PRACTICA 1

Aarón Rodríguez Bueno - Bryan Moreno Picamán

14/15

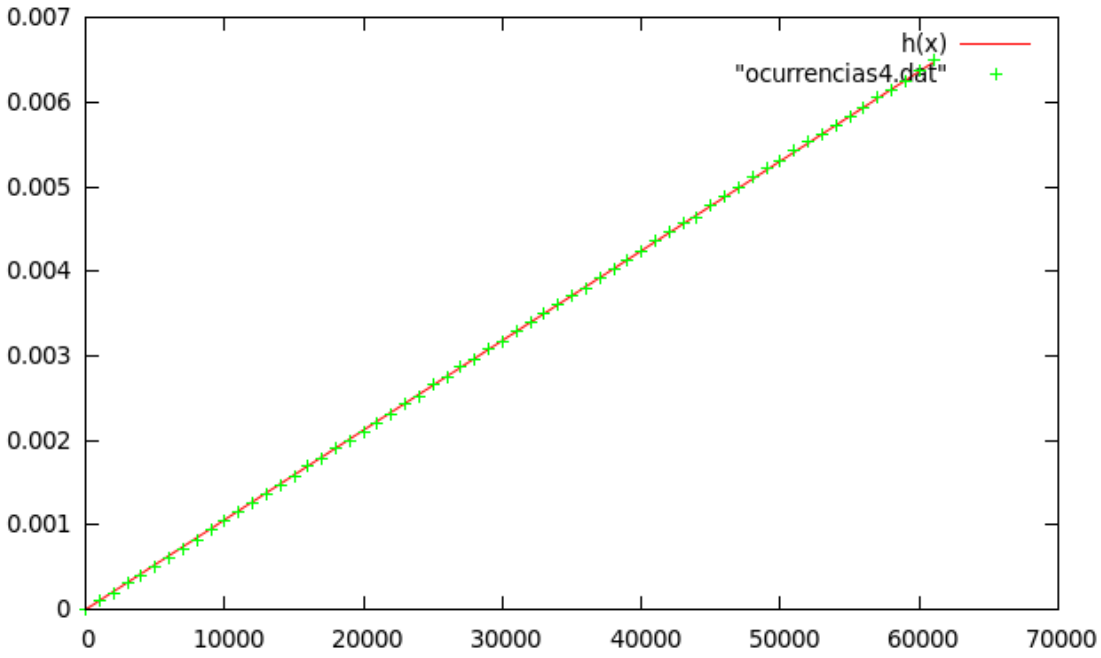
ETSIIT UGR

Estructura de Datos

Actividad 1

Realizar el análisis de eficiencia cuando consideramos el código en ocurrencias.cpp. En este caso, debemos de modificar el código para asegurarnos que tenemos la misma salida que en el ejemplo del algoritmo burbuja (tamaño y tiempo).

Después de realizar una recogida de datos normal, no se aprecia un tiempo para poder realizar un análisis, así que se usa un bucle con varias iteraciones para poder recoger los datos de forma mas correcta.



Actividad 2

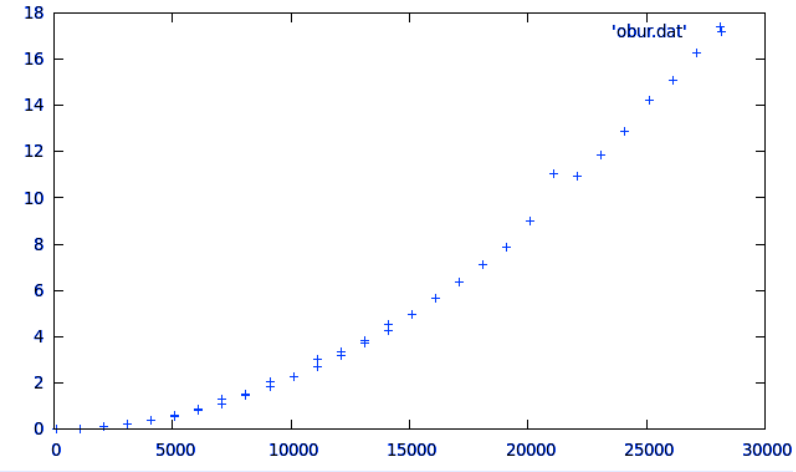
Realizar el análisis de eficiencia teórico y práctico con los algoritmos de ordenación que se conocen (burbuja, inserción y selección) considerando como entradas el fichero lema.txt. Una vez realizado el análisis individual realizar una comparación conjunta de su eficiencia.

