2º 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.

Ficheros entregados.

Junto a este informe entregamos los siguientes ficheros:

- bruteForce.java
- dinamico.java
- recursividad.java

Estos tres ficheros contienen el código de los algoritmos que analizan el fichero “entrada.txt” y devuelven la solución en la salida estándar como indica el enunciado de la práctica.
Además enviamos una carpeta “estudio” con los ficheros:

- bruteForce.java
- dinamico.java
- recursicidad.java
- generator.java
- generatorTestEstandar.java

Estas clases están modificadas para que cuenten el número de accesos al vector, el número de comparaciones y el tiempo que tarda en ejecutarse cada algoritmo. Además de esto lee de forma automática todos los ficheros de una carpeta dada. Modificamos estas clases para que leyeran los ficheros con casos aleatorios, creados por la clase *generator* y *generatorTestEstandar* , y nos devolvieran los datos con los que realizamos el estudio práctico.

Realización de la práctica.

Introducción.

A continuación explicamos cómo hemos realizado la segundo 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*.

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

Descripción del código.

El código describe 3 algoritmos diferentes para resolver el problema indicado en el enunciado de la practíca.

El primero de ellos es el algoritmo de fuerza bruta que busca secuencialmente el mínimo y el máximo con la restricciones que indica el enunciado de la práctica. Este algoritmo está optimizado ya que si el número que estás analizando es mayor que el mínimo absoluto actual, no analiza las diferencias, pasa al siguiente número del array.

El segundo implementa divide y vencerás donde el vector se divide hasta que solo hay un elemento este valor pasa a ser el mínimo y máximo relativo y mínimo y máximo absoluto. Se aplica recursividad y se van comparando y cambiando estas 4 variables según corresponda: comparamos las diferencias entre máximos y mínimos absolutos, si estas son menores que las diferencias entre máx y min relativo los valores no cambian las variables pero si es al revés, se cambian los max y min absolutos por los que correspondan.

El último muestra otra forma de resolver el problema usando programación dinámica. El algoritmo almacena el índice al valor del mínimo absoluto del vector, del mejor momento para comprar, desde el principio hasta el momento actual, (minDif) y del mejor momento para vender (maxDif). Por cada iteración se comprueba qué diferencia es mejor: la de los elementos en las posiciones de maxDif y minDif o el elemento actual menos el mínimo absoluto del vector. Si la primera diferencia es más pequeña que la segunda se actualizan estos valores, es decir el mejor momento para comprar y vender es la actual. Si el elemento actual es menor que el mínimo guardo se actualiza dicho mínimo.

Algoritmos escogidos.

Hemos decidido escoger los 3 tipos de algoritmos explicados en el apartado anterior porque pensando distintas soluciones al problema, fueron las 3 posibilidades que se nos ocurrieron y se adaptaban a los patrones explicados en clase.

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

Análisis de la eficiencia de los algoritmos.

A continuación analizamos las conclusiones obtenidas a partir del análisis del rendimiento de los algoritmos de forma práctica y de forma teórica.

Para ambos estudios decidimos analizar el número de accesos al vector principal y la operación de comparar en todos los algoritmos, además del tiempo de ejecución. Aunque es interesante estudiar el tiempo de ejecución de los algoritmos, es una variable que depende de más factores independientes de nuestro código, por lo que para tomar las decisiones finales no nos basamos en los valores de esta variable y en el rendimiento teórico no la estudiamos.

Rendimiento teórico.

Nuestros algoritmos tienen una parte común: abren el fichero que contiene los casos a estudiar, lo lee y guarda los datos en el vector principal de la misma forma. Por esto, para hacer el estudio teórico, obviamos esta parte y analizamos el código que diferencia a estos algoritmos.

Fuerza Bruta.

