

Trabajo_Prácticas_ED.pdf



LosCocos



Estructuras de Datos



2º Grado en Ingeniería Informática



Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

EJERCICIOS PRÁCTICA 1-ESTRUCTURA DE DATOS

1. Calcule la eficiencia teórica de este algoritmo

Crear un fichero `ordenacion.cpp` con el programa completo para realizar una ejecución del algoritmo.

Realizamos la función ordenación y el script `ejecuciones_ordenacion.csh` y generamos el archivo "tiempos_ordenacion.dat"

$$\begin{aligned}
 &\text{Void ordenar (int *v, int u)} \{ \\
 &\quad \text{for (int i=0; i < u-1; i++)} \\
 &\quad \text{for (int j=0; j < u-i-1; j++)} \\
 &\quad \text{if (v[j] > v[j+1])} \{ \\
 &\quad \quad \text{int aux = v[j];} \\
 &\quad \quad \text{v[j] = v[j+1];} \\
 &\quad \quad \text{v[j+1] = aux;} \\
 &\quad \} \\
 &\} \\
 \\
 &\sum_{i=0}^{u-2} \sum_{j=0}^{u-i-2} 1 = \sum_{i=0}^{u-2} (u-i-1) = \sum_{i=0}^{u-2} u - \sum_{i=0}^{u-2} i = \sum_{i=0}^{u-2} u - \sum_{i=0}^{u-2} i \\
 &= \sum_{i=0}^{u-2} u = u \sum_{i=0}^{u-2} 1 = u(u-1) \\
 &= \sum_{i=0}^{u-2} i = \frac{(u-2)(u-1)}{2} = \frac{u^2 - 2u - u + 2}{2} = \frac{u^2 - 3u + 2}{2} \\
 &= \sum_{i=0}^{u-2} 1 = u-2 + 1 = u-1 \\
 \\
 &\rightarrow = u(u-1) - \frac{u^2 - 3u + 2}{2} - (u-1) = u^2 - u - \frac{u^2 - 3u + 2}{2} \\
 &= -u + 1 = \frac{u^2 - 2u + 1}{2} - \frac{u^2 - 3u + 2}{2} \\
 &= \frac{2u^2 - 4u + 2 - u^2 + 3u - 2}{2} = \frac{u^2 - u}{2}
 \end{aligned}$$

```
#!/bin/csh
@ inicio = 100
@ fin = 30000
@ incremento = 500
set ejecutable = ordenacion
set salida = tiempos_busqueda.dat

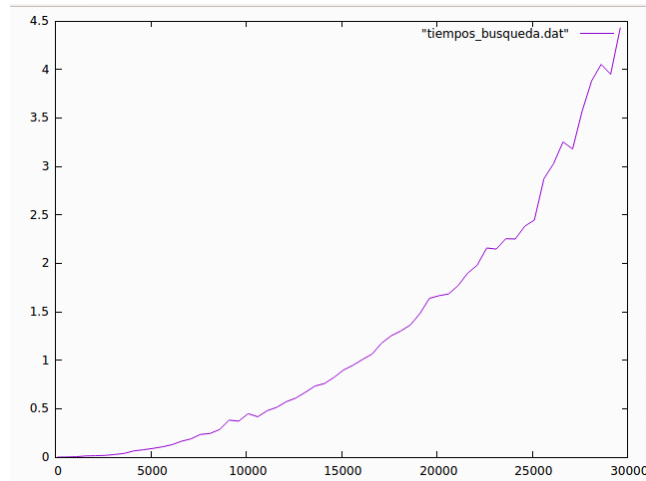
@ i = $inicio
echo > $salida
while ( $i <= $fin )
    echo Ejecución tam = $i
    echo `./{Sejecutable} $i 10000` >> $salida
    @ i += $incremento
end
```

```
1 #include <iostream>
2 #include <ctime> // Recursos para medir tiempos
3 #include <cstdlib> // Para generación de números pseudoaleatorios
4
5 using namespace std;
6
7 void ordenar(int *v, int n) {
8     for (int i=0; i<n-1; i++)
9         for (int j=0; j<n-i-1; j++)
10             if (v[j]>v[j+1]) {
11                 int aux = v[j];
12                 v[j] = v[j+1];
13                 v[j+1] = aux;
14             }
15 }
16
17 void sintaxis() {
18     cerr << "Sintaxis:" << endl;
19     cerr << " TAM: Tamaño del vector (>0)" << endl;
20     cerr << " VMAX: Valor máximo (>0)" << endl;
21     cerr << "Genera un vector de TAM números aleatorios en [0,VMAX]" << endl;
22     exit(EXIT_FAILURE);
23 }
24
25 int main(int argc, char * argv[]) {
26     if (argc!=3) // Lectura de parámetros
27         sintaxis();
28     int tam=atoi(argv[1]); // Tamaño del vector
29     int vmax=atoi(argv[2]); // Valor máximo
30     if (tam<=0 || vmax<=0)
31         sintaxis(); // Generación del vector aleatorio
32     int *v=new int[tam]; // Reserva de memoria
33     srand(time(0)); // Inicialización generador números pseudoaleatorios
34     for (int i=0; i<tam; i++) // Recorrer vector
35         v[i] = rand() % vmax; // Generar aleatorio [0,vmax]
36
37     clock_t tini; // Anotamos el tiempo de inicio
38     tini=clock();
39
40     int x = vmax+1; // Buscamos un valor que no está en el vector
41     ordenar(v,tam); // de esta forma forzamos el peor caso
42
43     clock_t tfini; // Anotamos el tiempo de finalización
44     tfini=clock();
45
46     // Mostramos resultados (Tamaño del vector y tiempo de ejecución en seg.)
47     cout << endl << tam << "\t" << (tfini-tini)/(double)CLOCKS_PER_SEC << endl;
48
49     delete [] v; // Liberamos memoria dinámica
50 }
```