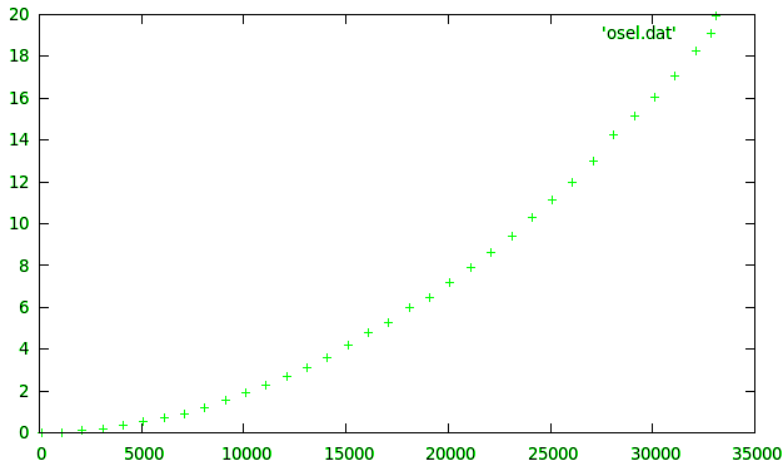
Se ha realizado la implementación de los 3 algoritmos especificados después de lo cual se han obtenido 3 archivos de datos para realizar un análisis de eficiencia:

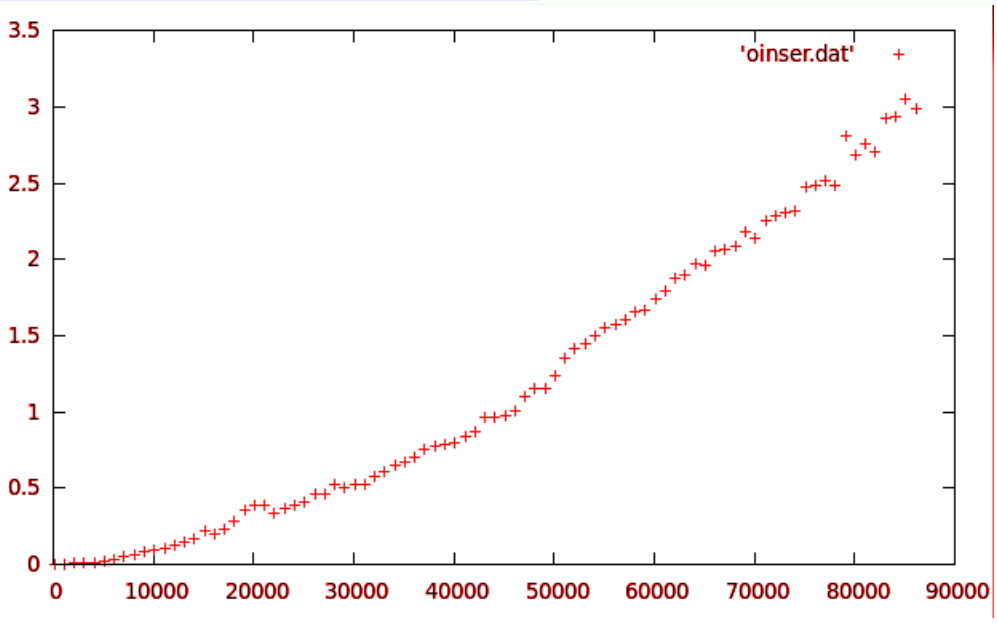
Burbuja – obur.dat

Selección – osel.dat

Inserción – oinser.dat







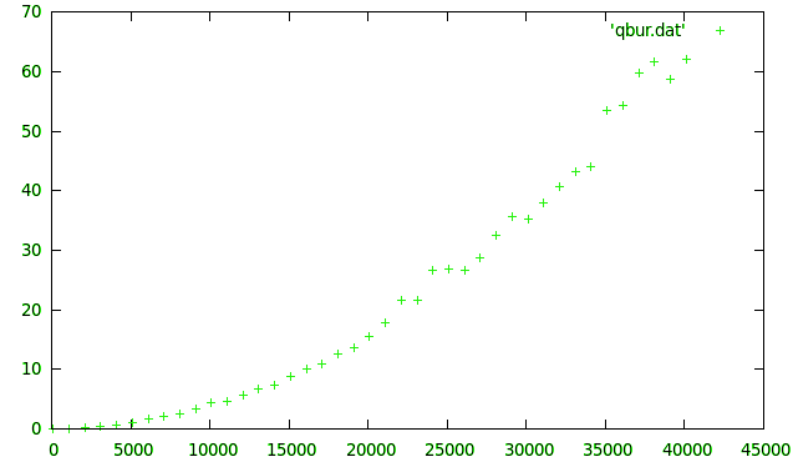
Estas mediciones han sido realizadas con lema.txt

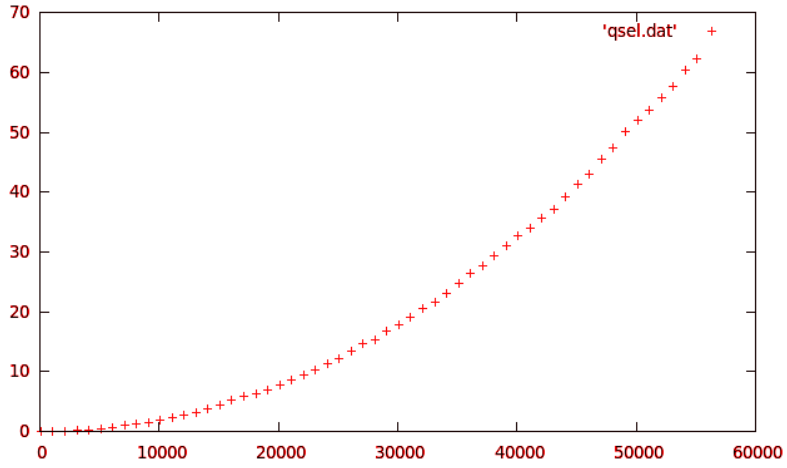
Repetir el análisis de los algoritmos de ordenación considerando que queremos ordenar en lugar del diccionario el texto del Quijote. Es suficiente con hacerlo con las 85.000 primeras palabras del fichero quijote.txt. Para ello, podemos sustituir en el esqueleto del código dado como ejemplo aux =Dicc; por aux =Q;

