1º PRÁCTICA DE LA ASIGNATURA

ANÁLISIS Y DISEÑO DE ALGORITMOS
GRADO EN INGENIERÍA INFORMÁTICA (UVa)
CURSO 2019/2020

Índice.

Hernández Blanco, María Isabel
Hernando Brecht, Rebeca
Herruzo Herrero, Juan

Alumnos que realizan la práctica.

- Hernández Blanco, María Isabel.
- Hernando Brecht, Rebeca.
- Herruzo Herrero, Juan.

Realización de la práctica.

Introducción.

A continuación explicamos cómo hemos realizado la primera práctica de la asignatura de Análisis y Diseño de Algoritmos del curso 2019/2020 del grado de Ingeniería Informática.

Programas y entornos utilizados.

Para la realización de esta práctica hemos usado el lenguaje de programación *Java* en el entorno de desarrollo *Intellij IDEA Ultimate* y para el análisis de los resultados y redacción de este documento hemos usado los *documentos de google Drive*.

El código del programa.

En cuanto al código que creamos usamos el código del QuickSort que utilizamos en la asignatura EDA del curso 2018/19, para ello cambiamos la asignación del pivote para que se ajustara al enunciado, es decir: para que escoja como pivote la mediana de 3 números aleatorios del vector, y para el algoritmo de inserción hemos usado el código de Enrique García Hernández, (s.f).

Para contar el número de asignaciones y comparaciones hemos declarado dos variables globales como contadores: *NUM_ASIGNACIONES* y *NUM_COMPARACIONES,* y para calcular el umbral hemos creado un bucle que rellena un vector con números aleatorios y realiza las ordenaciones 20 veces con cada tamaño para, después, calcular el promedio de los resultados obtenidos. Una vez haya calculado este promedio, el número del vector aumenta y se rellena con otros números para volver a repetir la ordenación. El tamaño del vector va aumentando dependiendo en qué intervalo se encuentre: si el tamaño del vector pertenece al intervalo [3, 103] aumenta de 5 en 5, si pertenece al [10 000, 100 000], lo hace de 10 000 en 10 000 y si se encuentra en [100 000, 1 000 000], aumenta en 100 000, estos saltos se deben a que no queríamos sobrecargar computacionalmente al ordenador. Todos los datos obtenidos en estas iteraciones se guardan en ficheros *.csv* y para que estos ficheros estén ordenados y sea más fácil localizarlos decidimos que en cada iteración se creara una carpeta llamada *umbralN*, donde *N* es tamaño del vector con el cual *QuickSort* va a parar de ordenar. Dentro de estas carpetas, *umbralN*, hay un fichero *promedio.csv*, donde están los datos para calcular

el promedio de ese umbral, y otra carpeta llamada *individuales* donde se encuentran 20 ficheros *.csv* con los datos individuales de cada ordenación en esa iteración. Todos estos archivos siguen la misma estructura: *tamañoDelVector, númeroDeAsignaciones, númeroDeComparaciones, tiempo.*

Decidimos que el algoritmo cambiara de *QuickSort* a *Inserción* cuando el tamaño del vector sea el mismo que el del umbral por lo que la primera ejecución del programa, umbral 1, nos devuelve los valores de *Quicksort* sin *inserción*.

Por último, añadimos una función que borrar los ficheros de la ejecución anterior automáticamente.

Rangos, repetición de medidas y umbrales.

Para escoger los rangos con los que comparamos los distintos vectores y así poder sacar el promedio y con ello el umbral nos hemos basado en los indicados en la práctica. Finalmente nos decidimos por los siguientes intervalos: [3, 103] para comparar vectores pequeños, [10 000, 100 000] y [100 000, 1 000 000] para comparar vectores grandes. Esta decisión se debe a que la distancia de los intervalos es múltiplo de la distancia de salto indicada en el apartado anterior (5, 10 000 y 100 000) por lo que conseguimos comparar los extremos de los intervalos.

Respecto a la repetición de medidas nos decidimos por iterar los bucles 20 veces, ya que es un número lo suficientemente grande como para que pueda ocultar posibles anomalías, como que caigamos en el peor caso de Quicksort en un momento determinado, pero no es tan grande como para el ordenador tenga una sobrecarga computacional. Además nos ha parecido conveniente ya que es uno de los ejemplos propuestos en el enunciado de la práctica.