Usar gnuplot para dibujar los datos obtenidos en el apartado previo.

En gnuplot llevamos a cabo las siguientes órdenes:

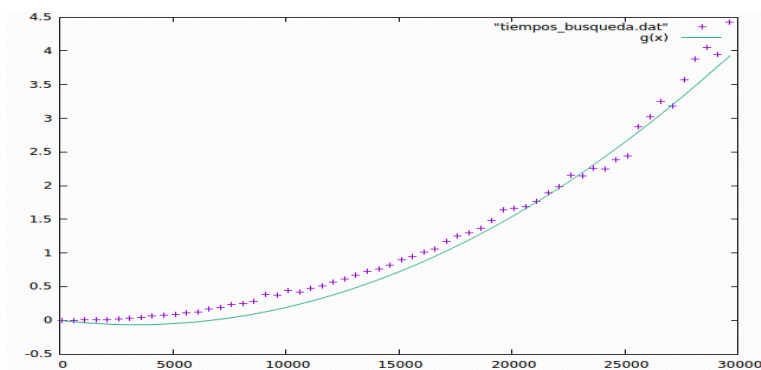
- $f(x)=a*x+b$ (declaramos la función)
- Fit "tiempos_ordenacion.dat" via a,b (para calcular los valores de a y b)
- Plot "tiempos_ordenacion.dat" (obtenemos esta grafica)



Pruebe a dibujar superpuestas la función con la eficiencia teórica y la empírica. ¿Qué sucede?

Establecemos una función nueva de la misma forma que la eficiencia teórica que hemos calculado

- $G(x)=a*x*x + b*x$
- Fit "tiempos_ordenacion.dat" via a,b
- Plot "tiempos_ordenacion.dat", $g(x)$ para mostrar ambas funciones superpuestas.



