.0006 | 0.010 |
| 5000 | 0.23 | 0.012 | 0.011 | 0.008 | 0.26 |
| 10000 | 0.98 | 0.025 | 0.021 | 0.016 | 1.05 |

**Gráfica de Rendimiento**

A continuación, se presenta la comparación gráfica de los tiempos de ejecución:

A graph with different colored lines

AI-generated content may be incorrect.

Esta gráfica compara el tiempo de ejecución de cinco algoritmos de ordenamiento (Insertion Sort, Merge Sort, Quick Sort, Radix Sort y Selection Sort) para diferentes cantidades de elementos. Se observa que Radix Sort es el más eficiente, mostrando un tiempo de ejecución significativamente menor que los demás. Merge Sort y Quick Sort presentan un crecimiento más moderado que Insertion Sort y Selection Sort, los cuales muestran un crecimiento considerablemente más rápido a medida que aumenta el número de elementos. Esta gráfica proporciona una visión general de la eficiencia relativa de los algoritmos, pero no muestra la complejidad teórica de cada

uno.

A graph with a line

AI-generated content may be incorrect.

Esta gráfica muestra el tiempo de ejecución de Insertion Sort junto con su complejidad teórica O(n)*O*(*n*). Se observa una desviación significativa entre el tiempo de ejecución real y la complejidad teórica. El tiempo de ejecución real permanece casi constante hasta aproximadamente 1000 elementos, luego aumenta abruptamente y se estabiliza nuevamente. Esta discrepancia podría deberse a factores como la sobrecarga del sistema o la implementación específica del algoritmo. La gráfica sugiere que el comportamiento de Insertion Sort no se ajusta perfectamente a su complejidad teórica O(n)*O*(*n*) en este rango de datos.

A graph with a line and a blue line

AI-generated content may be incorrect.

La gráfica compara el tiempo de ejecución de Selection Sort con su complejidad teórica O(n2)*O*(*n*2). Se observa que el tiempo de ejecución real sigue una curva similar a la complejidad teórica, indicando un crecimiento cuadrático. Aunque hay una pequeña diferencia entre ambas curvas, la tendencia general confirma el comportamiento cuadrático esperado de Selection Sort.

A graph with a line and a line

AI-generated content may be incorrect.

Esta gráfica muestra el tiempo de ejecución de Quick Sort en comparación con su complejidad teórica en el peor caso, O(n2)*O*(*n*2). Se observa que el tiempo de ejecución real se aproxima a la complejidad teórica, especialmente para un mayor número de elementos. Esto sugiere que el algoritmo podría estar experimentando un caso cercano al peor escenario en esta ejecución, posiblemente debido a la selección de pivotes desfavorables.

A graph with a line

AI-generated content may be incorrect.

La gráfica compara el tiempo de ejecución de Radix Sort con su complejidad teórica O(n)*O*(*n*). Se observa una buena aproximación entre el tiempo de ejecución real y la complejidad teórica, mostrando un crecimiento lineal. A diferencia de otros algoritmos, Radix Sort mantiene una eficiencia consistente a medida que aumenta el número de elementos, lo que confirma su complejidad lineal.

A graph with blue and orange lines

AI-generated content may be incorrect.

Esta gráfica muestra el tiempo de ejecución de Merge Sort junto con su complejidad teórica O(nlog⁡n)*O*(*n*log*n*). Se observa que el tiempo de ejecución real sigue una curva similar a la complejidad teórica, aunque con una ligera desviación. El crecimiento es menos pronunciado que el de los algoritmos con complejidad O(n2)*O*(*n*2), lo que confirma la eficiencia de Merge Sort en comparación con algoritmos cuadráticos. La pequeña discrepancia entre la curva real y la teórica puede atribuirse a factores como la sobrecarga del sistema o detalles de la implementación.

A graph with different colored lines

AI-generated content may be incorrect.

La gráfica muestra el tiempo de ejecución de Insertion Sort contra la cantidad de elementos. Se observa un comportamiento no lineal, con un tiempo de ejecución casi constante hasta aproximadamente 1000 elementos, y luego un aumento. Esto difiere significativamente de la complejidad teórica O(n) representada por la línea punteada, que predice un crecimiento lineal. La discrepancia podría deberse a la sobrecarga de la máquina o a la implementación específica del algoritmo, que podría no ser óptima para todos los tamaños de entrada.

A graph with a line and numbers

AI-generated content may be incorrect.

Esta gráfica compara el tiempo de ejecución de Selection Sort con su complejidad teórica O(n²). Se aprecia un crecimiento cuadrático del tiempo de ejecución, coincidiendo aproximadamente con la curva teórica. Esto confirma el comportamiento esperado de Selection Sort, donde el tiempo de ejecución aumenta proporcionalmente al cuadrado del número de elementos.

A graph with a line

AI-generated content may be incorrect.

La gráfica muestra el tiempo de ejecución de Quick Sort frente a su complejidad teórica en el peor caso, O(n²). El tiempo de ejecución real se aproxima a la complejidad teórica, especialmente para un mayor número de elementos. Esto sugiere que el algoritmo podría estar experimentando casos cercanos al peor escenario en esta ejecución, posiblemente debido a una mala elección de pivotes. En un caso promedio, Quick Sort suele tener una complejidad de O(n log n), pero esta gráfica indica un comportamiento cercano al peor caso.

A graph with a line and a line

AI-generated content may be incorrect.

La gráfica compara el tiempo de ejecución de Radix Sort con su complejidad teórica O(n). Se observa una aproximación razonable entre el tiempo de ejecución real y la complejidad teórica, mostrando un crecimiento lineal. A diferencia de los algoritmos cuadráticos, Radix Sort mantiene una eficiencia consistente a medida que aumenta el número de elementos, confirmando su complejidad lineal. Sin embargo, se observan algunas fluctuaciones en el tiempo de ejecución real.

A graph with a line and a line

AI-generated content may be incorrect.

La gráfica muestra el tiempo de ejecución de Merge Sort junto con su complejidad teórica O(n log n). Se observa que el tiempo de ejecución real sigue una curva similar a la complejidad teórica, aunque con una ligera desviación. El crecimiento es menos pronunciado que el de los algoritmos con complejidad O(n²), lo que confirma la eficiencia de Merge Sort. La pequeña discrepancia entre la curva real y la teórica puede atribuirse a factores como la sobrecarga del sistema o detalles de la implementación.

A graph with a line

AI-generated content may be incorrect.

Se observa que el tiempo de ejecución real (línea azul) sigue una tendencia similar a la complejidad teórica (línea naranja discontinua), mostrando un crecimiento que no es lineal, pero tampoco cuadrático. La curva del tiempo real se encuentra por encima de la curva teórica, lo que indica una discrepancia. Esta diferencia puede atribuirse a factores como la sobrecarga del sistema, la implementación específica del algoritmo o la influencia de factores constantes ocultos en la notación Big O. A pesar de la discrepancia, la gráfica confirma el comportamiento logarítmico-lineal característico de Merge Sort, mostrando un crecimiento significativamente menor que los algoritmos con complejidad O(n2)*O*(*n*2).

Esta gráfica permite visualizar la eficiencia de cada algoritmo a medida que aumenta el tamaño de los datos.

**Conclusiones**

Insertion Sort y Selection Sort presentan un crecimiento cuadrático en el tiempo de ejecución, lo que los hace ineficientes para grandes volúmenes de datos.

Merge Sort y Quick Sort son significativamente más rápidos, debido a su complejidad de O(n log n).

Radix Sort, en este caso, tuvo un rendimiento óptimo en comparación con los demás algoritmos.