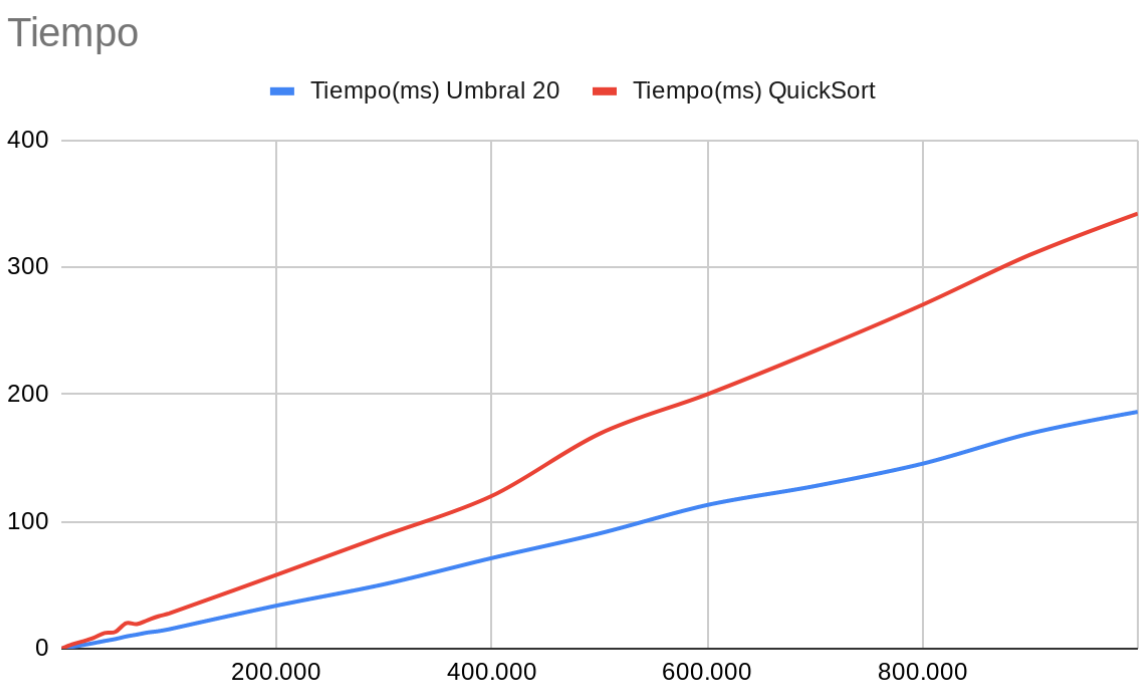
Decidimos sacar desde el umbral 1 al 30 debido a que, como hemos explicado antes, el primero nos da el algoritmo de ordenación de *Quicksort* puro y, aunque comparar 31 umbrales diferentes nos parecía un poco excesivo, al ejecutarlo y analizarlo por primera vez con 21 distintos, vimos que el 20, el extremo superior, era reseñable por ser el más rápido así que decidimos aumentar el número de umbrales para estudiar los superiores a 20.

Conclusiones del análisis de la práctica.

