Escritura del problema del ordenamiento de datos

Alejandro Morales Contreras[1]

[1]Departamento de Ingeniería de Sistemas, Pontificia Universidad Javeriana
Bogotá, Colombia
a.moralesc@javeriana.edu.co

28 de julio de 2022

Resumen

En este documento se presenta la formalización del problema de ordenamiento de datos, junto con la descripción de tres algoritmos que lo solucionan. Además, se presenta un análisis experimental de la complejidad de esos tres algoritmos. **Palabras clave:** ordenamiento, algoritmo, formalización, experimentación, complejidad.

Índice

1. Introducción

Los algoritmos de ordenamiento de datos son muy útiles en una cantidad considerable de algoritmos que requieren orden en los datos que serán procesados. En este documento se presentan tres de ellos, con el objetivo de mostrar: la formalización del problema (sección 2), la escritura formal de tres algoritmos (sección 3) y un análisis experimental de la complejidad de cada uno de ellos (sección 4).

2. Formalización del problema

Cuando se piensa en el *ordenamiento de números* la solución inmediata puede ser muy simplista: inocentemente, se piensa en ordenar números. Sin embargo, con un poco más de reflexión, hay tres preguntas que pueden surgir:

1. ¿Cuáles números?
2. ¿Cómo se guardan esos números en memoria?
3. ¿Solo se pueden ordenar números?

Recordemos que los números pueden ser naturales ($\mathbb{N}$), enteros ($\mathbb{Z}$), racionales o quebrados ($\mathbb{Q}$), irracionales ($\mathbb{I}$) y complejos ($\mathbb{C}$). En todos esos conjuntos, se puede definir la relación de *orden parcial* $a < b$.

Esto lleva a pensar: si se puede definir la relación de orden parcial $a < b$ en cualquier conjunto $\mathbb{T}$, entonces se puede resolver el problema del ordenamiento con elementos de dicho conjunto.

2.1. Definición del problema del "ordenamiento de datos"

Así, el problema del ordenamiento se define a partir de:

1. una secuencia S de elementos $a \in \mathbb{T}$ y
2. una relación de orden parcial $a < b \ \forall a, b \in \mathbb{T}$

producir una nueva secuencia S' cuyos elementos contiguos cumplan con la relación $a < b$.

- Entradas:
  - $S = \langle a_i \in \mathbb{T} \rangle \mid 1 \leq i \leq n$.
  - $a < b \in \mathbb{T} \times \mathbb{T}$, una relación de orden parcial.
- Salidas:
  - $S' = \langle e_i \in Sm \rangle \mid e_i < e_{i+1} \forall i \in [1, n)$.

3. Algoritmos de solución

3.1. Burbuja "inocente"

La idea de este algoritmo es: comparar todos las parejas de elementos adyacentes e intercambiarlos si no cumplen con la relación de orden parcial $<$.

3.1.1. Análisis de complejidad

Por inspección de código: hay dos ciclos *para-todo* anidados que, en el peor de los casos, recorren todo la secuencia de datos; entonces, este algoritmo es $O(|S|^2)$.

Algoritmo 1 Ordenamiento por burbuja "inocente".

Require: $S = \langle s_i \in \mathbb{T} \rangle \wedge a < b \in \mathbb{T} \times \mathbb{T}$
Ensure: S será cambiado por $S' = \langle e_i \in S \mid e_i < e_{i+1} \forall i \in [1, n) \rangle$

```
1: procedure NaiveBubbleSort(S)
2:     for i ← 1 to |S| do
3:         for j ← 1 to |S| − 1 do
4:             if s_{j+1} < s_j then
5:                 Swap(s_j, s_{j+1})
6:             end if
7:         end for
8:     end for
9: end procedure
```

3.1.2. Invariante

Después de cada iteración controlada por el contador i, los i elementos más grandes quedan al final de la secuencia.

1. Inicio: $i = 0$, la secuencia vacía está ordenada.
2. Iteración: $1 \leq i < |S|$, si se supone que los $i-1$ elementos más grandes ya están en su posición, entonces la nueva iteración llevará los i-ésimo elemento a su posición adecuada.
3. Terminación: $i = |S|$, los $|S|$ elementos más grandes están en su posición, entonces la secuencia está ordenada.

3.2. Burbuja "mejorado"

La idea de este algoritmo es: comparar todos las parejas de elementos adyacentes e intercambiarlos si no cumplen con la relación de orden parcial $<$, con la diferencia que las comparaciones se detienen en el momento que se alcanzan los elementos más grandes que ya están en su posición final.

