Memoria de la práctica de eficiencia de algoritmos

Javier León Palomares, José Gómez Baena

Índice

1. Introducción.

El objetivo de esta práctica es ayudar a comprender la importancia de la eficiencia de los algoritmos a la hora de escoger entre varias opciones.

En primer lugar, analizaremos tres algoritmos de ordenación simples pero cuyos tiempos de ejecución son mejorables. Sabiendo su eficiencia teórica, procederemos a ejecutarlos en una máquina concreta para saber su eficiencia empírica en ella. Una vez hecho esto, haremos coincidir la teoría con la práctica mediante una regresión que ajustará ambos tipos de curvas. Además, se provee una comparación de los tiempos según el nivel de optimización utilizado al compilar los ejecutables, lo cual muchas veces mejora la eficiencia notablemente.

En segundo lugar, presentamos un algoritmo para contar las ocurrencias de una palabra.

En tercer y último lugar, se estudia la diferencia entre varias implementaciones de un mismo algoritmo que calcula la frecuencia de aparición de una palabra en un documento.

2. Algoritmos de ordenación.

Los tres algoritmos son $O(n^2)$. Se utilizarán distintos niveles de optimización del código para el análisis empírico de cada uno de ellos, aunque sólo se empleará la variante sin optimizar para el análisis híbrido. La entrada se obtendrá del archivo *lema.txt*, que contiene las palabras del diccionario ordenadas alfabéticamente.

2.1. Algoritmo *Burbuja*.

2.1.1. Análisis híbrido.

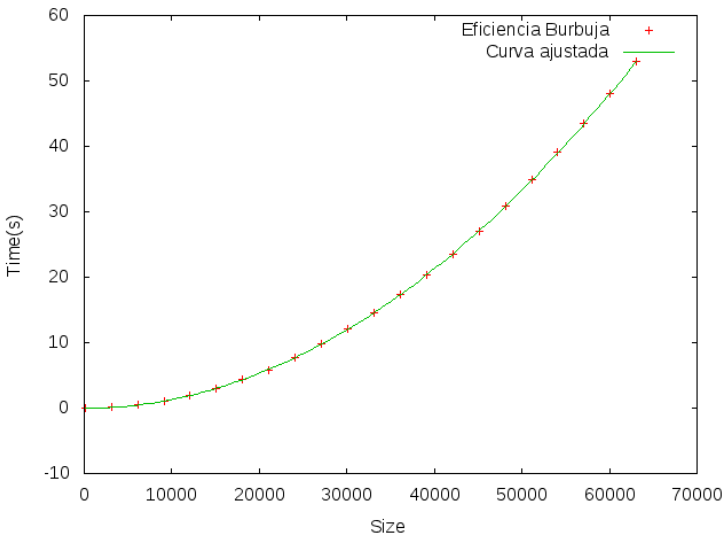

Figura 1: Intel® Core™ i7-4770K CPU @ 3.50GHz.

La función para este algoritmo es del tipo $a_0 \cdot x^2 + a_1 \cdot x + a_2$. Al ajustarla a los resultados de la ejecución obtenemos:

- $a_0 = 1{,}33424 \cdot 10^{-8}$
- $a_1 = -9{,}94981 \cdot 10^{-7}$
- $a_2 = -0{,}012692$

2.1.2. Gráfica comparativa de las optimizaciones.

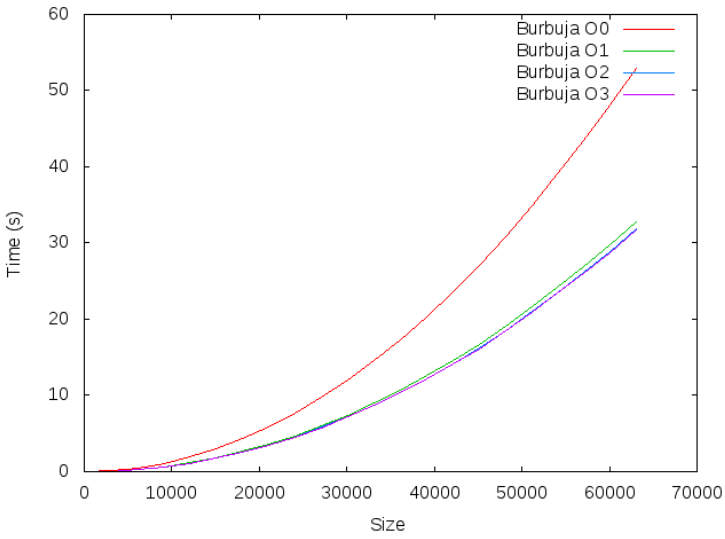

Figura 2: Intel® Core™ i7-4770K CPU @ 3.50GHz.

