Algorítmica

Curso 2023-2024

Grupo Viterbi

PRÁCTICA 1-ANÁLISIS DE EFICIENCIA DE ALGORITMOS

Integrantes:

Miguel Ángel De la Vega Rodríguez	miguevrod@correo.ugr.es
Alberto De la Vera Sánchez	joaquin724@correo.ugr.es
Joaquín Avilés De la Fuente	adelaveras01@correo.ugr.es
Manuel Gomez Rubio	e.manuelgmez@go.ugr.es
Pablo Linari Perez	e.pablolinari@go.ugr.es

Facultad de Ciencias UGR
Escuela Técnica Ingeniería Informática UGR
Granada
2023-2024

Índice

Participación

- **Miguel Ángel De la Vega Rodríguez:** 20%
  - Plantilla y estructura del documento LaTeX
  - Cómputo de la eficiencia de los algoritmos (Resultados y Ajuste)
- **Joaquín Avilés De la Fuente:** 20%
  - Descripción del Objetivo de la práctica
  - Diseño del estudio
- **Alberto De la Vera Sánchez:** 20%
- **Manuel Gomez Rubio** 20%
- **Pablo Linari Pérez:** 20%
  - Estudio y comparación de las gráficas
  - Diseño del estudio

Equipo de trabajo

- **Miguel Ángel De la Vega Rodríguez:** (Ordenador donde se ha realizado el computo)
  - AMD Ryzen 7 2700X 8-Core
  - 16 GB RAM DDR4 3200 MHz
  - NVIDIA GeForce GTX 1660 Ti
  - 1 TB SSD NvMe

Objetivos

En esta práctica, se han implementado los siguientes algoritmos de ordenación: **quicksort, mergesort, heapsort, inserción, burbuja,** y **selección**. Además, se han implementado los algoritmos de **Floyd**, que calcula el costo del camino mínimo entre cada par de nodos de un grafo dirigido, de **Fibonacci**, que calcula los números de la sucesión de Fibonacci , y de **Hanoi**, que resuelve el famoso problema de las torres de Hanoi. Se ha aplicado la siguiente metodología:

- En primer lugar, aunque tenemos la eficiencia teórica de estos algoritmos, se realizarán los calculos necesarios para demostrar como se obtiene dicha eficiencia utilizando los distintos métodos estudiados en teoría.

- En segundo lugar, se pasará al estudio empírico de los algoritmos de ordenación de vectores para distintos tipos de datos, es decir, para datos tipo **int**, **float**, **double** y **string**. Posteriormente, se creará las gráficas para cada algoritmo en las que visualizaremos el tiempo de ejecución en función del tamaño del vector y del tipo de dato. Finalmente para esta parte, se hara un calculo de **eficiencia híbrida** que se basa en ajustar la gráfica obtenida a la función de su eficiencia teórica por mínimos cuadrados, obteniendo por tanto los literales de dicha función que ajustan la gráfica.

- En tercer lugar, se hará el estudio de los otros tres algoritmos de forma similar, es decir, se estudiará la eficiencia de estos de modo empírica, cuyo estudio se mostrará en las gráficas, y se calculará la eficiencia híbrida de estos, a partir de la eficiencia teórica.

Diseño del estudio

Los estudios empíricos han sido realizados en el ordenador con las características mencionadas anteriormente. Además, hemos realizado el estudio empírico de forma aislada para el algoritmo de ordenación de vectores quicksort en los distintos ordenadores de los participantes del grupo para ver como afectan las características hardware de cada ordenador en el tiempo de ejecución, cuyas gráficas se mostrarán en la sección de Algoritmos. En ambos casos se ha hecho uso del sistema operativo Linux, concretamente de Debian, y se ha utilizado el compilador gcc para la compilación de los programas con el flag -Og para la optimización.

4.1. Algoritmos de ordenación de vectores

Para los algoritmos de ordenación se han usado entradas de datos de tipo int, float, double y string mientras que para los algoritmos de Hanoi, Floyd y Fibonnaci solo se han usado entradas de tipo int ya que no tendría sentido usar entradas de otro tipo.

- En los algoritmos con eficiencia $O(n^2)$ como los de Burbuja, Selección e Inserción los saltos usados entre los tipos de datos int, float y double generados aleatoriamente son de 5000 en 5000 empezando con una muestra de 5000 datos y llegando a un máximo de 125000 datos.

