Alejandro Campoy Nieves
David Criado Ramón

PRÁCTICA EFICIENCIA ESTRUCTURA DE DATOS

Alejandro Campoy Nieves
David Criado Ramón

En esta práctica vamos a analizar la eficiencia híbrida de distintos algoritmos propuestos por el profesor.

frecuencias.cpp

En Frecuencias v1, nos encontramos con una eficiencia de orden cuadrático. El algoritmo presentado cuenta las ocurrencias de una palabra desde la posición inicial hasta la final del vector de palabras) las añade al final de un vector auxiliar pal y otro vector auxiliar en el que se tienen el número de repeticiones de la palabra.

Es importante también tener en cuenta que una palabra que aparece n veces repetida en el libro aparecerá también n veces repetida en pal y en la posición i en la que aparezca en frec[i] aparecerá n menos el número de veces que la hemos encontrado anteriormente.

Aun así, analicemos su eficiencia. En el cuerpo del bucle for nos encontramos una llamada a una función y la llamada a pushback.

Gracias a la referencia de C++ sabemos que *std::vector::push_back* $\in O(1)$.

Y la llamada a la función hace referencia a un bucle de cuerpo de $O(1)$ que se repite desde inicio hasta fin, tomando $n = fin - inicio$ la función contar_hasta $\in O(n)$

Por tanto contar_frecuencias_V1 se calcula teóricamente de la siguiente forma:

$$Suponiendo\ n = fin - inicio$$

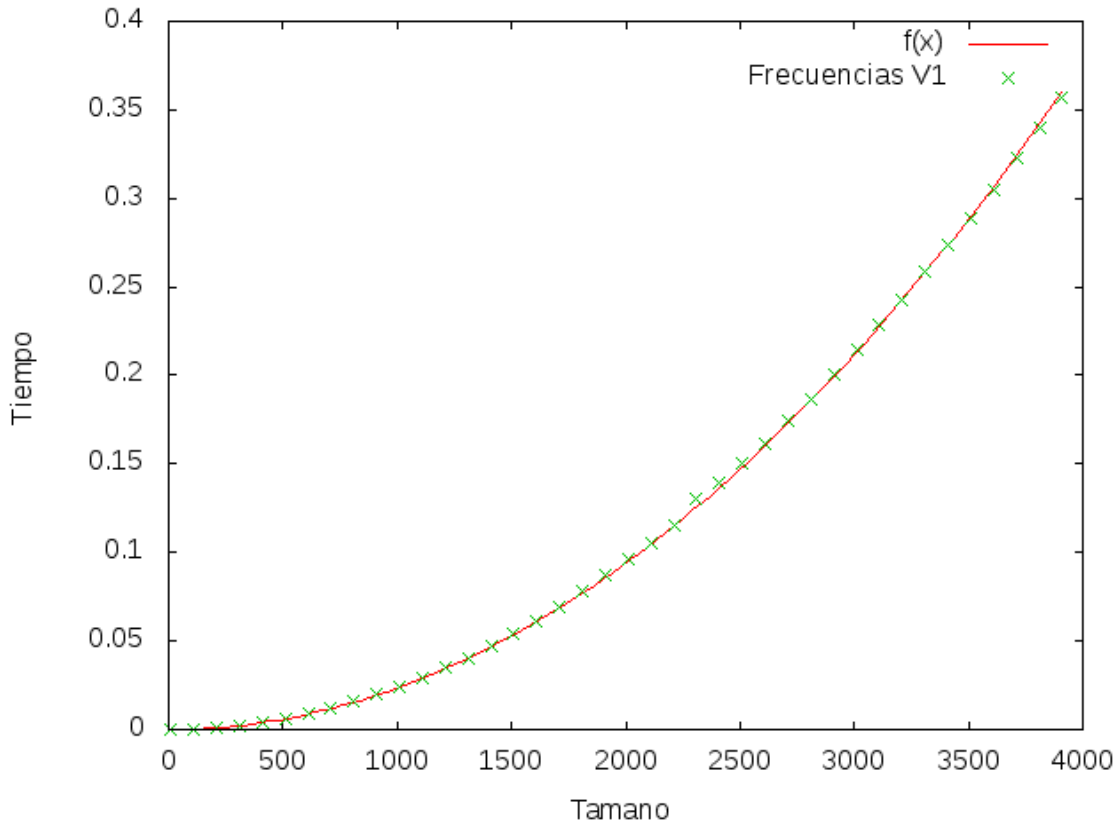
$$\sum_{i=1}^{n} O(contar_{hasta}) + O(incremento, comparador, push\_back = 1)$$