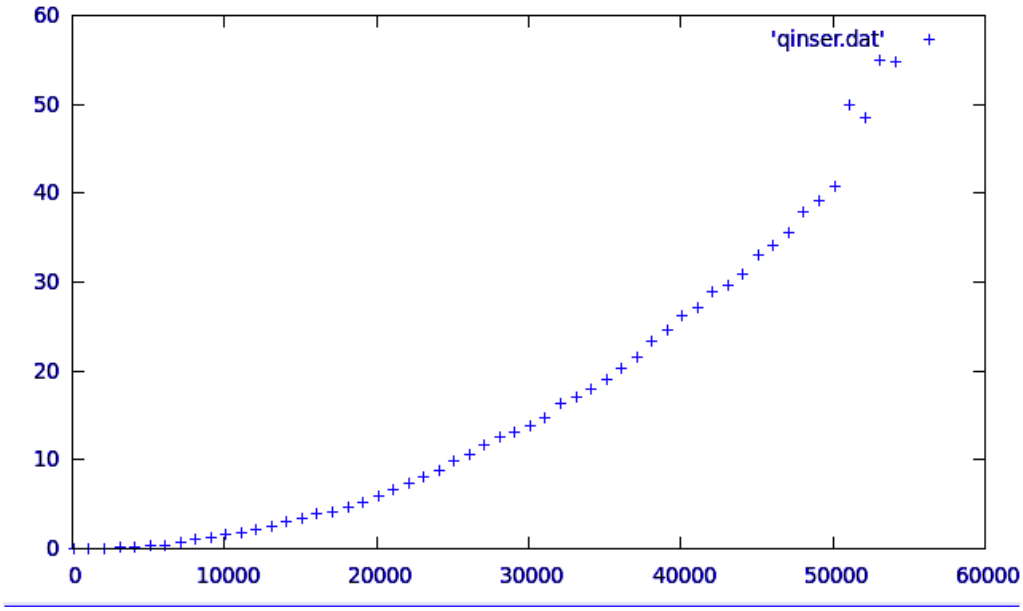
Burbuja – qbur.dat

Selección – qsel.dat

Inserción – qinser.dat







Estas mediciones han sido realizadas con quijote.txt

Analizar qué está ocurriendo.

Podemos observar que el algoritmo de inserción es más efectivo y que el resto de algoritmos tienen un comportamiento similar para cantidades pequeñas, pero se diferencian cuando crece en datos dando una pequeña ventaja al algoritmo de selección.

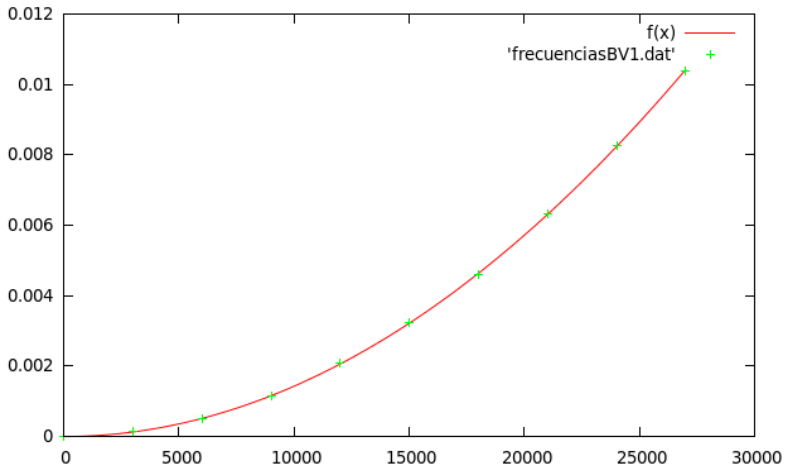
Actividad 3

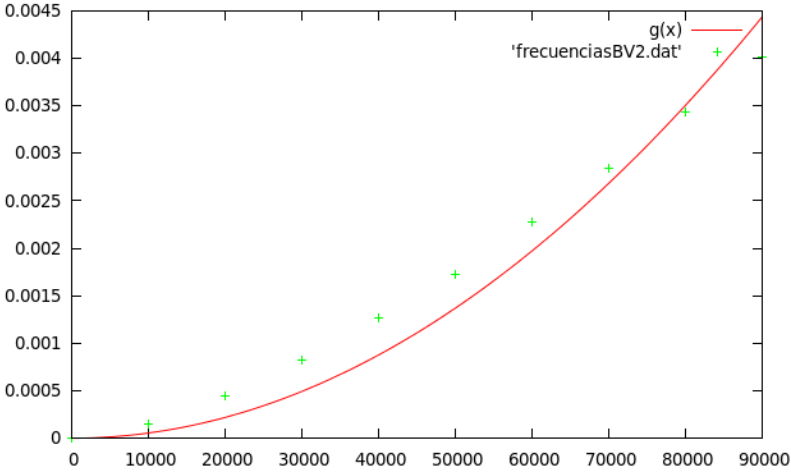
Realizar el análisis comparativo de eficiencia para los algoritmos del fichero frecuencia.cpp. En este caso, nuestro objetivo es ver cómo el uso de distintas estructuras de datos nos permite mejorar la eficiencia de nuestro algoritmo. En este caso, se utilizan distintos contenedores de la STL. En este apartado, el objetivo de la práctica es permitir al alumno ver las ventajas, en tiempo de ejecución, del uso de estructuras de datos más adecuadas, y no tanto el comprender el cómo funcionan los distintos algoritmos.