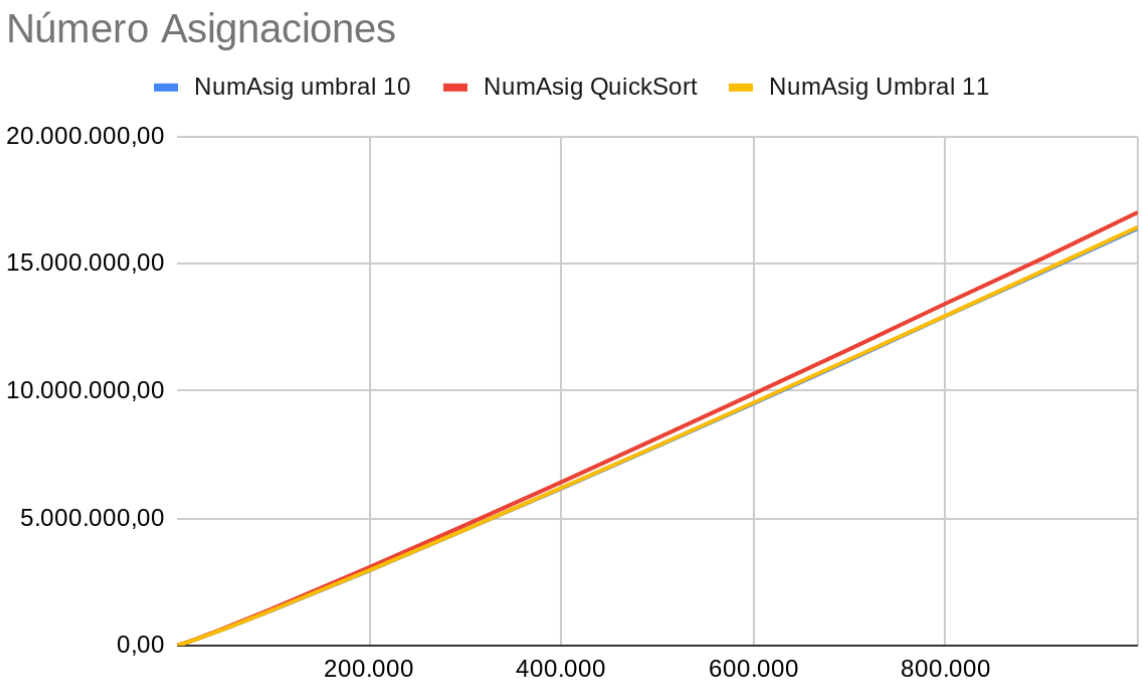
En primer lugar comparamos todas las gráficas y los valores numéricos de los umbrales obtenidos con la gráfica y los valores del *Quicksort* sin *inserción* después de ejecutar el código. Vemos claramente que el umbral 10 es el que menos asignaciones realiza, el umbral 11 es el más óptimo en cuanto a comparaciones y el umbral 20 es el que menos tiempo de ejecución conlleva. Descartamos esta última gráfica ya que como el tiempo cambia dependiendo de la computadora en la que se ejecute el código y el estado de dicha computadora, es la variable menos fiable de las 3 que analizamos. En este programa en concreto también afecta al tiempo las llamadas recursivas del algoritmo de *Quicksort*, como con el umbral 20 se llama rápidamente al algoritmo de *inserción* no realiza tantas llamadas recursivas, motivo por el que el tiempo es mucho menor.

Hernández Blanco, María Isabel
Hernando Brecht, Rebeca
Herruzo Herrero, Juan

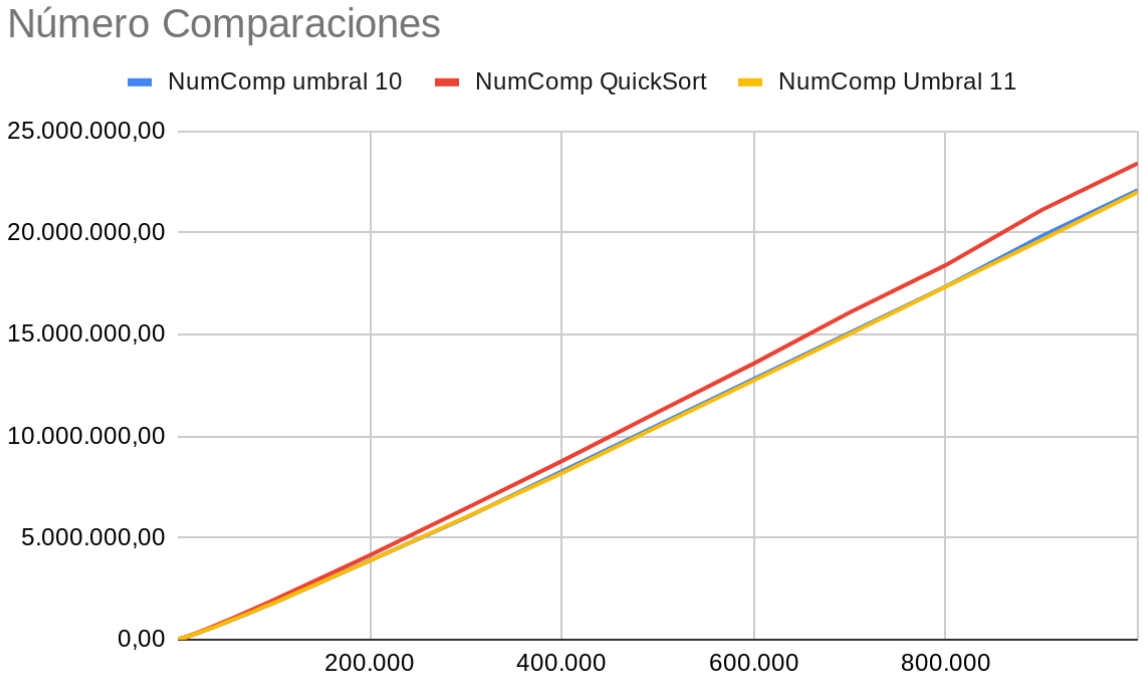
Además a partir del umbral 20 en adelante los tiempos empiezan a fluctuar entre 180ms y 190ms con vectores de 1 millón.