- En los lagoritmos con eficiencia $O(\log(n))$ como el mergesort o el quicksort los saltos usados entre los tipos de datos int, float y double generados aleatoriamente son de 50000 en 50000 empezando con una muestra de 50000 datos y llegando a un máximo de 1250000 datos.

4.2. Otros algoritmos

En los algoritmos restantes se han usado datos de tipo int generados aleatoriamente y proporcionados en la siguiente medida:

- Para el algoritmo de Floyd que es de orden $O(n^3)$ se han usado enteros aleatorios desde 50 hasta 1250 con saltos de 50 en 50.

- Para el algoritmo de Fibonnaci que es de orden $O((\frac{1+\sqrt{5}}{2})^n)$ se han usado enteros aleatorios desde 50 hasta 1250 con saltos de 50 en 50.

- Para el algoritmo de Hanoi que es de orden $O(2^n)$ se han usado enteros aleatorios desde 3 hasta 33 con saltos de 50 en 50.

Por último para el tipo de dato string se han extraido las muestras del archivo *quijote.txt* para simular una generación aleatoria de palabras, esta entrada de datos no ha sido totalmente aleatoria ya que al usar un lenguaje determinado par el texto ,en este caso el español, se repiten con mas frecuencia algunas palabras por tanto esto se verá reflejado en el comportamiento de los algoritmos. En este caso el Quijote tiene un total de 202308 palabras por lo que se comenzará con una muestra de 12308 palabras con saltos de 10000 en 10000 hasta llegar a 202308 palabras.

4.3. Scripts usados para la ejecución

- **[AutoCompile.sh]** Este script se encarga de compilar todos los ficheros en una misma carpeta con las mismas opciones de compilación, para garantizar la máxima igualdad posible entre cada algoritmo y organizar la estructura de ficheros.
- **[AutoFinal.sh]** Este script es el encargado de ejecutar todos los algoritmos varias veces con las opciones respectivas para cada uno, el resultado se pasa por un programa AutoMedia.py que se encarga de realizar la media de las ejecuciones de los algoritmos, este resultado es guardado en una carpeta llamada Resultados de la que posteriormente el mismo script genera las graficas de cada algoritmo.
- **[AutoIndividual.sh]** Este script es como el descrito previamente pero unicamente ejecuta un script, esto ha sido útil para hacer pruebas sin la necesidad de esperar la gran cantidad de tiempo que requiere la ejecución de todos los algoritmos.

Algoritmos

Esta sección esta dedicada a mostrar los resultados obtenidos en el estudio de los algoritmos, la estructura seguida para mostrar los resultados consiste en mostrar, para cada algoritmo, los tiempos de ejecución, junto con las gráficas obtenidas y los ajustes correspondientes. Previo a ello, se analizará en cada caso teoricamente la eficiencia prevista para cada algoritmo.