Al no tratarse de un algoritmo de fuerza bruta estándar ya que está optimizando, nos encontramos con un mejor, peor y caso promedio.
El peor caso sería un vector ordenado de forma ascendente, esto haría fallar siempre a la optimización por lo que se trataría siempre de un bucle i=0 hasta n y un segundo bucle anidado j=i hasta n, nos encontramos con un orden de n^2.
El mejor caso sería un vector con su mínimo absoluto en su primera posición, esto activaría siempre la optimización y tendríamos sólo un bucle i=0 hasta n, por lo que se obtiene un orden de n.

Para el caso promedio hay que estudiar la probabilidad de que se utilice la optimización, es decir de que el valor en la posición i no es el mínimo absoluto del vector[0..i].
La probabilidad de que el número i sea el menor del vector[0..i] es de 1/i, si ocurre esto caemos en el peor caso, es decir $O(n^2)$.
La probabilidad de que el número i no sea el menor del vector[0..i] es de (i-1)/i, si ocurre esto caemos en el mejor caso, es decir O(n).
Entonces el caso promedio sería:

$$\sum_{i=0}^{n} \left(\frac{i-1}{i} + \frac{1}{i} \sum_{j=i}^{n} 1 \right)$$

El orden de este sumatorio es $\Theta(n \times log\ n)$.
Por lo tanto en el caso promedio el orden es de $n \times log\ n$.

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

Recursividad.

Es independiente de la distribución del vector de entrada, es decir no tiene un mejor o peor caso, siempre tiene el mismo orden.
Este algoritmo puede ser analizado con el teorema maestro.

$T(n) = 2 \times T(\frac{n}{2}) + f(n^0)$

Como a=2 b=2 k=0 entonces tenemos el siguiente orden asintótico Θ(n $\log_b$a) que es igual a Θ(n).

Dinámico.

Es independiente de la distribución del vector de entrada, es decir no tiene un mejor o peor caso, siempre tiene el mismo orden.
En este algoritmo solo hay que recorrer una vez el vector gracias a almacenar datos en memoria, por lo tanto al tratarse de solo un bucle tiene un orden de n.

Debido a este estudio cabe esperar que a la hora de ejecutar los algoritmos y estudiar las variables, el algoritmo dinámico sea el que menos recursos gaste, seguido del recursivo y el de fuerza bruta, siendo este último el que más comparaciones y accesos al vector haga.

Resumen

	Orden estudiado
Fuerza bruta con optimización	Peor caso: $\Theta(n^2)$ Mejor caso: $\Theta(n)$ Caso promedio: $\Theta(n \times log(n))$
Dinámico	$\Theta(n)$
Recursividad	$\Theta(n)$
Gasto de recursos	dinámico < recursivo < fuerza bruta