$$= \sum_{i=1}^{n} O(contar_{hasta})$$

$$= \sum_{i=1}^{n} \sum_{i=1}^{n} O(comparador, incremento, asignación, incremento = 1)$$

$$= \sum_{i=1}^{n} n = \frac{n(n+1)}{2} = \frac{n^2 + n}{2} \in O(n^2), \qquad por\ tanto:$$

$f(x) = a \cdot x^2 , \in O(n^2)$, y con gnuplot y la muestra de datos que hemos obtenido realizamos el siguiente gráfico híbrido:

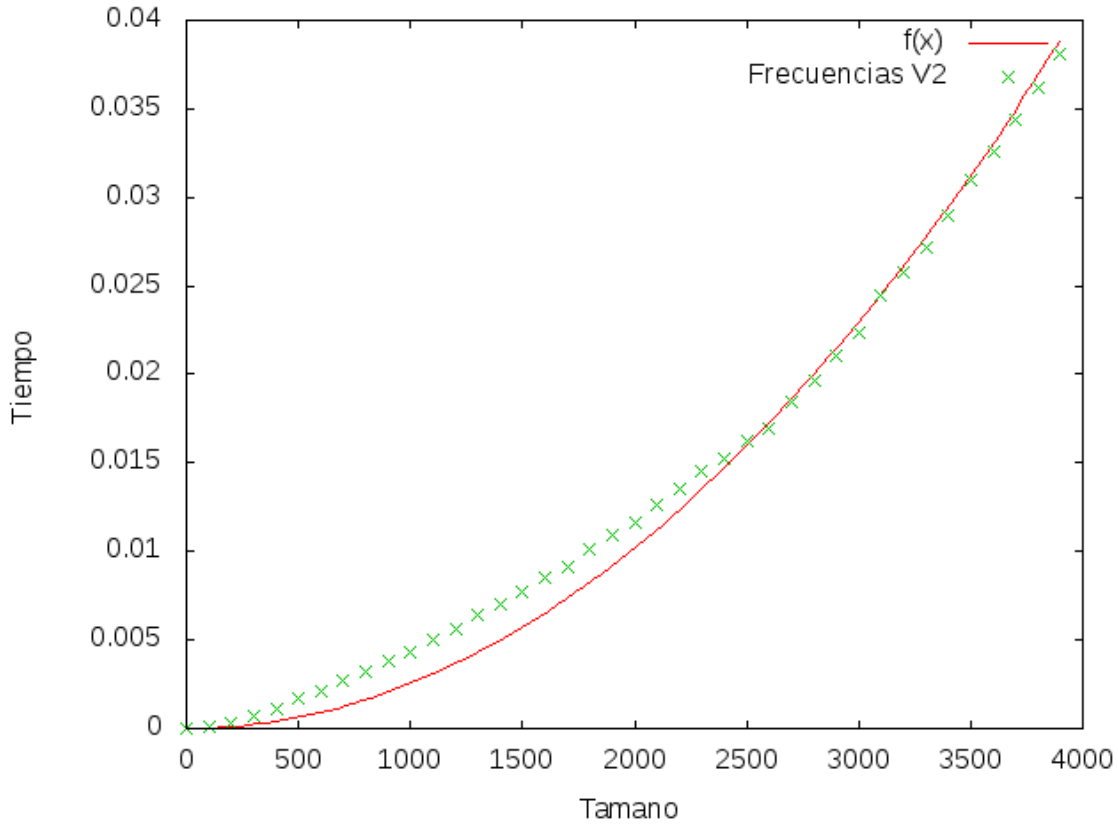


En **Frecuencias v2**, nos encontramos ante otra expresión cuadrática. Veamos en qué consiste el algoritmo esta vez:

Ahora contamos con una función auxiliar que realiza una búsqueda secuencial desde inicio hasta fin de una palabra, por tanto dicha función que solo hace incrementos, y comparaciones y devuelve la posición en la que se encuentra pertenecerá a $O(n)$, como quedó demostrado en el ejemplo anterior para una función de complejidad similar. Y análogamente este se hará desde las posiciones previamente dadas inicio y fin, por tanto esta función también será de orden cuadrático.

$$f(x) = a \cdot x^2 \in O(n^2)$$

Alejandro Campoy Nieves
David Criado Ramón

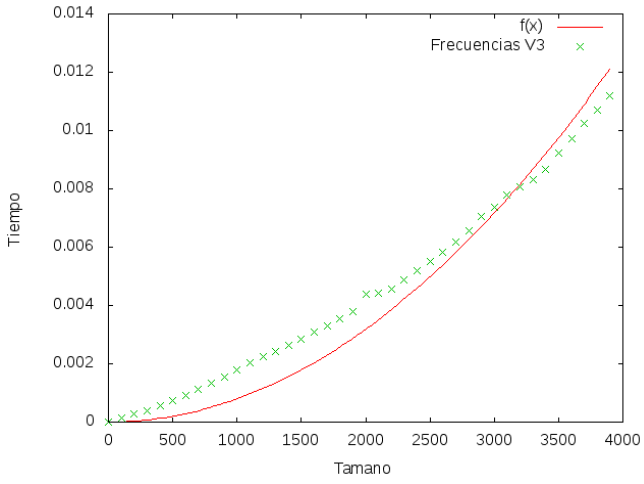


En frecuencias v3, nos encontramos con otro algoritmo de orden cuadrático. Dicho algoritmo realiza una búsqueda binaria en el vector de palabras $O(\log n)$, si dicha palabra ha sido previamente encontrada en el libro entonces se realiza un incremento $O(1)$, sin embargo si no se encuentra se inserta en una posición determinada del vector con la orden insert que es de complejidad $O(n)$. Además como esto se hace en un bucle desde inicio hasta fin obtenemos la siguiente eficiencia:

$$\sum_{i=1}^{n} O(busc_{binaria} = \log n) + O(comp, inc = 1) + O(vector :: insert = n) = \sum_{i=1}^{n} n$$

$$\in O(n^2), \qquad demostrado\ previamente$$

Además su análisis híbrido corresponde con la siguiente gráfica donde $f(x) = a \cdot x^2$



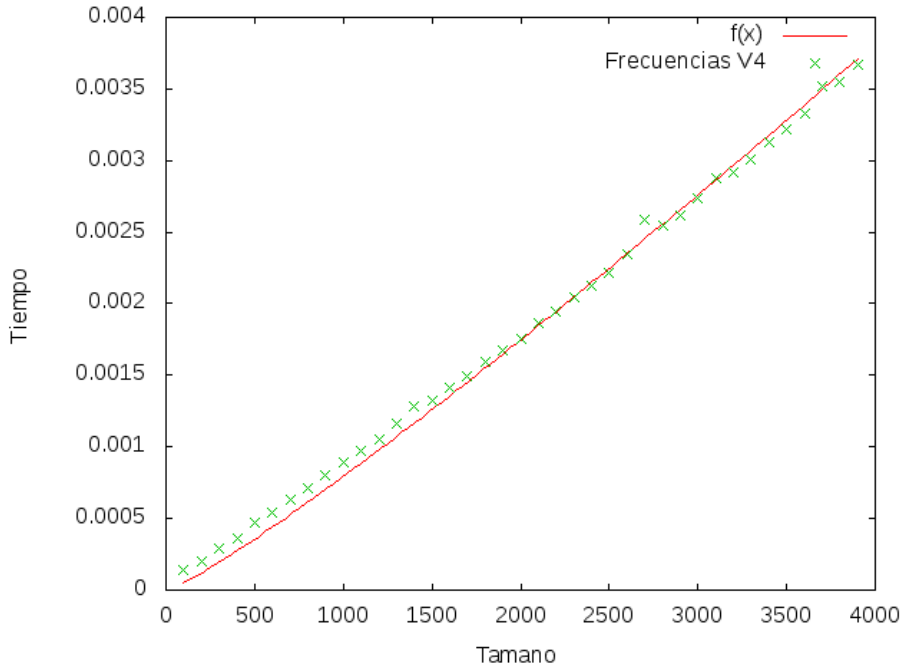
Alejandro Campoy Nieves
David Criado Ramón