2.2. Algoritmo de Inserción.

2.2.1. Análisis híbrido.

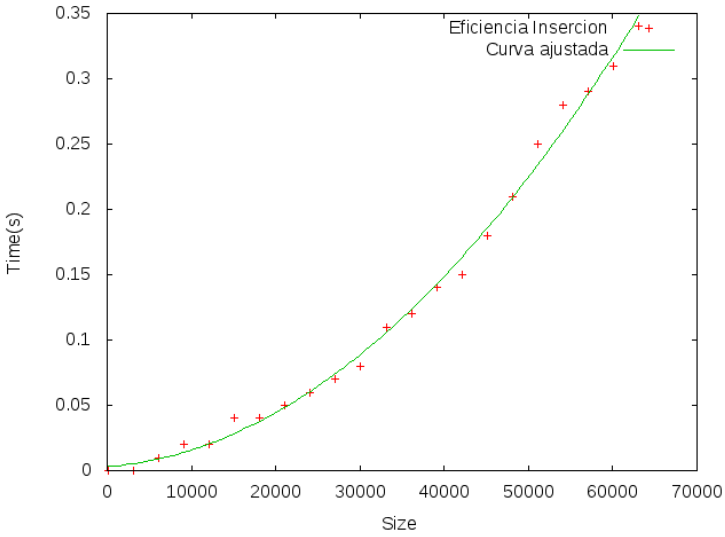

Figura 3: Intel® Core™ i7-4770K CPU @ 3.50GHz.

La función para este algoritmo es del tipo $a_0 \cdot x^2 + a_1 \cdot x + a_2$. Al ajustarla a los resultados de la ejecución obtenemos:

- $a_0 = 7{,}88945 \cdot 10^{-11}$
- $a_1 = 4{,}89115 \cdot 10^{-7}$
- $a_2 = 0{,}00309259$

2.2.2. Gráfica comparativa de las optimizaciones.

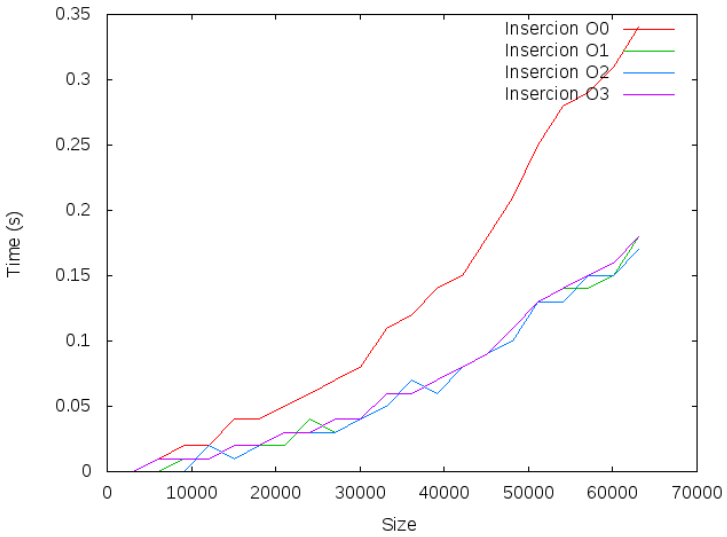

Figura 4: Intel® Core™ i7-4770K CPU @ 3.50GHz.

2.3. Algoritmo de Selección.

2.3.1. Análisis híbrido.

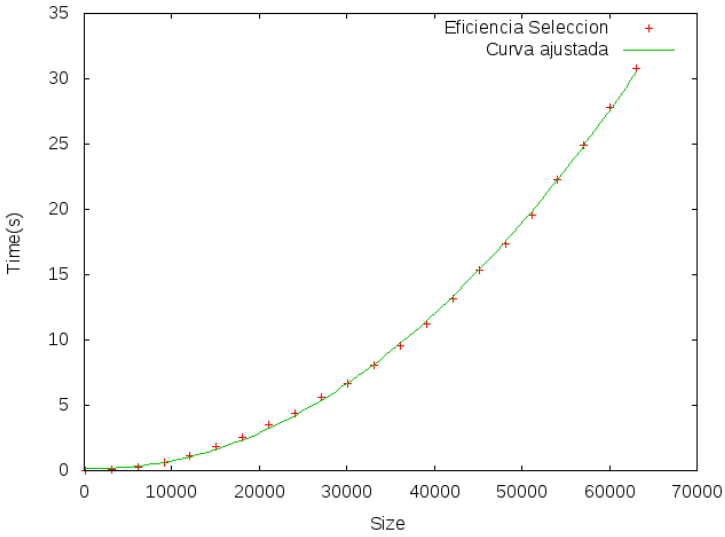

Figura 5: Intel® Core™ i7-4770K CPU @ 3.50GHz.