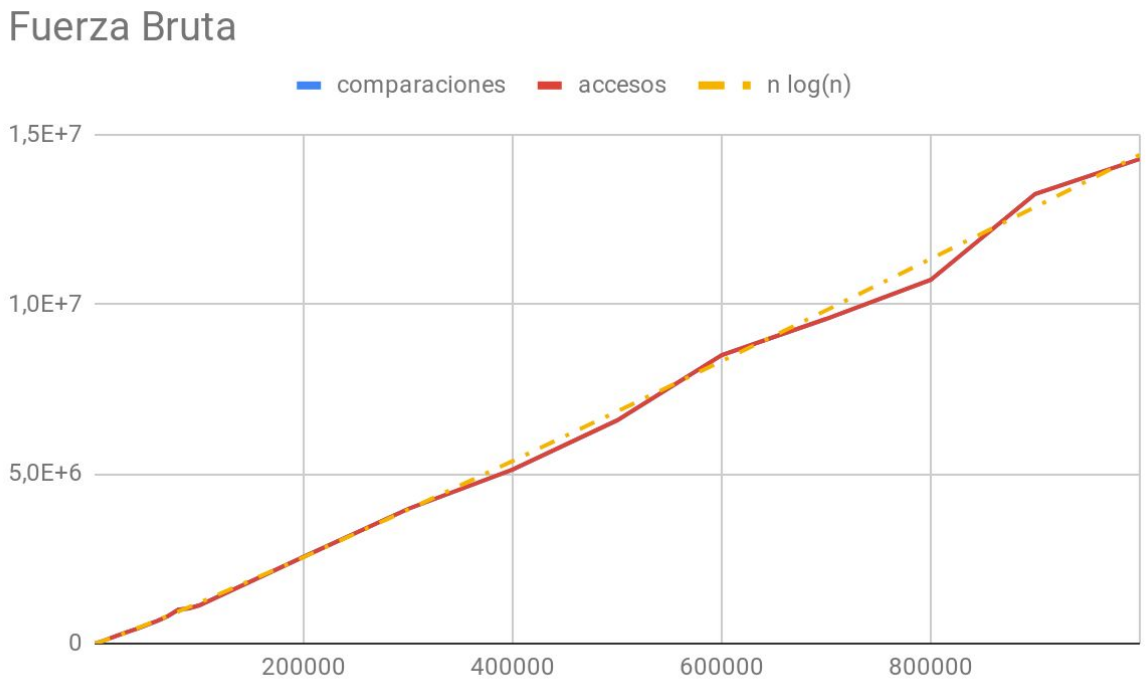
Hernández Blanco, María Isabel
Hernando Brecht, Rebeca
Herruzo Herrero, Juan

Rendimiento práctico.

En el primer estudio que realizamos con el fichero de ejemplo que se nos facilitaba en aulas vimos que el tiempo de ejecución era mucho mayor que el de las pruebas realizadas con los vectores creados aleatoriamente y vimos que el parsear de string a int era lo que más ralentizaba el código. Cambiamos la clase FileReader por la de Scanner y vimos una mejora bastante notable en el tiempo de ejecución. Anteriormente estimábamos que iba a tardar varias horas, no le dejamos acabar al programa debido a la lentitud, y después del cambio resolvió el problema en pocos segundos.

Fuerza bruta.

Para sacar la pendiente del polinomio calculamos el coeficiente de proporcionalidad del número de comparaciones o accesos al vector, en este caso da igual cual usemos porque es el mismo, de cada tamaño del vector, es decir, dividimos el número de comparaciones para el tamaño de vector n entre el $log(n)$, sumamos todos y obtenemos que la pendiente es 2,4 por tanto el polinomio interpolador es $f(x) = 2,4 \times n \times log(n)$.



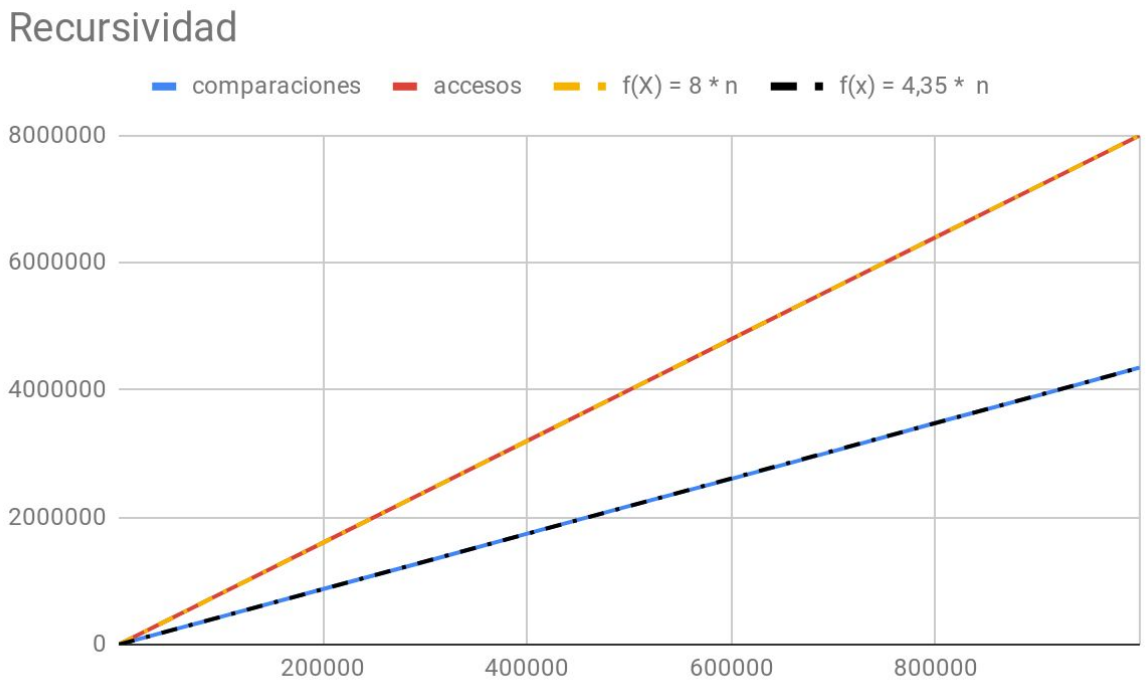
Respecto al tiempo, para un vector de 1.000.000 elementos el algoritmo tarda 8,72 milisegundos y para el archivo “entrada.txt” dado, usando el comando *time*, tarda 6,045ss

