Quicksort Central
peor: n^2
mejor: n*logn

El objetivo es calcular le complejidad promedio

ANÁLISIS Y DISEÑO DE ALGORITMOS

Complejidad temporal: Análisis empírico (I)

Práctica 1 de laboratorio

Entrega: Hasta el domingo 18 de febrero, 23:55h. A través de Moodle

Realizar un análisis empírico del tiempo promedio que el algoritmo de ordenación *Quicksort central* requiere para realizar su cometido.[1]

Para guiarte en este trabajo sigue las pautas del archivo `bubbleSort.tar.gz` suministrado (en este caso se refiere al algoritmo de ordenación *Burbuja*). Debes crear exactamente los mismos ficheros que los contenidos en dicho archivo comprimido pero relativos todos ellos a *Quicksort*.

De manera resumida, el trabajo a realizar consiste en:

1. Completar el código fuente del archivo `quickSort.cc` creando una función `main()` que actúe de la siguiente manera:

   a) Generando vectores de distintas tallas y con contenido aleatorio[2] para que quicksort los ordene. El tamaño de los vectores debe ser suficientemente grande para que el análisis no se desvirtúe por la influencia de factores ajenos a la propia ordenación, en este sentido, las potencias exactas de 2 desde 15 hasta 20 son tamaños apropiados.[3]

   b) Cronometrando el tiempo que tarda *Quicksort* en ordenar cada una de las instancias generadas. Para ello se debe seguir el procedimiento utilizado en `bubbleSort.cc`.

   c) Es importante tener en cuenta que este algoritmo, a diferencia de Burbuja, presenta *caso mejor* y *caso peor* en cuanto a complejidad temporal por lo que, para que el valor (de tiempo de ejecución) obtenido tenga validez estadística, hay que hacer varias repeticiones distintas para cada talla. Un número de repeticiones igual a 30 es suficiente.[4]

   d) La única salida por pantalla que debe proporcionar dicha función `main()` es una tabla que recoge el **tiempo medio** que el algoritmo de ordenación ha empleado en resolver todas las repeticiones de cada una de las tallas analizadas, similar a la que se muestra a continuación:

```
# QuickSort CPU-times in milliseconds:
# Size           Average CPU time (ms.)
#--------------------------------------
32768                          2.20
65536                          4.76
131072                         9.92
262144                        20.32
524288                        43.84
1048576                       92.04
```

2. Análisis de resultados mediante la herramienta `gnuplot`. A partir de la tabla de tiempos obtenida anteriormente y almacenada en un fichero con nombre `quickSort.CPUtimes`,[5] escribir un fichero de órdenes de `gnuplot`, similar a `bubbleSort.gpi` pero con nombre `quickSort.gpi` para realizar lo siguiente:

   a) Dibujar la gráfica que corresponde a dichos resultados, similar a la que se muestra en la figura 1 pero relativa al algoritmo que nos ocupa. La gráfica debe llamarse `quickSort.png`.

[1] Todos los materiales se proporcionan a través de *Moodle*; en este caso es el archivo `quickSort.cc` que contiene el código de este algoritmo de ordenación.

[2] La función `int rand(void)`, en `stdlib.h`, devuelve un entero pseudo-aleatorio entre 0 y `RAND_MAX`. Las series generadas son siempre las mismas aún en distintas ejecuciones puesto que siempre parten del mismo valor (semilla). Se usa la función `void srand(unsigned int semilla)` para modificar ese valor.

[3] Nótese que los tamaños de los vectores son significativamente más grandes que en el algoritmo Burbuja dada la gran diferencia entre ambos algoritmos en cuanto a tiempo de proceso.

[4] El *teorema central del límite* establece que a partir de un tamaño muestral suficiente los promedios muestrales seguirán la misma distribución de probabilidad que la de la variable de interés.

[5] Este fichero se obtiene utilizando la redirección de salida del intérprete de órdenes (*shell*) de *GNU/Linux*, a partir del programa realizado en el apartado 1.

b) Utilizar la función ***fit***[6] de `gnuplot` para analizar de qué tipo de función analítica podría tratarse. Hay que probar las siguientes funciones de coste: lineal; "$n \cdot \log(n)$" y "$n \cdot \log(n) + n$" y seleccionar la que mejor se ajuste.[7]

c) Dibujar conjuntamente ambas gráficas: la que se ha obtenido de forma empírica y la que consideras que mejor se ajusta, similar a la que se muestra en la figura 2 pero relativa a *Quicksort*. La gráfica debe llamarse `quickSort_and_Fit-Function.png`.