La función para este algoritmo es del tipo $a_0 \cdot x^2 + a_1 \cdot x + a_2$. Al ajustarla a los resultados de la ejecución obtenemos:

- $a_0 = 8{,}03728 \cdot 10^{-9}$
- $a_1 = -2{,}64505 \cdot 10^{-5}$
- $a_2 = 0{,}21044$

2.3.2. Gráfica comparativa de las optimizaciones.

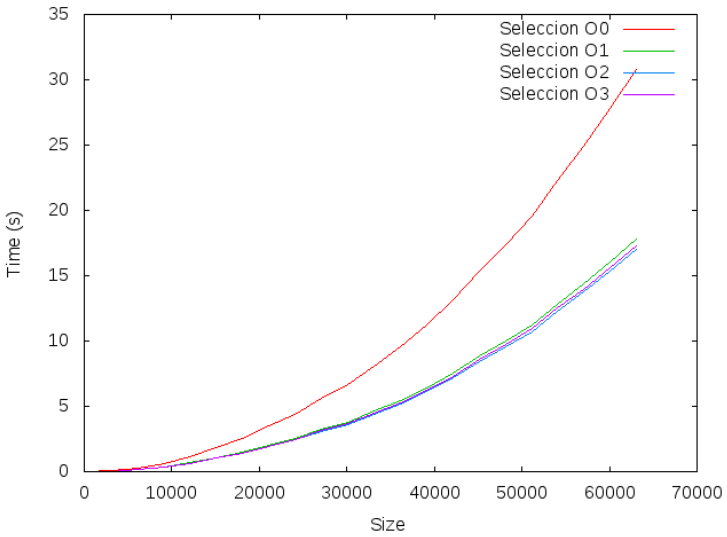

Figura 6: Intel® Core™ i7-4770K CPU @ 3.50GHz.

2.4. Comparativa.

Para esta comparativa se han usado los datos correspondientes a la compilación con un nivel de optimización 0.

2.4.1. Gráfica comparativa de tiempos entre los algoritmos.

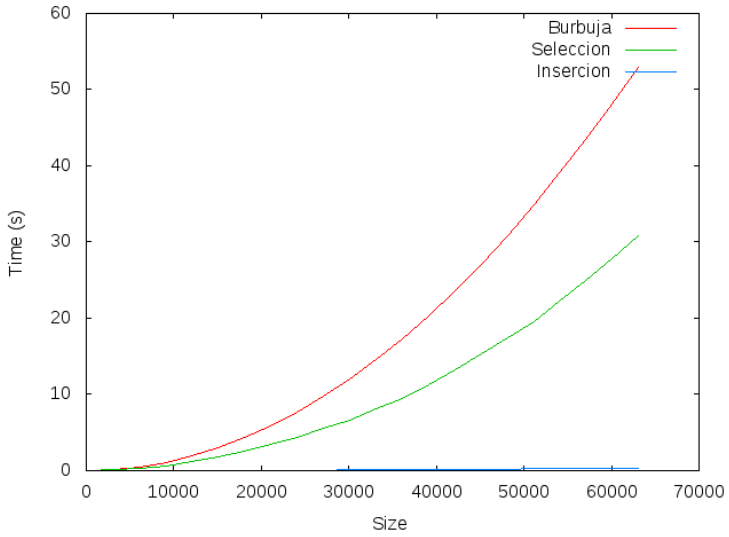

Figura 7: Intel® Core™ i7-4770K CPU @ 3.50GHz.

2.5. Análisis alternativo.

En este caso, se analizarán los tiempos de ejecución utilizando un archivo de entrada diferente, *quijote.txt*, que contiene el texto del Quijote.

2.5.1. Gráfica comparativa.

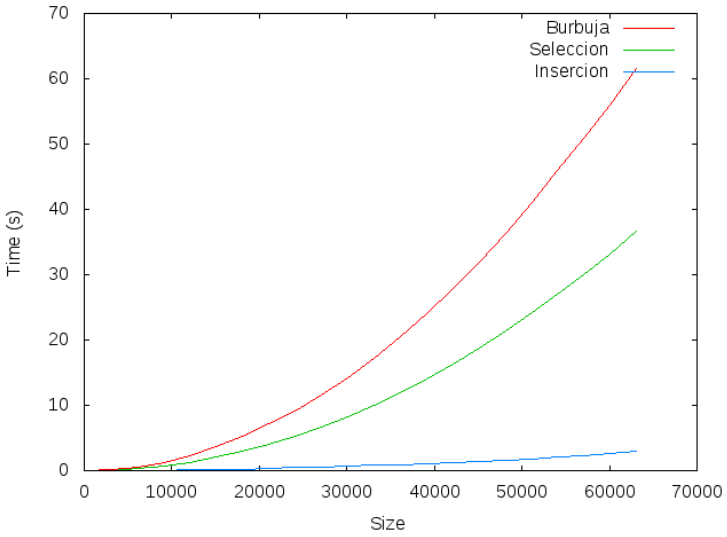

Figura 8: Intel® Core™ i7-4770K CPU @ 3.50GHz.