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

Recursividad.

En el caso del algoritmo de recursividad no hay nada destacable. A la vista de los datos se observa fácilmente que ambas variables sí que sigue un orden de $\Theta(n)$, como se estimó en el estudio teórico.

Hemos concretado que el número de accesos se interpola con el polinomio $f(x) = 8 \times n$ y el número de comparaciones con $f(x) = 4,35 \times n$.



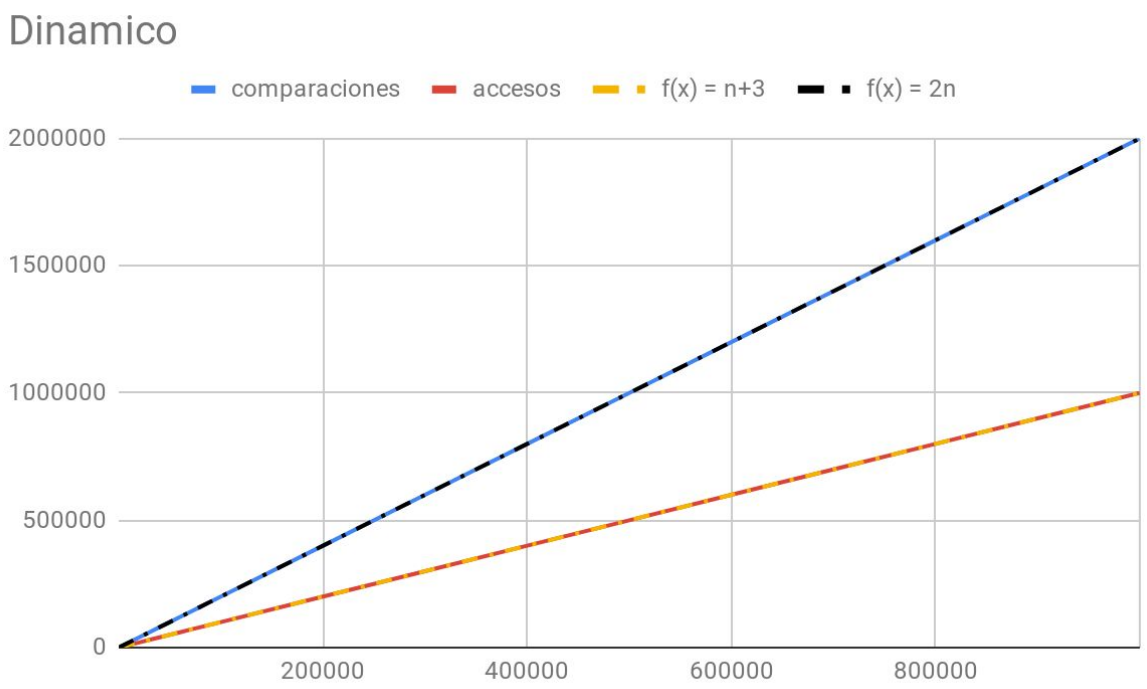
Respecto al tiempo, para un vector de 1.000.000 elementos el algoritmo tarda 17,03 milisegundos y para el archivo “entrada.txt” dado, usando el comando *time,* tarda 6,238s . Aumenta respecto a fuerza bruta debido a las llamadas recursivas que este algoritmo hace.