Tras este primer análisis nos centramos en las asignaciones y comparaciones de los umbrales 10 y 11 por lo que estudiamos estas dos variables y las contrastamos analizando las siguientes las gráficas:



Hernández Blanco, María Isabel
Hernando Brecht, Rebeca
Herruzo Herrero, Juan



Observando estas gráficas apreciamos que la oscilación de los valores de las 2 variables es, a simple vista, casi imperceptible y por tanto optamos por agrupar los valores de los umbrales 10 y 11 de forma que obtenemos esta tabla:

Umbral	Comparaciones	Asignaciones
10	22 095 205,70	16 387 355,75
11	22 011 092,45	16 438 142,15

Datos tomados con vectores de tamaño 1 millón

A la vista de estos valores nos damos cuenta que las comparaciones promedio entre los umbrales varían en aproximadamente 84 000 y las asignaciones promedio en 150 000, número bastante significativo, por lo que decidimos quedarnos con el 10 como el umbral óptimo ya que es el que menos asignaciones tiene y, a pesar de que tiene más comparaciones, nos parece una cantidad más asumible que las 150 000 comparaciones de más que hace el otro umbral.

Comparamos las gráficas este umbral óptimo con el *Quicksort puro* y, como se ve a continuación, llegamos a la conclusión de que el *Quicksort puro* en todo el intervalo [3 , 1 000 000 000] es menos eficiente en cuanto a asignaciones y comparaciones sin importar el tamaño del vector.

Hernández Blanco, María Isabel
Hernando Brecht, Rebeca
Herruzo Herrero, Juan

Análisis teórico.

Partiendo de la base de que el caso promedio de Quicksort pertenece al orden n*log(n) nos aseguramos de que las gráficas pertenecen a ese orden y además obtenemos las constantes de proporcionalidad con el siguiente procedimiento de análisis:

1. Calculamos n log(n) siendo la entrada n el Tamaño del vector de entrada (TamArray).

TamArray	n log(n)
3	1,43
8	7,22
13	14,48
...	...
98	195,14
103	207,32
10.000	40.000,00
20.000	86.020,60
...	...
900.000	5.358.818,26
1.000.000	6.000.000,00

Hernández Blanco, María Isabel
Hernando Brecht, Rebeca
Herruzo Herrero, Juan

2. Dividimos el número de operaciones elementales por el n log(n) calculado anteriormente para sacar la proporcionalidad entre ellas para cada tamaño de vector.

NumAsig QuickSort	n log(n)	Constante de proporcionalidad de asignaciones (CA)
9,75	1,43	6,81
37,35	7,22	5,17
72,15	14,48	4,98
...	...	...
746,70	195,14	3,83
774,00	207,32	3,73
122.731,35	40.000,00	3,07
259.819,35	86.020,60	3,02
...	...	...
13.436.541,90	4.722.471,99	2,85
15.190.010,10	5.358.818,26	2,83
17.017.544,85	6.000.000,00	2,84

3. Sacamos la constante de proporcionalidad teniendo en cuenta el apartado 2.
Viendo la tabla de las proporciones nos damos cuenta de que, a medida que crece el vector, las proporcionalidades se van aproximando al mismo número. Por lo que decidimos coger como constante de proporcionalidad el valor que obtenemos con vectores de 1 millón, es decir CA = 2,84.