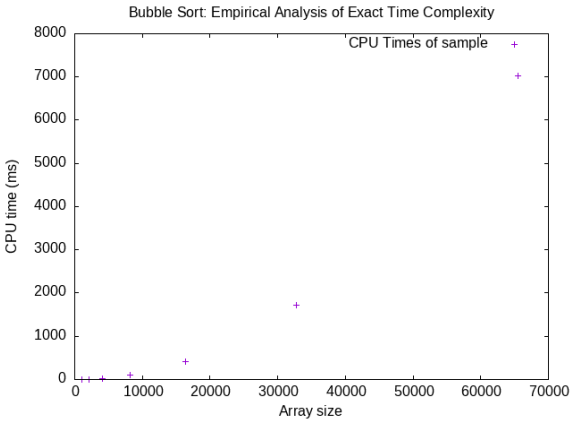

figura 1: Coste temporal exacto del algoritmo *Burbuja*. Datos obtenidos de forma empírica.

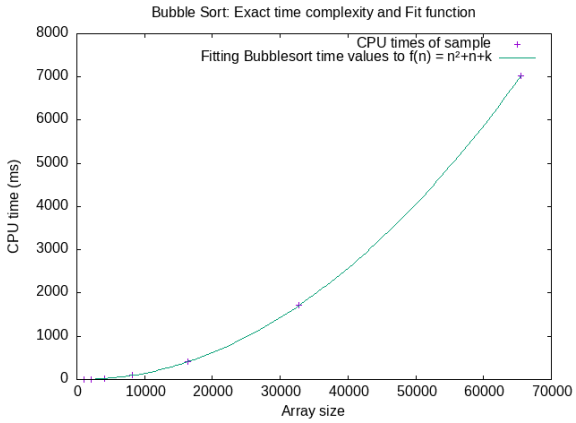

figura 2: En azul, coste temporal exacto del algoritmo *Burbuja*. En verde, ajuste a la función $an^2 + bn + c$.

3. Por último, hay que crear un fichero `makefile` que contenga los siguientes objetivos:

   - `quickSort`, para crear el ejecutable que se describe en el apartado 1. Este objetivo dependerá únicamente del código fuente `quickSort.cc`.
   - `quickSort.CPUtimes`, para redirigir a un fichero del mismo nombre la salida del programa anterior (tabla de tiempos). Este objetivo dependerá por tanto de dicho programa.
   - `graphs`, para llamar a `gnuplot` con el fichero de órdenes creado en el punto 2. Este objetivo dependerá del fichero anterior y del fichero de órdenes de `gnuplot`. El resultado será las dos gráficas pedidas.
   - `all`, este objetivo dependerá de los tres anteriores y se utilizará para invocarlos de forma secuencial, rehaciéndolos si fuera necesario.

4. **Normas para la entrega.**

   ATENCIÓN: Estas normas son de obligado cumplimiento para que esta práctica sea evaluada.

   a) Se debe entregar únicamente los ficheros `quickSort.cc`, `quickSort.gpi` y `makefile`. Sigue escrupulosamente los nombres de ficheros, objetivos, etc. que se citan en este enunciado. **No hay que entregar nada más.**

   b) Es imprescindible que no presente errores ni de compilación ni de interpretación (según corresponda), en los ordenadores del laboratorio asignado y en el sistema operativo *GNU/Linux*.[8] Se tratará de evitar también cualquier tipo de aviso (*warning*).

   c) Todos los ficheros que se entregan deben contener el nombre del autor y su DNI (o NIE) en su primera línea (entre comentarios apropiados según el tipo de archivo).

   d) Se comprimirán en un archivo `.tar.gz` cuyo nombre será el DNI del alumno, compuesto de 8 dígitos y una letra (o NIE, compuesto de una letra seguida de 7 dígitos y otra letra). Por ejemplo: `12345678A.tar.gz` o `X1234567A.tar.gz`. **Solo se admite este formato de compresión y solo es válida esta forma de nombrar el archivo.**

   e) En el archivo comprimido **no deben existir subcarpetas**, es decir, al extraer sus archivos estos deben quedar guardados en la misma carpeta donde está el archivo que los contiene.

   f) La práctica se debe subir a *Moodle* respetando las fechas expuestas en el encabezado de este enunciado.

[6] La función `fit` implementa la técnica denominada "mínimos cuadrados" que, dado un conjunto de valores (x, y) y una función analítica suministrada, encuentra los coeficientes de dicha función que mejor se ajustan a los valores suministrados de acuerdo con el criterio de mínimo error cuadrático.

[7] Se trata de observar cuál de esas funciones minimiza la media cuadrática de los residuos (*rms of residuals*). Es necesario advertir que este análisis no tiene por qué ser concluyente puesto que, por regla general, cuanto mayor es el grado de una curva mejor puede ajustarse a unos datos concretos aunque sin capacidad para generalizar de forma correcta. Por lo tanto, para saber cuál es la función teórica que mejor se ajusta se deben utilizar los métodos analíticos estudiados en clase de teoría.

[8] Si trabajas con tu propio ordenador o con otro sistema operativo asegúrate de que este requisito se cumple.