Algoritmo 2 Ordenamiento por burbuja "mejorado".

Require: $S = \langle S_i \in \mathbb{T} \rangle \wedge a < b \in \mathbb{T} \times \mathbb{T}$
Ensure: S será cambiado por $S' = \langle e_i \in S \mid e_i < e_{i+1} \forall i \in [1, n) \rangle$

```
1: procedure ImprovedBubbleSort(S)
2:     for i ← 1 to |S| do
3:         for j ← 1 to |S| − i do        ▷ Mejora: parar cuando se encuentren los elementos más grandes.
4:             if s_{j+1} < s_j then
5:                 Swap(s_j, s_{j+1})
6:             end if
7:         end for
8:     end for
9: end procedure
```

3.2.1. Análisis de complejidad

Por inspección de código: hay dos ciclos *para-todo* anidados que, en el peor de los casos, recorren todo la secuencia de datos; entonces, este algoritmo es $O(|S|^2)$.

3.2.2. Invariante

Después de cada iteración controlada por el contador i, los i elementos más grandes quedan al final de la secuencia.

1. Inicio: $i = 0$, la secuencia vacía está ordenada.
2. Iteración: $1 \leq i < |S|$, si se supone que los $i-1$ elementos más grandes ya están en su posición, entonces la nueva iteración llevará los i-ésimo elemento a su posición adecuada.
3. Terminación: $i = |S|$, los $|S|$ elementos más grandes están en su posición, entonces la secuencia está ordenada.

3.3. Inserción

La idea de este algoritmo es: en cada iteración, buscar la posición donde el elemento que se está iterando quede en el orden de secuencia adecuado.

Algoritmo 3 Ordenamiento por inserción.

Require: $S = \langle S_i \in \mathbb{T} \rangle \land a < b \in \mathbb{T} \times \mathbb{T}$
Ensure: S será cambiado por $S' = \langle e_i \in Sm \rangle \mid e_i < e_{i+1} \forall i \in [1, n)$

```
1:  procedure InsertionSort(S)
2:      for j ← 2 to |S| do
3:          k ← s_j
4:          i ← j − 1
5:          while 0 < i ∧ k < s_i do
6:              s_{i+1} ← s_i
7:              i ← i − 1
8:          end while
9:          s_{i+1} ← k
10:     end for
11: end procedure
```

3.3.1. Análisis de complejidad

Por inspección de código: hay dos ciclos (un *mientras-que* anidado dentro de un ciclo *para-todo*) anidados que, en el peor de los casos, recorren todo la secuencia de datos; entonces, este algoritmo es $O(|S|^2)$.

El ciclo interior, por el hecho de ser *mientras-que*, puede que en algunas configuraciones no se ejecute (i.e. cuando la secuencia ya esté ordenada); entonces, este algoritmo tiene una cota inferior $\Omega(|S|)$, dónde solo el *para-todo* recorre la secuencia.

3.3.2. Invariante

Después de cada iteración j, los primeros j siguen la relación de orden parcial $a < b$.

1. Inicio: $j \leq 1$, la secuencia vacía o unitaria está ordenada.
2. Iteración: $2 \leq j < |S|$, si se supone que los $j-1$ elementos ya están ordenados, entonces la nueva iteración llevará un nuevo elemento y los j primeros elementos estarán ordenados.
3. Terminación: $j = |S|$, los $|S|$ primeros elementos están ordenados, entonces la secuencia está ordenada.

4. Análisis experimental

En esta sección se presentarán algunos los experimentos para confirmar los órdenes de complejidad de los tres algoritmos presentados en la sección 3.

4.1. Secuencias aleatorias

Acá se presentan los experimentos cuando los algoritmos se ejecutan con secuencias de entrada de orden aleatorio.

4.1.1. Protocolo

1. Cargar en memoria un archivo de, al menos, 200Kb.
2. Definir un rango $(b, e, s) \in \mathbb{N}^3$, donde: b es un tamaño inicial, e es un tamaño final y s es un salto. Se generarán secuencias, a partir del archivo de entrada, de diferentes tamaños desde b hasta e, adicionando cada vez s elementos.
3. Cada algoritmo se ejecutará 10 veces con cada secuencia y se guardará el tiempo promedio de ejecución.
4. Se generan los gráficos necesarios para comparar los algoritmos.