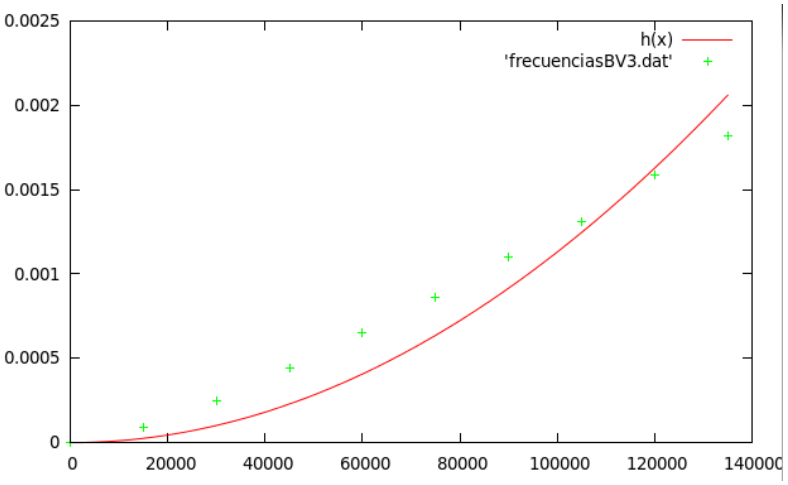
En el cpp, comentamos algunas líneas de cout para que cuando vayamos a crear el archivo con los datos nos salgan sólo los datos. Cambiamos el tamaño máximo de cada algoritmo y el tamaño en el que va incrementándose para acomodarlo a cada algoritmo y que tenga la precisión suficiente para que no salga 0. Por último, cambiamos la salida de los algoritmos para que salga como la de los anteriores ejercicios.

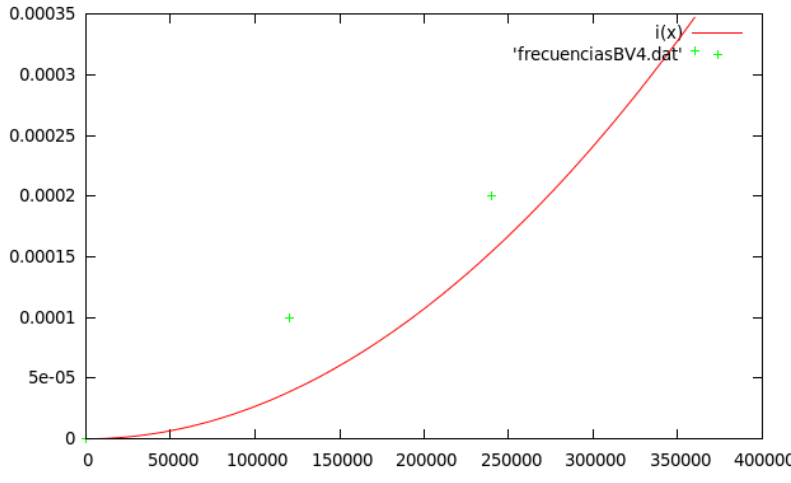
Se adjunta código cpp de los algoritmos así como los archivos .dat correspondientes a cada uno de ellos.