Observamos un aumento del tiempo de ejecución; el cambio se debe a que esta vez la entrada no tiene ningún tipo de orden, por lo que el algoritmo se aleja más del mejor caso posible.

3. Algoritmo para contar las ocurrencias de una palabra (O(n)).

Este algoritmo es de orden O(n), ya que recorre el vector una única vez secuencialmente, almacenando cada aparición de una palabra.

3.1. Análisis híbrido.

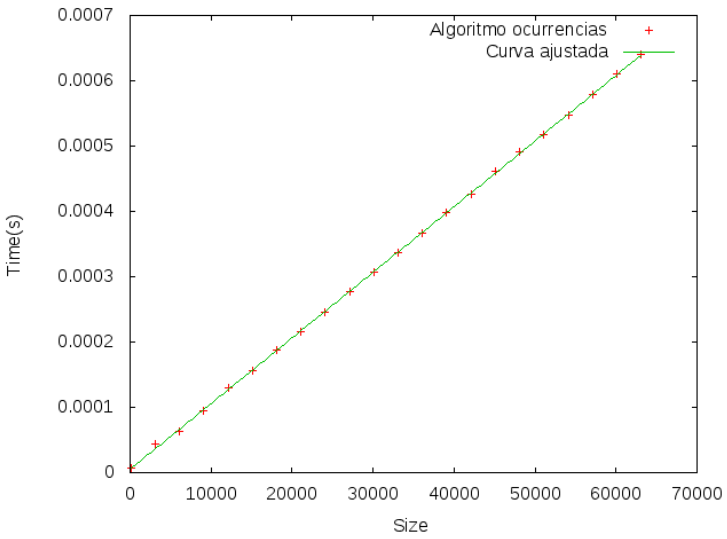

Figura 9: Intel® Core™ i5-3470 CPU @ 3.20GHz.

La función para este algoritmo es del tipo $a_0 \cdot x + a_1$. Al ajustarla a los resultados de la ejecución obtenemos:

- $a_0 = 1{,}00489 \cdot 10^{-8}$
- $a_1 = 5{,}3627 \cdot 10^{-6}$

4. Algoritmo para obtener la frecuencia de aparición de una palabra.

El objetivo de esta sección es comprobar cómo varía la eficiencia de un algoritmo según la estructura de datos utilizada. Para ello, disponemos de 4 versiones.

4.1. Gráfica comparativa.

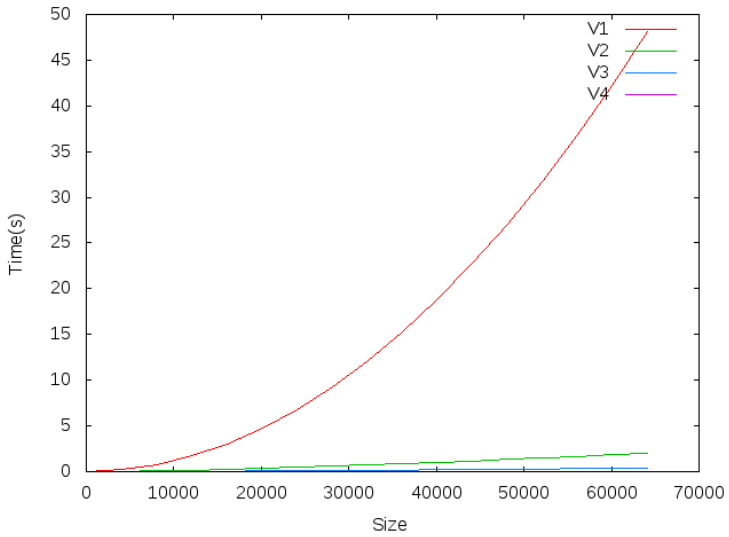

Figura 10: Intel® Core™ i5-3470 CPU @ 3.20GHz.

4.2. Conclusión de la sección.

Con esta gráfica podemos observar cómo el tiempo que consume un algoritmo puede variar enormemente según las estructuras de datos que se usen para implementarlo, ya que unas son más apropiadas que otras dependiendo del problema a resolver.