4. Multiplicamos n log(n) por la constante de proporcionalidad para cada tamaño del vector.

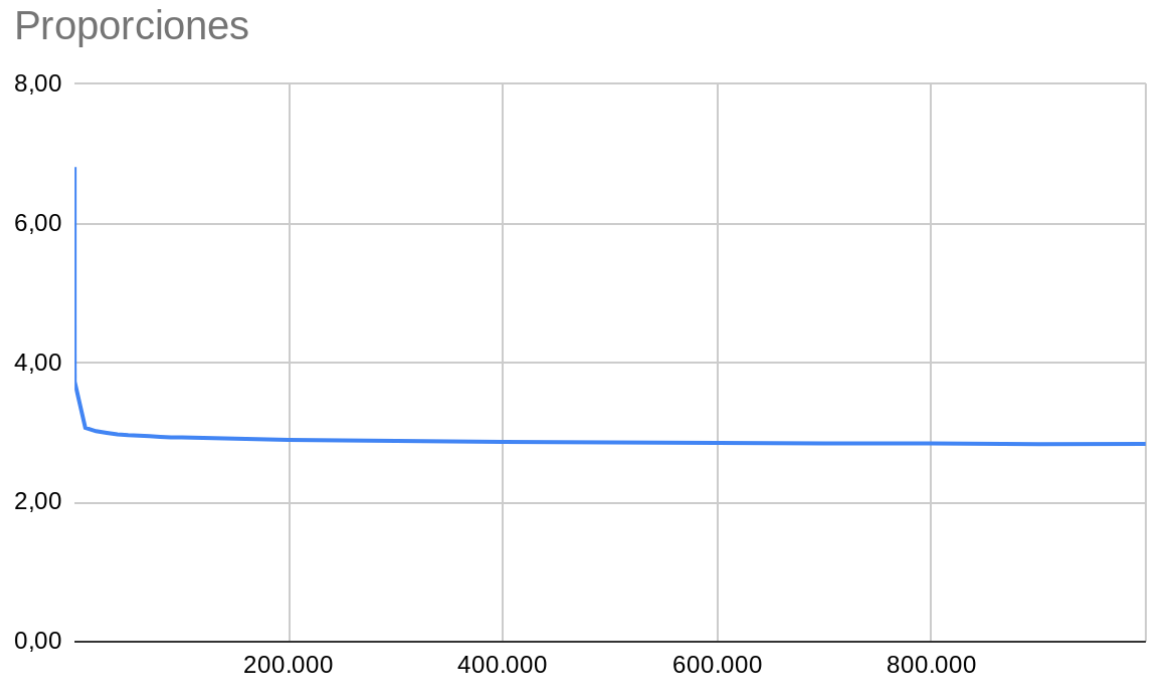
n log(n)	Constante de proporcionalidad de asignaciones (CA)	CA * n log(n) Normal
1,43	6,81	4,06
7,22	5,17	20,48
14,48	4,98	41,05
...	...	...
195,14	3,83	553,14
207,32	3,73	587,67
40.000,00	3,07	113.383,28
86.020,60	3,02	243.832,45
...	....	...
5.358.818,26	2,83	15.190.010,10
6.000.000,00	2,84	17.007.492,36

Hernández Blanco, María Isabel
Hernando Brecht, Rebeca
Herruzo Herrero, Juan

5. Comparamos los datos obtenidos teóricamente con los experimentales.

NumAsig QuickSort	CA * n log(n) Normal
9,75	4,06
37,35	20,48
72,15	41,05
...	...
746,70	553,14
774,00	587,67
122.731,35	113.383,28
259.819,35	243.832,45
...	...
15.190.010,10	15.190.010,10
17.017.544,85	17.007.492,36