En frecuencias v4, tenemos un bucle que os de orden $k \log k$ y otro que se ejecuta $n = fin - inicio$ veces una orden $\log n$. Dicho algoritmo utiliza un map como estructura de datos.

$$Como\ k = numero\ de\ palabras\ distintas\ \ y\ n\ es\ el\ número\ de\ palabras\ totales$$

$$k \leq n, entonces \sum_{i=1}^{n} O(n \log n) + O(k \log k) = O(n \log n)$$

Y la gráfica híbrida de $f(x) = a \cdot x \cdot \log x$, generada con gnuplot es la siguiente:



ocurrencias.cpp

En ocurrencias v1 y en ocurrencias v2, se utiliza el mismo algoritmo que realiza n veces algo de orden $O(1)$, por tanto el algoritmo es de $O(n)$. La diferencia es que para el segundo se toman muchas más muestras para obtener resultados más precisos. Así pues ambos algoritmos responden a la función con parámetro a $f(x) = a \cdot x$ y sus gráficas híbridas son las que se muestran a continuación después de explicar los ordenamientos.

ordenacion.cpp

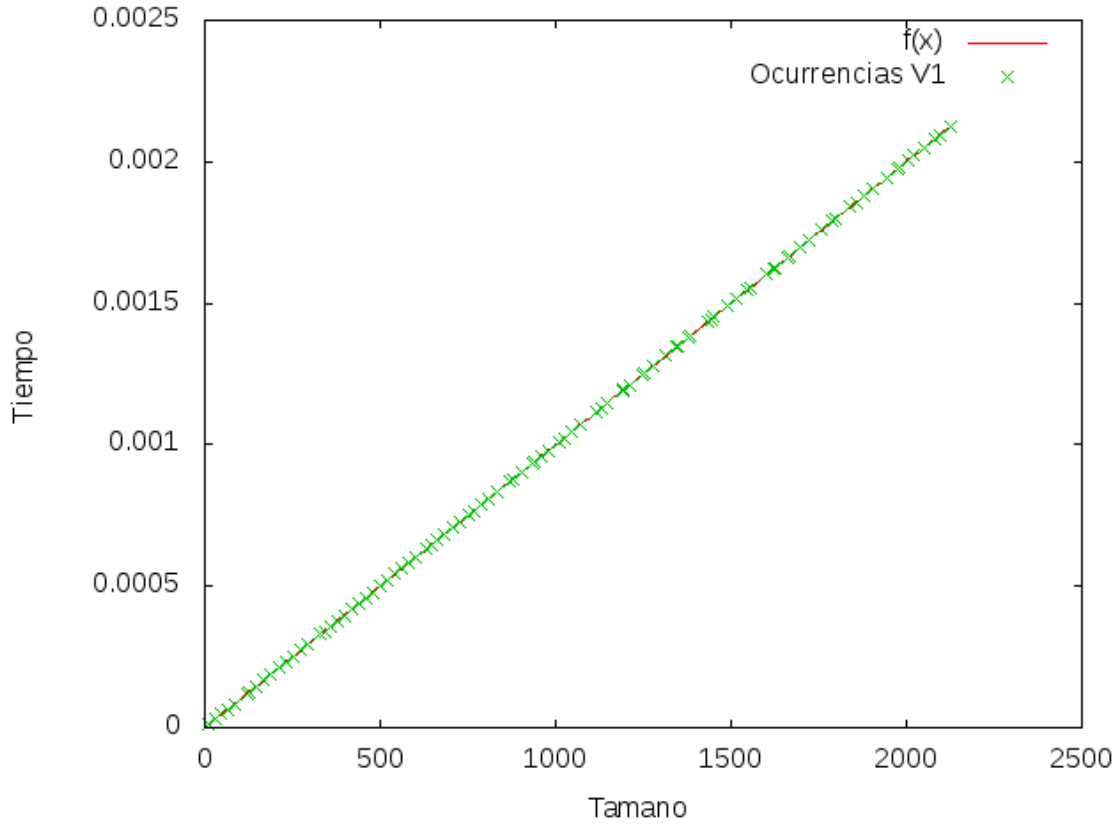
Aquí se presentan dos algoritmos, sort de $O(n \log n)$ y proporcionado por la STL y burbuja o mergeshot $O(n^2)$. El algoritmo burbuja tiene dicha complejidad porque se realizan dos bucles de $O(n)$ que realiza acciones de $O(1), previamente\ demostrado\ que\ esto \in O(n^2)$. El segundo algoritmo no sabemos cómo está implementado pero la referencia de C++ nos dice que :