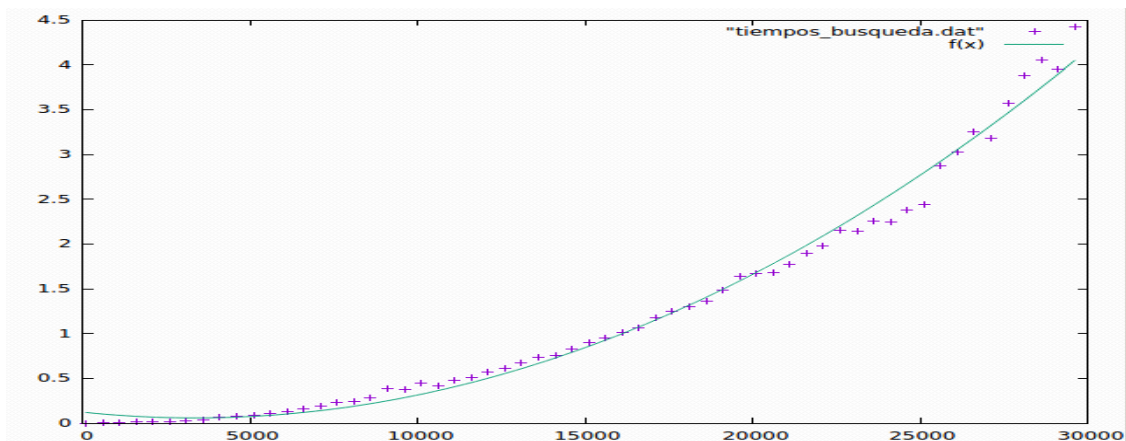
"Los tiempos suelen quedar por encima de la eficiencia teórica"

2. Calcule la siguiente eficiencia:

$$F(x) = a \cdot x^2 + b \cdot x + c$$

Fit "tiempos_ordenacion.dat" via a,b,c

Plot "tiempos_ordenacion.dat", $f(x)$ obtenemos una gráfica muy ajustada a los tiempos de ordenación



3. Explique qué hace este algoritmo.

Se trata del algoritmo de búsqueda muy eficiente y rápido conocido como "búsqueda binaria". Este método nos permite encontrar un elemento en un vector ordenado de la siguiente manera:

Empezará evaluando el centro del vector, si el elemento se encuentra, finaliza la ejecución del programa. En caso contrario, el programa comprueba si el elemento que buscamos es mayor o menor, quedándose en una de las dos mitades del vector. El programa sigue iterando y dividiendo en dos mitades al segmento del vector hasta encontrar nuestro elemento.

Calcule su eficiencia teórica.

```
int operacion (int *v, int u, int x, int *iug, int *sup) {
    int med;
    bool enc = false;
    while ((iug < sup) && (!enc)) {
        med = (iug + sup) / 2;
        if (v[med] == x)
            enc = true;
        else if (v[med] < x)
            iug = med + 1;
        else
            sup = med - 1;
    }
}
```


$O(\log_2(u))$ puesto que en cada iteración vamos reduciendo a la mitad el tamaño del vector.

Con $u = \text{tamaño_vector}$ corresponden $\log_2(u)$ iteraciones

WUOLAH

Calcule su eficiencia empírica.

Obtenemos los siguientes tiempos de búsqueda y los representamos con plot "tiempos_binaria.dat"



```
100 2e-06
500 2e-06
1100 2e-06
1600 2e-06
2100 1e-06
2600 2e-06
3100 2e-06
3600 2e-06
4100 1e-06
4600 3e-06
5100 2e-06
5600 1e-06
6100 2e-06
6600 2e-06
7100 2e-06
7600 2e-06
8100 2e-06
8600 2e-06
9100 2e-06
9600 3e-06
10100 1e-06
10600 2e-06
11100 2e-06
11600 2e-06
12100 2e-06
12600 2e-06
13100 2e-06
13600 2e-06
14100 2e-06
14600 2e-06
15100 2e-06
15600 1e-06
16100 2e-06
16600 2e-06
17100 1e-06
17600 4e-06
18100 1e-06
18600 2e-06
```

Como el algoritmo es muy eficiente, los tiempos de respuesta que obtenemos son muy pequeños y no se pueden comparar entre ellos. Para solucionarlo proponemos aumentar el tiempo que tarda el algoritmo en calcular las respuestas. Para ello, encapsulamos el algoritmo de la búsqueda binaria en un for.