5.1. Burbuja

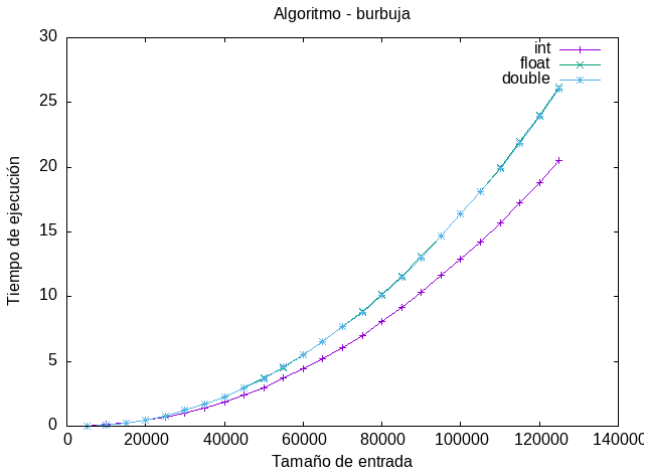

Figura 1: Ejecución algoritmo burbuja

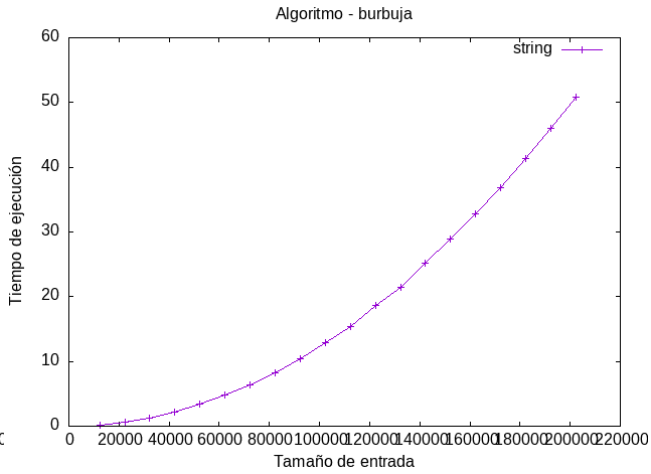

Figura 2: Ejecución algoritmo burbuja con string

5.2. Seleccion

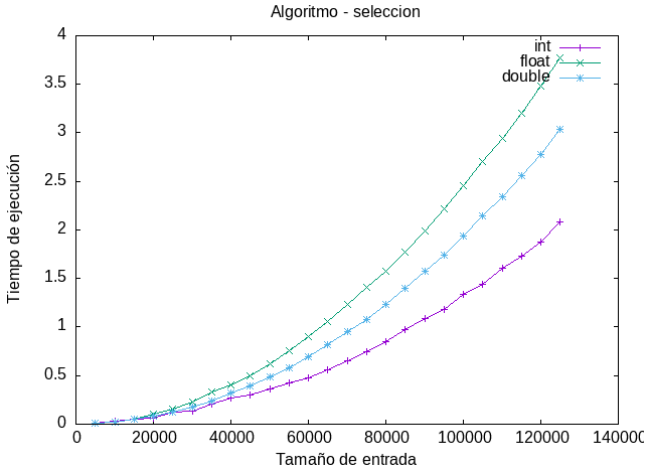

Figura 3: Ejecución algoritmo seleccion

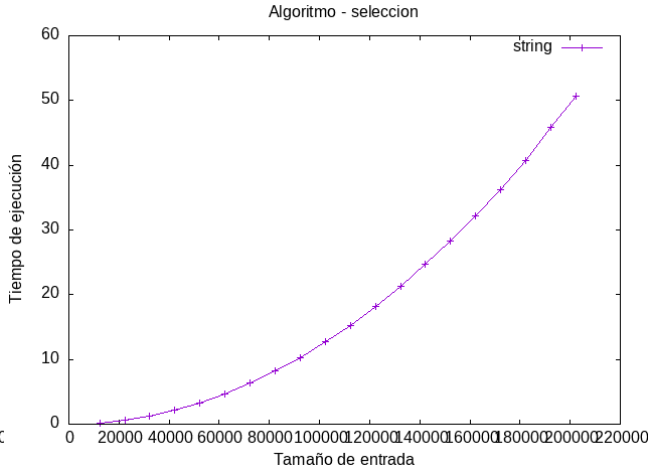

Figura 4: Ejecución algoritmo seleccion con string

5.3. Inserción

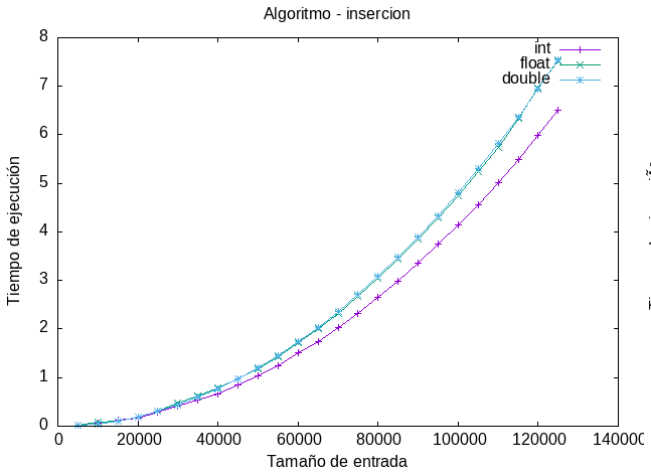