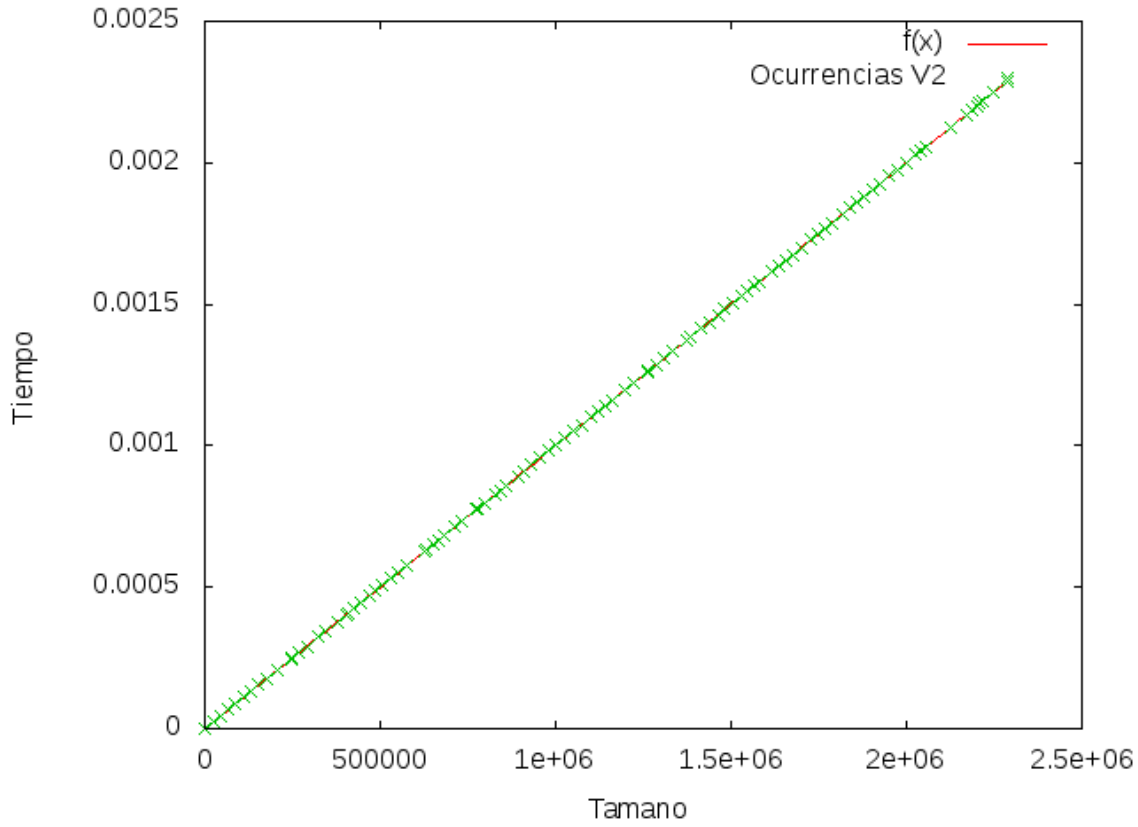
$$std :: sort \in O(n \log n)$$

Por tanto las funciones con parámetro a correspondientes son:

$$burbuja\ f(x) = a \cdot x^2, \ \ sort\ f(x) = a \cdot x \cdot \log x$$

Alejandro Campoy Nieves
David Criado Ramón





Alejandro Campoy Nieves
David Criado Ramón