./frecuenciasV1 > frecuenciasBV1.dat
./frecuenciasV2 > frecuenciasBV2.dat
./frecuenciasV3 > frecuenciasBV3.dat
./frecuenciasV4 > frecuenciasBV4.dat

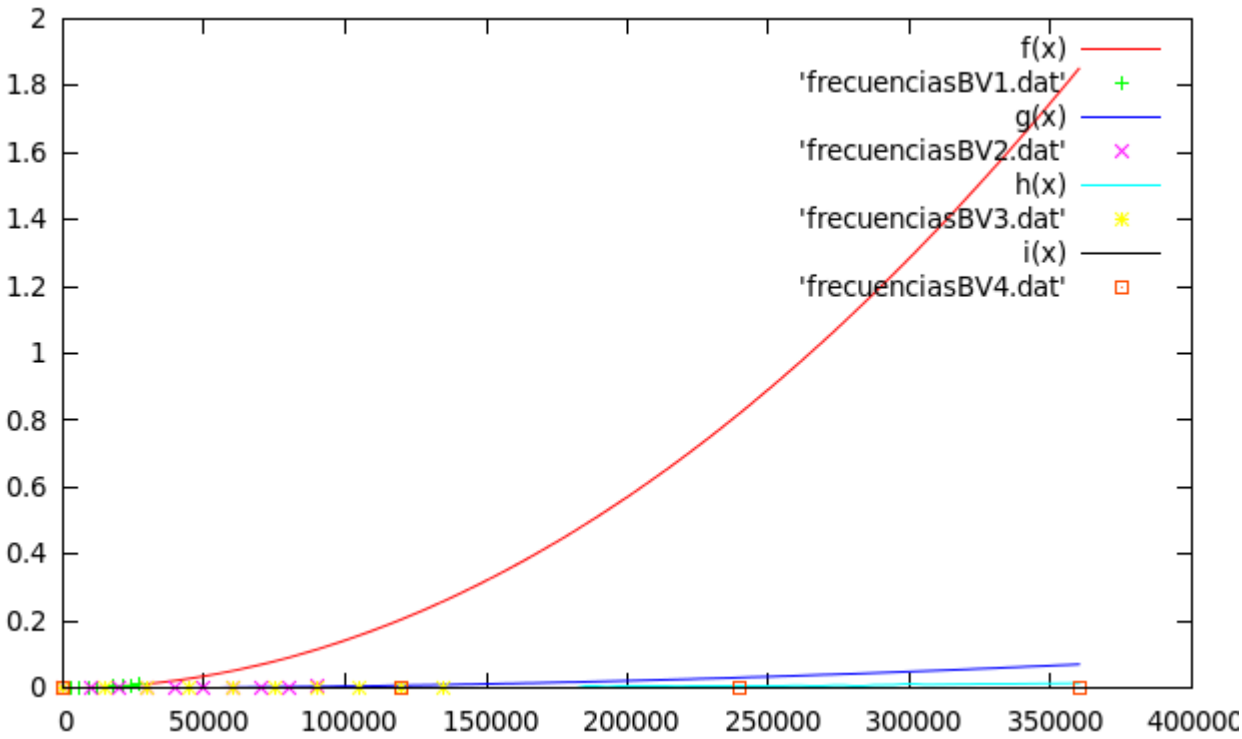








Aquí tenemos la gráfica comparativa de todos los algoritmos:



Luego el mejor algoritmo de frecuencia de todos es con diferencia el 4, mientras que el primero infinitamente menos eficiente que los demás

Resumen de equipos utilizados

Practica1.-

· Intel® Celeron® Processor 430 (512K Cache, 1.80 GHz, 800 MHz FSB) 2GB de RAM DDR2-800Mhz, torre.

· Compilador usado GNU GCC Compiler

· Sin opciones de compilación

Practica2.-

· Intel® Core™ i3 CPU 540 (512K Cache, 3,07Ghz, 3,06Ghz) 4GB de RAM DDR2-800Mhz, torre

· Compilador usado GNU GCC Compiler

· Sin opciones de compilación

Practica3.-

· 1 GB de RAM sobre Intel(R) Core(TM) i5-2300 CPU @2.80GHz 3.00 GHz y 4GB RAM DDR3, máquina virtual (VirtualBox)

· Compilador usado G++ Compiler

· Sin opciones de compilación