4.2. Secuencias ordenadas

Acá se presentan los experimentos cuando los algoritmos se ejecutan con secuencias de entrada ordenadas de acuerdo al orden parcial $a < b$.

4.2.1. Protocolo

1. Definir un rango $(b, e, s) \in \mathbb{N}^3$, donde: b es un tamaño inicial, e es un tamaño final y s es un salto. Se generarán secuencias aleatorias de diferentes tamaños desde b hasta e, adicionando cada vez s elementos.
2. Se usará el algoritmo `sort(S)`, disponible en la librería básica de python, para ordenar dicha secuencia.
3. Cada algoritmo se ejecutará 10 veces con cada secuencia ordenada y se guardará el tiempo promedio de ejecución.
4. Se generan los gráficos necesarios para comparar los algoritmos.

4.3. Secuencias ordenadas invertidas

Acá se presentan los experimentos cuando los algoritmos se ejecutan con secuencias de entrada ordenadas de forma invertida de acuerdo al orden parcial $a < b$.

4.3.1. Protocolo

1. Definir un rango $(b, e, s) \in \mathbb{N}^3$, donde: b es un tamaño inicial, e es un tamaño final y s es un salto. Se generarán secuencias aleatorias de diferentes tamaños desde b hasta e, adicionando cada vez s elementos.
2. Se usará el algoritmo `sort(S)`, disponible en la librería básica de python, para ordenar dicha secuencia.
3. Cada algoritmo se ejecutará 10 veces con cada secuencia ordenada y se guardará el tiempo promedio de ejecución.
4. Se generan los gráficos necesarios para comparar los algoritmos.

5. Resultados de los experimentos

Para la realización de los experimentos, se sigue el protocolo para cada tipo de secuencias que fueron definidas en la sección 4. Los resultados de los experimentos esperan probar (o refutar) los análisis de complejidad realizados para cada uno de los algoritmos en la sección 3.

Para analizar estos resultados, se hace primeramente una inspección empírica de la gráfica resultante en donde se grafica el tamaño de la secuencia (número de elementos) contra el tiempo promedio que le toma a cada algoritmo ordenarla. Por ejemplo, para todos los algoritmos, el peor de los casos son secuencias aleatorias, las cuales representan una complejidad de $O(|S|^2)$. Se esperaría que la gráfica resultante fuera una parábola.

Cabe resaltar que, aunque los algoritmos tengan una complejidad equivalente, las gráficas no serán exactamente similares. Las optimizaciones que los algoritmos implementen podrán significar un crecimiento más lento que un algoritmo menos optimizado. Así mismo, la secuencia dada para el experimento también variará considerablemente el desempeño del algoritmo. Sin embargo, la medida de complejidad sigue siendo la adecuada para escoger el algoritmo.

Después de la inspección de la gráfica, es necesario confirmar más detalladamente si realmente los resultados prueban el análisis de complejidad. Una buena forma de confirmar esto es mediante una regresión de los datos. A excepción del algoritmo por inserción en el mejor de sus casos (una secuencia ordenada), todos los algoritmos presentan una complejidad de $O(|S|^2)$. Es por esto que una regresión cuadrática es la operación adecuada para confirmar los resultados.

La regresión cuadrática pretende ajustar una curva de la forma:

$$f(X) = aX^2 + bX + c$$

donde X representaría la variable independiente (tamaño de la secuencia) y $f(X)$ la variable dependiente (tiempo que toma ordenar la secuencia). Después de la regresión, se obtienen los coeficientes del polinomio a, b y c.

La medida que se utiliza para determinar si la regresión se cumple es el coeficiente de determinación (R^2), el cual refleja que tan bien ajustado es el polinomio resultante a los datos de entrada. Esta se calcula así:

$$R^2 = \frac{\sum_{n=1}^{N}(\hat{Y_n} - \bar{Y})^2}{\sum_{n=1}^{N}(Y_n - \bar{Y})^2}$$

en donde:

- N representa la cantidad de elementos o tamaño de la secuencia
- $\bar{Y}$ representa la media de la variable dependiente de entrada
- $\hat{Y_n}$ representa $f(X_n)$ o el resultado de evaluar el polinomio de regresión en X_n
- Y_n representa el n-ésimo dato de la variable dependiente de entrada

R^2 varía en el rango $(0, 1)$ y a mayores medidas de este se espera una mayor confianza en que los resultados se asemejan a un polinomio cuadrático.