Figura 5: Ejecución algoritmo insercion

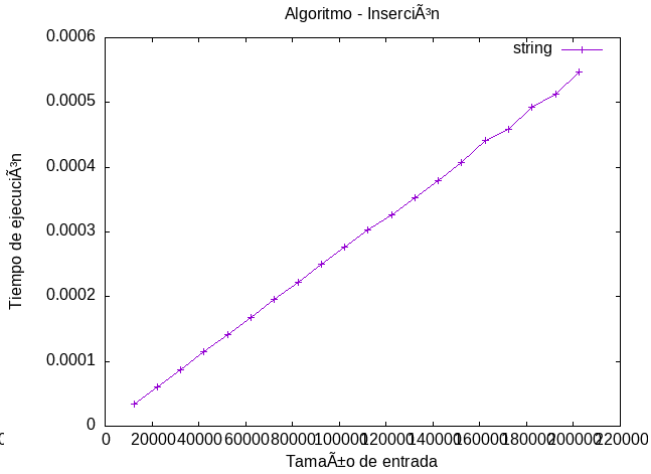

Figura 6: Ejecución algoritmo inserción con string

5.4. Hanoi

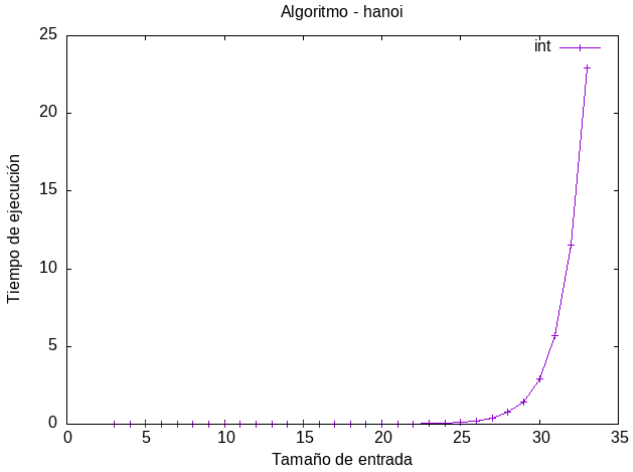

Figura 7: Ejecución algoritmo Hanoi

5.5. Fibonacci

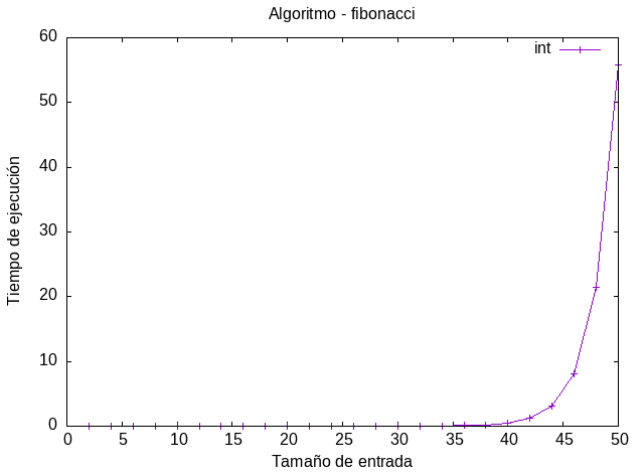

Figura 8: Ejecución algoritmo Fibonacci

5.6. Floyd

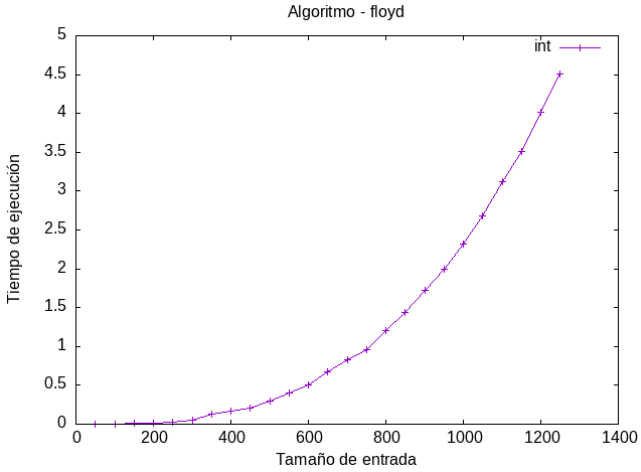

Figura 9: Ejecución algoritmo Floyd

Conclusiones