EXAMEN DE PRÁCTICAS

FAA – 1º Ingeniería Informática

Modificación añadiendo el método QuickSort

Alba Márquez Rodríguez

3. Introducción. Algoritmos de Ordenación.

En esta modificación, a partir de la práctica 2 implementaremos un nuevo método para analizar: QuickSort. Con su estudio empírico correspondiente. Por separado y comprándolo con los ya implementados.

4. Cálculo de los tiempos teóricos:

Aplicando las reglas estudiadas en clase calcularemos el número de operaciones elementales del algoritmo. Lo pondremos como una función de T(n). Siendo n las veces que se ejecutará el bucle. Al depender el tiempo de ejecución de la talla, n=talla

4.1. Pseudócódigos y análisis de coste

Este es el pseudocódigo que se nos da en el enunciado del ejercicio, completamos con las operaciones elementales (asignaciones, accesos…).

QuickSort

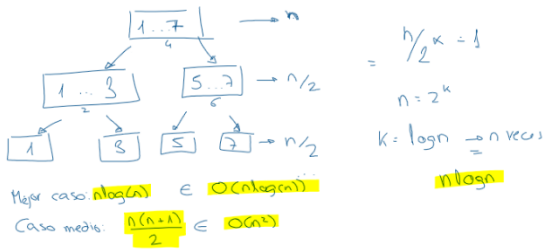
Calculamos y tenemos en cuenta que para el mejor caso el si no se ejecuta así que sólo se tendrían en cuenta la condicional mientras que en el caso peor sí. Todos los casos son O(n^2) El caso medio es:

Quicksort: algoritmo.

```
QuickSort(A,p,r)
  if (p < r) {
      q = Partition(A, p, r)
      QuickSort(A, p, q)
      QuickSort(A, q+1, r)
      }
```

Llamada inicial:
QuickSort(A, 0, n-1);
n: número de elementos del vector A

```
Partition(A, p, r)
  piv = A[p]; i = p − 1; j = r + 1;
  do{
    do{
      j = j − 1;
    }while (A[j] > piv)
    do{
      i = i + 1;
    }while (A[i] < piv)
    if (i < j) {
      A[i] ⟷ A[j]; //Intercambiar valores
    }
  }while (i < j)
  return j;
```



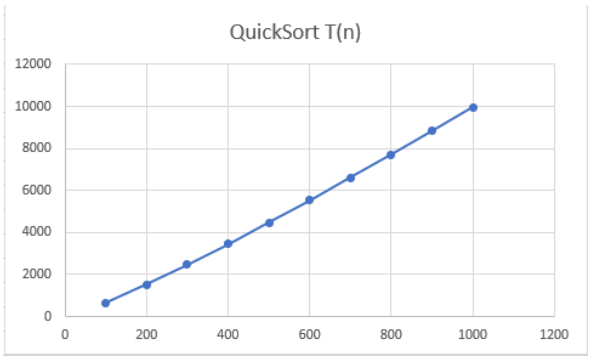
El mejor caso será en el que ya está ordenado y se hacen particiones proporcionales. Así no hace ningún intercambio. En el caso peor las particiones no serán nada proporcionales, una muy grande y otra muy pequeña. El caso medio será nlog(n).

De estos datos podemos comparar las fórmulas con los algoritmos analizados en la práctica y suponer los resultados (cuáles serán más/menos eficiente). Por el orden de complejidad el más eficiente será QuickSort. Los otros son de orden cuadrático y este es de orden logarítmico que es menor.

4.2. Tablas (ficheros) y Gráficas de coste

QuickSort

Talla	QuickSort T(n)
100	664,385619
200	1528,771238
300	2468,645607
400	3457,542476
500	4482,892142
600	5537,291214
700	6615,847778
800	7715,084952
900	8832,403072
1000	9965,784285



4.3. Conclusiones

Talla	Burbuja T(n)	nserción T(n	elección T(n	uickSort T(n
100	62573,5	28416	40312	664,385619
200	250148,5	111841	155637	1528,77124
300	562723,5	250266	345962	2468,64561
400	1000298,5	443691	611287	3457,54248
500	1562873,5	692116	951612	4482,89214
600	2250448,5	995541	1366937	5537,29121
700	3063		1857262	6615,84778
800	4000	1767391	2422587	7715,08495
900	5063173,5	2235816	3062912	8832,40307
1000	6250748,5	2759241	3778237	9965,78428

Los tres algoritmos ya implementados tienen crecimientos cuadráticos en sus casos medios y el nuevo logarítmico. Tras los cálculos y comparación de las fórmulas obtenidas intuimos que el más eficiente será QuickSort con diferencia y después los demás que son del mismo orden. La gráfica de QuickSort se ve como una línea recta por la diferencia.