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

Dinámico.

Para el algoritmo dinámico observamos que también coincide los dos estudios y que los datos muestran que es cierto que sigue una distribución $\Theta(n)$.
Este algoritmo está optimizado en cuanto al número de accesos al vector ya que antes del bucle for se inicializan las variables sobre las que se van a ir trabajando. Esto hace que en el bucle sólo tenga que acceder al elemento del vector de la iteración correspondiente. Así conseguimos reducir los accesos un 300% y ahora el número de acceso al vector es igual al tamaño del vector + 3.

Por tanto, el polinomio que interpola al número de comparaciones es $f(x) = 2 \times n$ y al número de accesos al vector $f(x) = n + 3$ como se ve en la gŕafica.



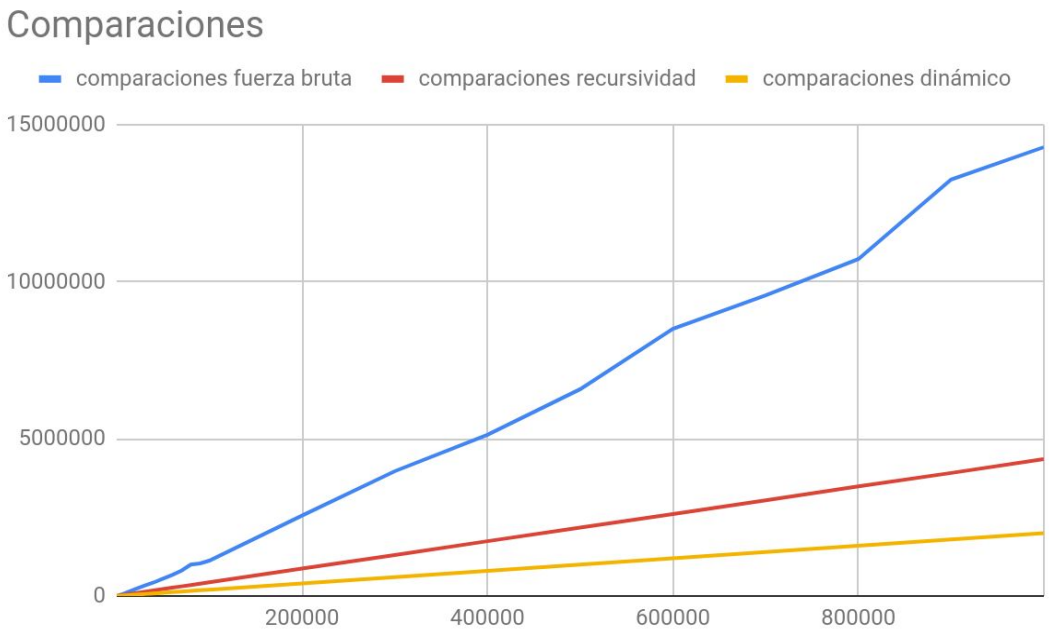
Respecto al tiempo, para un vector de 1.000.000 elementos el algoritmo tarda 1,53 milisegundos y para el archivo “entrada.txt” dado, usando el comando time, tarda 5,732s