5.1. Secuencias aleatorias

Para el experimento con secuencias aleatorias, se sigue el protocolo definido en la sección 4.1, dando como rango (b, e, s) los datos $(0, 10000, 100)$. Obtenidos los resultados, se genera el gráfico comparativo de los algoritmos presentado en la figura 1.

La inspección empírica de la gráfica parece revelar que los tres algoritmos siguen una parábola, cumpliendo con el análisis de complejidad. Así mismo, se resalta que el algoritmo de inserción tiene un mejor desempeño que los demás, mientras que el algorito de burbuja "inocente" tiene un peor desempeño que el resto.

Posteriormente, se realiza la regresión cuadrática y se obtienen los siguientes resultados:

Algoritmo	a	b	c	R^2
Burbuja inocente	6.19524e-08	3.15077e-06	1.04866e-03	0.9993555
Burbuja mejorado	3.94208e-08	2.43998e-06	1.05354e-03	0.9996105
Inserción	1.75967e-08	3.64248e-06	-2.70654e-03	0.9992336

Como se puede ver, todos los algoritmos obtienen un $R^2 > 0,999$. Esto representa un ajuste bastante alto mediante la regresión cuadrática. Es posible confirmar que, con secuencias aleatorias, estos algoritmos tienen una complejidad de $O(|S|^2)$.

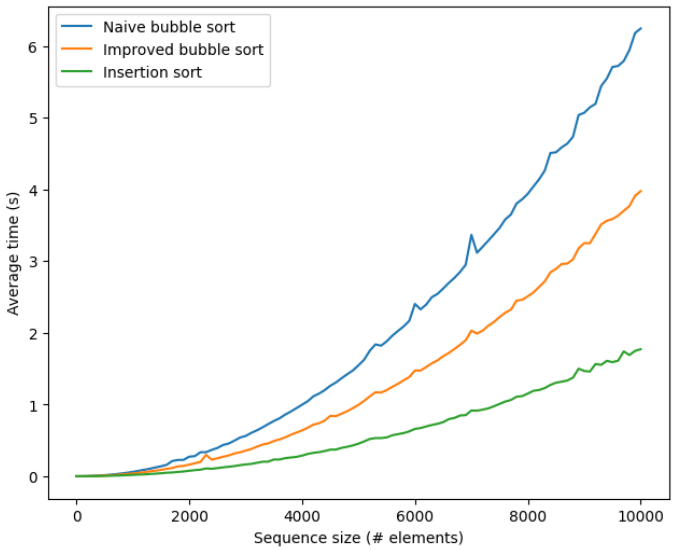

Figura 1: Gráfica comparativa con secuencias aleatorias

5.2. Secuencias ordenadas

Para el experimento con secuencias aleatorias, se sigue el protocolo definido en la sección 4.2, dando como rango (b, e, s) los datos $(0, 10000, 100)$. Obtenidos los resultados, se genera el gráfico comparativo de los algoritmos presentado en la figura 2.

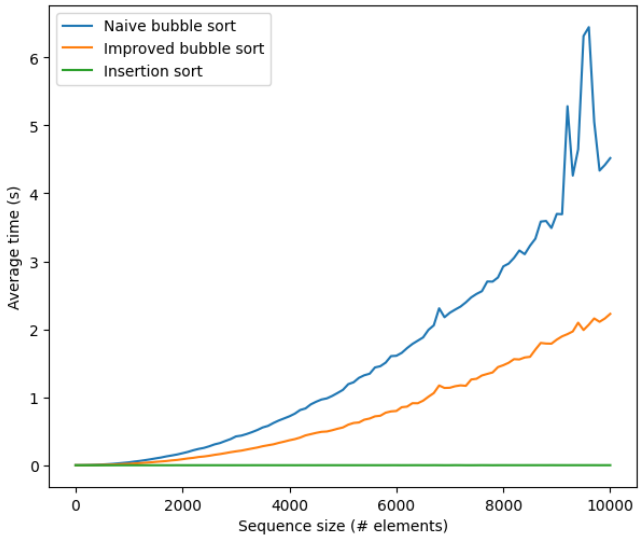

Figura 2: Gráfica comparativa con secuencias ordenadas

La inspección empírica de la gráfica parece revelar que ambos algoritmos burbuja siguen una parábola, pero el algoritmo por inserción parece ser una recta en $y = 0$. Así mismo, el algoritmo de burbuja "inocente" sigue comportándose de la misma forma que en el anterior experimento (aunque con un picos que podrían ser explicados por la prioridad que el SO asigna a este proceso). Por su parte, el algoritmo de burbuja "mejorado" tiene un desempeño levemente mejor que en el anterior experimento.