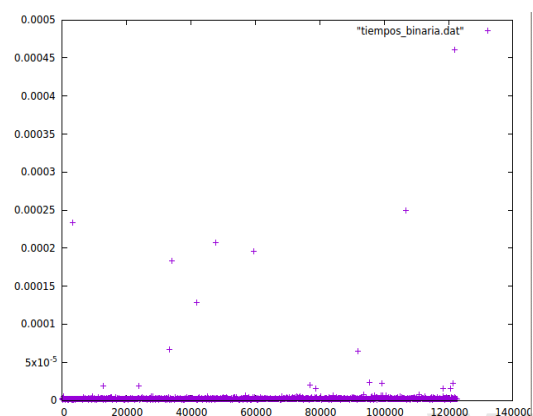
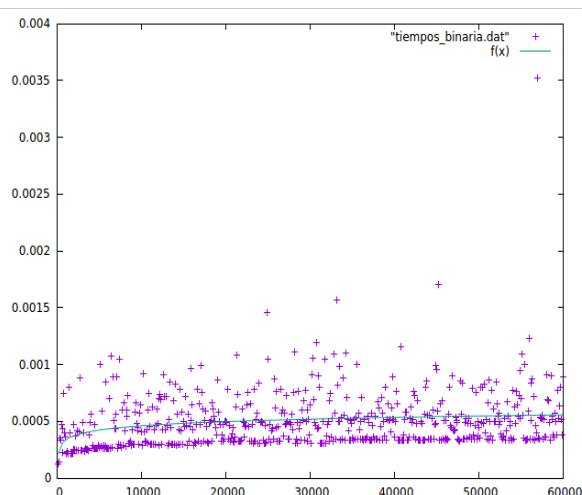
```
// Algoritmo a evaluar
for(int i=0; i<5000; i++){
operacion(v,tam,tam+1,0,tam-1);
}
```

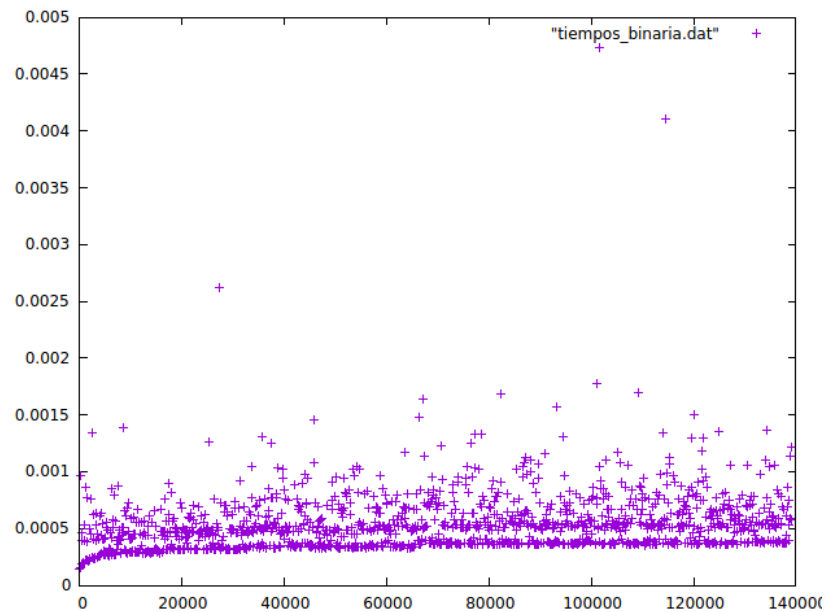
Ya podemos comparar la eficiencia teórica con la empírica:

$$F(x)=a*\log(x)/\log(10)+b$$

Fit $f(x)$ "tiempos_binaria.dat" via a,b

Plot "tiempos_binaria.dat" , $f(x)$ y obtenemos:





4. Retome el ejercicio de ordenación mediante el algoritmo de la burbuja. Debe modificar el código que genera los datos de entrada para situarnos en dos escenarios diferentes:

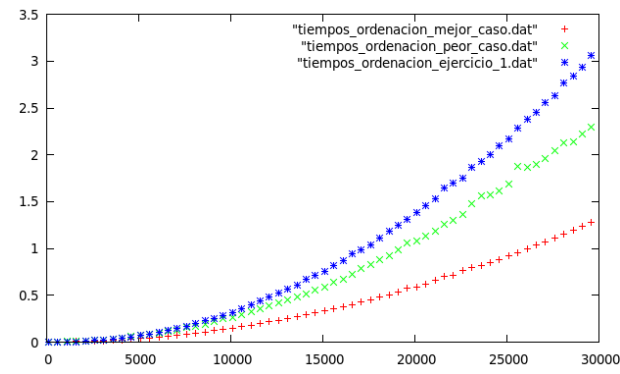
- El mejor caso posible. Para este algoritmo, si la entrada es un vector que ya está ordenado el tiempo de cómputo es menor ya que no tiene que intercambiar ningún elemento.
- El peor caso posible. Si la entrada es un vector ordenado en orden inverso estaremos en la peor situación posible ya que en cada iteración del bucle interno hay que hacer un intercambio.

Calcule la eficiencia empírica en ambos escenarios y compárela con el resultado del ejercicio 1.

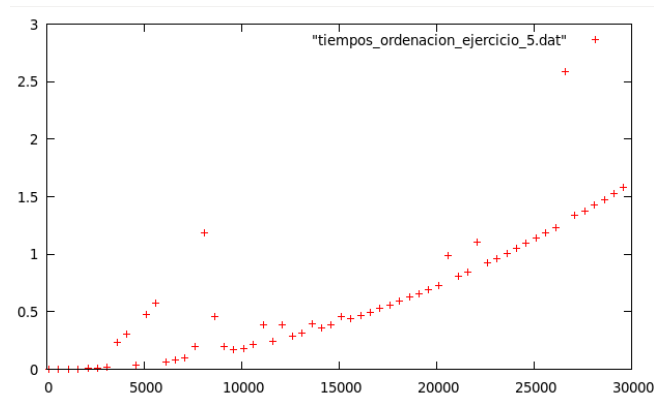
<pre> 1 void orde_bur_mejor_caso(int v, int n){ 2 for(int i=0; i=n-1; i++){ 3 for(int i=0; i=n-1; i++){ 4 if(v[j]>v[j+1]){ 5 int aux=v[j]; 6 v[j]=v[j+1]; 7 v[j+1]=aux; 8 } 9 } 10 } 11 } </pre>	<pre> 14 void orde_bur_mejor_caso(int v, int n){ 15 bool parar=true; 16 for(int i=0; i=n-1&&parar; i++){ 17 parar=false; 18 for(int i=0; i=n-1; i++){ 19 if(v[j]>v[j+1]){ 20 parar=true; 21 int aux=v[j]; 22 v[j]=v[j+1]; 23 v[j+1]=aux; 24 } 25 } 26 } 27 } </pre>
--	--

Una vez hechos los algoritmos, conseguidos los tiempos de ejecución nos damos cuenta de:

El mejor algoritmo es Mejor Caso, seguido del peor Caso y por último el Ejercicio 1.



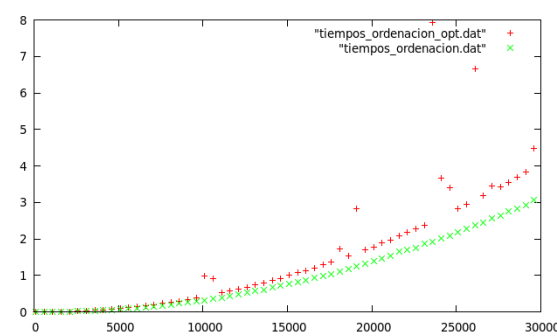
5. Considere ahora la situación del mejor caso posible en la que el vector de entrada ya está ordenado. Muestre la gráfica con la eficiencia empírica y compruebe si se ajusta a la previsión.



"Queda reflejado que se ajusta según lo previsto anteriormente"

6. Retome el ejercicio de ordenación mediante el algoritmo de la burbuja. Ahora replique dicho ejercicio pero previamente deberá compilar el programa indicándole al compilador que optimice el código.

Compare las curvas de eficiencia empírica para ver cómo mejora esto la eficiencia del programa.



Aunque al principio haya muy poca diferencia, conforme avanza la ejecución el método optimizado se hace menos eficiente que el algoritmo de ordenación.

Con la colaboración de PS Management , Master Card e Informáticos Valencia