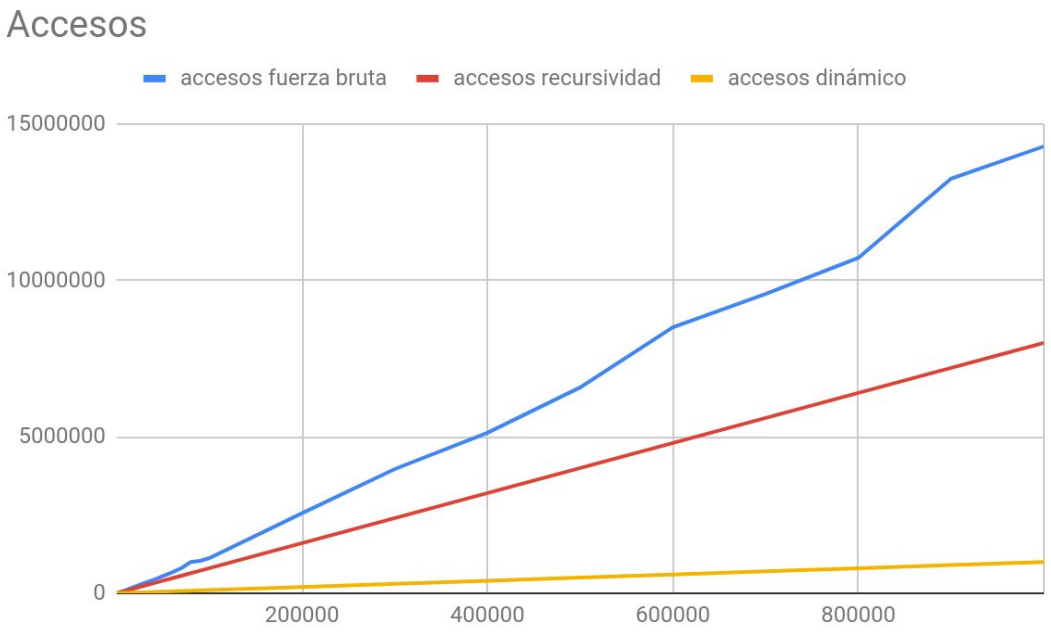
Hernández Blanco, María Isabel
Hernando Brecht, Rebeca
Herruzo Herrero, Juan

Resumen.

	Polinomio interpolador para las comparaciones	**Polinomio interpolador para los accesos**
Fuerza bruta	$f(x) = 2,4 \times n \times log\ n$	$f(x) = 2,4 \times n \times log\ n$
Dinámico	$f(x) = 2 \times n$	$f(x) = n + 3$
Recursividad	$f(x) = 4,35 \times n$	$f(x) = 8 \times n$

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





Conclusión.

Examinado el estudio teórico vemos que tenemos 2 algoritmos, dinámico y recursividad, con orden $\Theta(n)$ que superan en rendimiento al algoritmo de recursividad con orden $\Theta(n \times log(n))$. Después de realizar el estudio práctico y de calcular los polinomios interpoladores vemos que el algoritmo dinámico se interpola con $f(x) = 2 \times n$ para el número de comparaciones y con $f(x) = n + 3$ para el número de accesos al vector mientras que el algoritmo de recursividad se interpola con $f(x) = 4,35 \times n$ y $f(x) = 8 \times n$

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

respectivamente. Comparamos sus pendientes, $2 < 4,35$ y $1 < 8$, y concluimos que nuestro algoritmo dinámico es el más óptimo de los 3.

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

Bibliografía.

- Oracle Java Platform SE 8, Class Files: https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html
- Oracle Java Platform SE 8, Class Buffer: https://docs.oracle.com/javase/8/docs/api/java/nio/Buffer.html
- Oracle Java Platform SE 8, Class Scanner: https://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html
- Wolfram, 2019. Wolfram, WolframAlpha computational intelligence: https://www.wolframalpha.com/