Tras realizar tablas en Excel observamos que estábamos en lo cierto, todas tienen crecimiento cuadrático y QuickSort logarítmico y se cumple el orden de eficiencia supuesto.

5. Cálculo del tiempo experimental:

Para el estudio empírico tendremos que medir los recursos empleados (tiempo). Para ello elaboramos pruebas en diferentes condiciones (incremento de la talla del problema) ya que el tiempo de ejecución del algoritmo dependerá del tamaño del problema. Para cada caso la medida computacional será el tiempo de ejecución.

Para el caso “estudio promedio” al ser aleatorio introduciremos un bucle for que tome diferentes valores y luego haga una media. Con la constante NUMREPETICIONES. Así la medida final será un promedio de todas las efectuadas para conseguir un resultado lo más empírico y veraz posible. Estos datos se mostrarán en la pantalla de la terminal y guardados en el fichero de datos y en el fichero de gráficos. En el fichero gráficos queda guardado por la clase Gráficas que crea un fichero de gnuplot (.gpl) con el código necesario para que se ejecute con la gráfica correspondiente.

Esto queda implementado gracias a la clase Mtime, con QueryPerformanceCounter obtenemos el tiempo que se tarda en ejecutar el algoritmo de ordenación. En AlgoritmosOrdenación hemos implementado el nuevo algoritmo QuickSort. Haciendo uso de los métodos de la clase ConjuntoInt podemos obtener vectores de datos aleatorios del tamaño talla (que es lo que vamos incrementando para analizar y estudiar el incremento del tiempo según la talla del problema). Estos vectores son los que ordenará el Algoritmo de Ordenación.

5.1. Tablas (ficheros) y Gráficas de coste

QuickSort

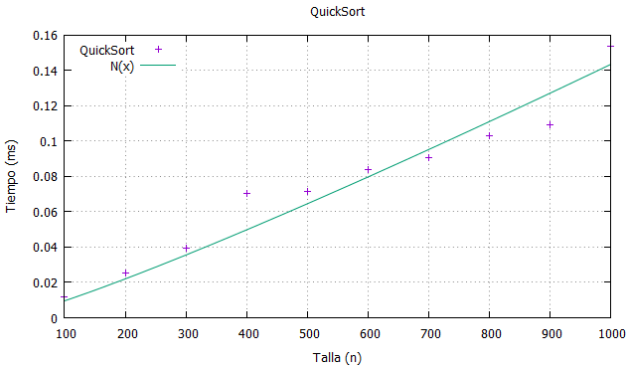
```
*** Ordenacion por QuickSort ***

Tiempos de ejecucion promedio

        Talla           Tiempo (mseg)

        100                 0.012
        200                 0.025
        300                 0.039
        400                 0.071
        500                 0.072
        600                 0.084
        700                 0.091
        800                   0.1
        900                  0.11
        1000                 0.15

Datos guardados en el fichero QuickSort.dat
```

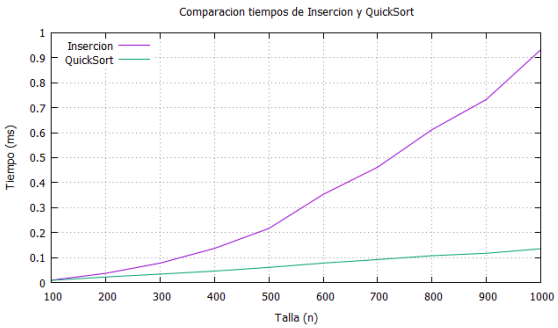


En este algoritmo obtenemos una gráfica creciente que parece una línea recta, esto se debe a que tomamos una porción de la curva muy ampliada. Esto es lo esperado ya que a mayor tamaño de problema mayor tiempo de ejecución. Hay puntos que no coinciden con la curva. Esto se debe a que al tratarse de una medida empírica y en tiempo real, el ordenador puede tardar más o menos en el proceso (por programas que tengamos abiertos por ejemplo).