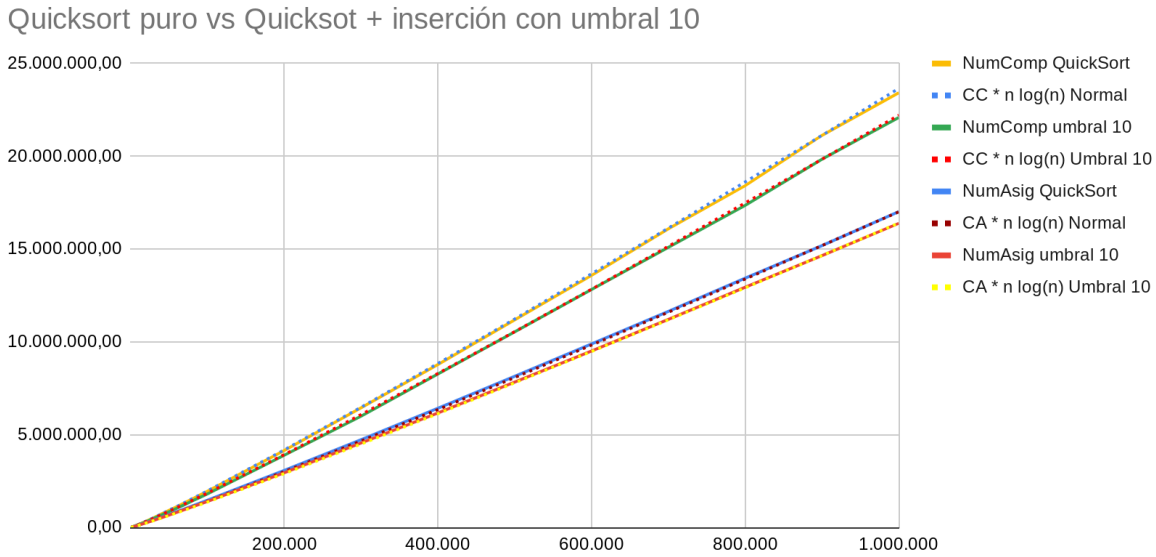
A la vista de esta de esta tabla podemos concluir que cuanto más pequeño es el vector menos exacto es el valor teórico. Esto se debe a que la CA tiende muy rápidamente de un valor grande como puede ser 6,81 (proporción de los vectores de tamaño 3) a un número más pequeño como 2,84, valor de proporción de los vectores de tamaño 1 millón, el cual hemos usado CA, por lo que cuanto más se acerque el tamaño del vector a 1 millón más exacto será nuestro valor teórico.



Hernández Blanco, María Isabel
Hernando Brecht, Rebeca
Herruzo Herrero, Juan

El proceso anterior ha sido usado para obtener los datos comparados con el número de asignaciones de *Quicksort puro,* este proceso lo hacemos 3 veces más para sacar el resto de constantes.

Una vez tenemos todas las constantes las comparamos con los datos experimentales y obtenemos la siguiente gráfica:



Como podemos ver en la gráfica, los valores teóricos son muy similares a los experimentales, por lo que podemos concluir que todos los algoritmos siguen un orden n log(n) con las siguientes constantes de proporcionalidad:

- CA Quicksort puro = 2,84
- CA Quicksort híbrido con umbral 10 = 2,73
- CC Quicksort puro = 3,90
- CC Quicksort híbrido con umbral 10 = 3,68

Con estos valores corroboramos que el *Quicksort híbrido* es mejor que el *Quicksort puro*.

Hernández Blanco, María Isabel
Hernando Brecht, Rebeca
Herruzo Herrero, Juan

Bibliografía.

- García Hernández, Enrique (s.f), Ordenación por inserción, Programación Java: http://puntocomnoesunlenguaje.blogspot.com/2015/02/ordenamiento-insercion-directa-java.html
- Baeldung (December 20, 2018), Delete a Directory Recursively in Java, Baeldung: https://www.baeldung.com/java-delete-directory
- Create whole path automatically when writing to a new file, stackoverflow: https://stackoverflow.com/questions/2833853/create-whole-path-automatically-when-writing-to-a-new-file
- C. Vaca, C. González (2017/18), Tema 1 - Análisis de Algoritmos: Notación asintótica.