Posteriormente, se realiza la regresión cuadrática y se obtienen los siguientes resultados:

Algoritmo	a	b	c	R^2
Burbuja inocente	6.04217e-08	-1.11239e-04	1.24728e-01	0.9574511
Burbuja mejorado	2.21122e-08	6.39194e-06	-9.27824e-03	0.9986932
Inserción	-7.54501e-08	9.294548e-08	-9.51056e-06	0.7012384

Analizando los resultados, los algoritmos de burbuja obtienen un $R^2 > 0,95$. Esto sigue representando un ajuste alto mediante regresión cuadrática (La disminución de la medida para la burbuja "inocente" puede ser explicada por el fenómeno del SO). Es posible confirmar que, con secuencias ordenadas, estos algoritmos tienen una complejidad de $O(|S|^2)$.

Por otro lado, el algoritmo de inserción no tiene un buen ajuste mediante regresión cuadrática. Esto se debe a que la cota inferior del algoritmo es $\Omega(|S|)$ y esta se presenta en el mejor de los casos del algoritmo (cuando la secuencia ya está ordenada). Se realiza entonces una regresión lineal de la forma $f(X) = aX + b$ y se obtiene:

a	8.54004e-08
b	2.93870e-06
R^2	0.7008665

Como se evidencia por el valor de R^2, la confianza en que los datos representen una recta tampoco es muy grande. Sin embargo, dando un vistazo a los resultados de los tiempos, se encuentra que todos están por debajo de los 0,001s con una tendencia en crecimiento. En este punto, se puede asumir que el algoritmo sí cumple con la complejidad de la cota inferior, pero el fenómeno del SO no permite tener una medición exacta en la práctica.

5.3. Secuencias ordenadas invertidas

Finalmente, para el experimento con secuencias ordenadas invertidas, se sigue el protocolo definido en la sección 4.3, dando como rango (b, e, s) los datos $(0, 10000, 100)$. Obtenidos los resultados, se genera el gráfico comparativo de los algoritmos presentado en la figura 3.

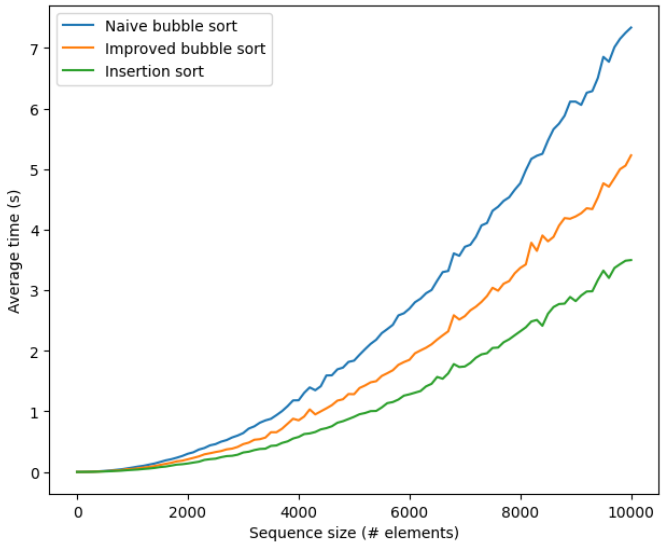

Figura 3: Gráfica comparativa con secuencias ordenadas invertidas

La inspección empírica de la gráfica vuelve a revelar que los tres algoritmos siguen una parábola, cumpliendo con el análisis de complejidad. Así mismo, todos los algoritmos tienen un desempeño levemente peor que con secuencias meramente aleatorias.

Posteriormente, se realiza la regresión cuadrática y se obtienen los siguientes resultados:

Algoritmo	a	b	c	R^2
Burbuja inocente	7.32901e-08	1.61687e-05	-2.53077e-02	0.9994136
Burbuja mejorado	5.12133e-08	1.16794e-05	-1.69050e-02	0.9988019
Inserción	3.50858e-08	7.19180e-06	-1.16396e-02	0.9989983

Como se puede observar, todos los algoritmos obtienen un $R^2 > 0,99$. Esto vuelve a representar un ajuste bastante alto mediante la regresión cuadrática. Es posible confirmar que, con secuencias ordenadas invertidas, estos algoritmos tienen una complejidad de $O(|S|^2)$.