5.2. Conclusiones

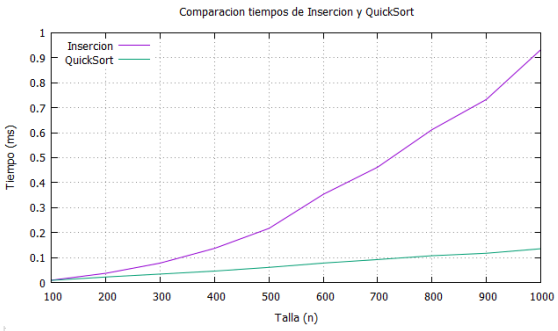
Para sacar conclusiones compararemos los datos y gráficas que comparan 2 algoritmos. Esto está implementado en el método comparar. Dos algoritmos, seleccionados por el usuario, ordenan el mismo vector variando la talla. Partiendo de la práctica 2 compararemos el más eficiente de los otros tres con este: Inserción con QuickSort:

Talla	Tiempo (mseg)	
	Insercion	QuickSort
100	0.0093	0.0086
200	0.037	0.022
300	0.078	0.034
400	0.14	0.046
500	0.22	0.061
600	0.35	0.078
700	0.46	0.092
800	0.61	0.11
900	0.73	0.12
1000	0.93	0.14



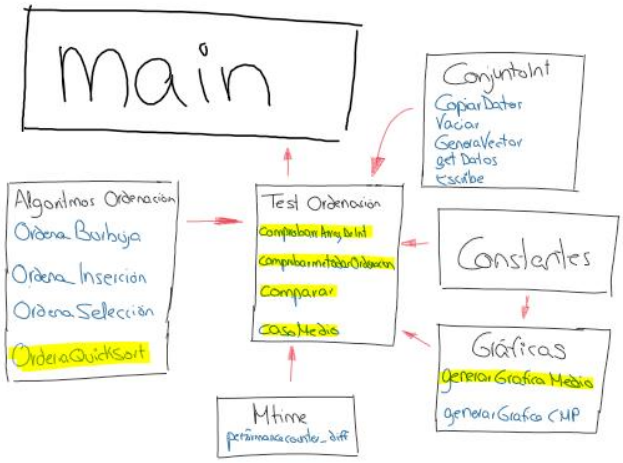
Entre estos dos métodos, podemos observar rápidamente que el más eficiente es QuickSort. Tanto para vectores de una talla pequeña como para mayores tiene un tiempo de ejecución menor que el de inserción y por tanto que el resto de algoritmos de orden cuadrático, además de tener un crecimiento más lento.

6. Comparación de los resultados teórico y experimental.



En este caso las escalas no son importantes ya que son aproximadamente proporcionales. Aún así podemos observar que la escala de la teórica es mucho más grande que la empírica. Esto se debe a que la teórica está medida en OE mientras la empírica en mseg. Por otro lado tenemos los picos, la gráfica teórica es mucho más suave al ser datos que siguen una fórmula mientras que en la empírica nos encontramos una gráfica con picos. Teniendo en cuenta que esto se debe al tiempo de ejecución de cada ordenador. Los resultados son coherentes y coinciden, QuickSort es el más eficiente.

7. Diseño de la aplicación.



El programa se compone de 5 clases: ConjuntoInt, Mtime, TestOrdenación, Graficas y AlgoritmosOrdenación además de un fichero de constantes.

Lo modificado o añadido es lo subrayado en amarillo.

Se ha introducido el nuevo algoritmo dentro de Algoritmos Ordenación adaptando el pseudocódigo a c++.

La clase TestOrdenación es la que contiene las funciones principales del programa que se llaman en el main. Tiene un constructor que inicia con los nombres de los algoritmos. Hemos tenido que modificarlo para introducir el nuevo.

El método comprobar métodos de ordenación se generar un vector y es ordenado por el método seleccionado. Si ejecutamos el programa nosotros podremos introducir el tamaño del vector. Se genera con valores aleatorios. Así podemos observar el funcionamiento de cada algoritmo y si ordena correctamente independientemente de la talla introducida. También ha sido modificado. Al igual que comparar y casoMedio.

En Gráficas se ha modificado generarGráficaMedio para tener en cuenta el orden del nuevo algoritmo. También se ha tenido que editar constantes para incluir NlogN.

En el main se ha introducido el nuevo algoritmo como opción en los menús correspondientes.

8. Conclusiones y valoraciones personales de la práctica.

Esta modificación nos ha permitido comprobar que hemos entendido el proceso de elaboración de la práctica. Ha sido una modificación sencilla al tratarse de incluir un nuevo algoritmo. Todo estaba hecho, sólo se ha tenido que tener en cuenta el orden de eficiencia del nuevo algoritmo e incluirlo en las funciones correspondientes.