UNIVERSIDAD
DE GRANADA

Algoritmica
Grado en Ingeniería Informática
Curso 2018-2019

Ejercicio Propuesto 1.
Principio de Invarianza.

Félix Ramírez García
felixramirezgarcia@correo.ugr.es

22 de febrero de 2019

Índice

1 Introduccion **3**

2 Ampliacion del problema **8**

Índice de figuras

1.1 Grafica BurbujaC 5
1.2 Grafica BurbujaJava y BurbujaC 7
2.1 Grafica BurbujaJava y BurbujaC con mayor tamaño N 8

Índice de tablas

1. Introduccion

El enunciado del ejercicio nos propone usar los programas BurbujaC.cpp y BurbujaJava.java para ilustrar el cumplimiento del principio de invariaza. Para ello ejecutaremos la misma función en ambos programas con diferentes tamaños del problema anotando sus tiempos de ejecucion para comparar los resultados gráficamente.

Recordemos que el principio de invarianza plantea que dos versiones del mismo algoritmo no difieren en su eficiencia en más que una constante. Por lo que al ejecutar los dos programas obtendremos un resultado similar independientemente de la maquina y del lenguaje de programación usado.

Empezando con el siguiente programa en C++ (BurbujaC.cpp) que ordena una lista desordenada en orden inverso (peor caso) de 20.000 numeros , cada 1000 (se ordenan 20 arrays), compilado y ejecutado con las ordenes :

g++ -o BurbujaC.cpp BurbujaC
./BurbujaC

```
#include <iostream>
#include <ctime>
using namespace std;

void Burbuja(double *v, int posini, int posfin) {
        int i, j;
        double aux;
        bool haycambios= true;
        i= posini;

        while (haycambios) {
                haycambios=false; // Suponemos vector ya ordenado

                // Recorremos vector de final a i
                for (j= posfin; j>i; j--) {

                        // Dos elementos consecutivos mal ordenados
                        if (v[j-1]>v[j]) {
                                aux= v[j]; // Los intercambiamos
                                v[j]= v[j-1];
                                v[j-1]= aux;

                                // Al intercambiar, hay cambio
                                haycambios= true;
                        }
                }

                i++;
        }
}
```

```
int main()
{
        const int SIZE= 20000;
        double vect[SIZE];
        unsigned long tini, tfin;

        for (int TAM= 1000; TAM<=SIZE; TAM+= 1000) {
                // Ejemplo: Vector al reves
                for (int i= 0; i<TAM; i++)
                vect[i]= TAM-i;

                tini= clock(); // Tiempo inicial
                Burbuja(vect, 0, TAM-1);
                tfin= clock(); // Tiempo final

                cout<<"N: "<<TAM<<" T (ms.): "<<1000.0*(tfin-tini)/(
                    double)CLOCKS_PER_SEC<<endl;
        }
        return 0;
}
```

Obtenemos la siguiente salida :

```
N: 1000 T (ms.): 0
N: 2000 T (ms.): 15.625
N: 3000 T (ms.): 31.25
N: 4000 T (ms.): 62.5
N: 5000 T (ms.): 93.75
N: 6000 T (ms.): 140.625
N: 7000 T (ms.): 234.375
N: 8000 T (ms.): 312.5
N: 9000 T (ms.): 328.125
N: 10000 T (ms.): 421.875
N: 11000 T (ms.): 609.375
N: 12000 T (ms.): 656.25
N: 13000 T (ms.): 671.875
N: 14000 T (ms.): 812.5
N: 15000 T (ms.): 937.5
N: 16000 T (ms.): 1156.25
N: 17000 T (ms.): 1250
N: 18000 T (ms.): 1421.88
N: 19000 T (ms.): 1406.25
N: 20000 T (ms.): 1656.25
```

Y para una mejor visualización representamos los resultados en una gráfica (figura 1.1).

Ahora procedemos a realizar los mismo con el programa BurbujaJava.java, que ordena una lista desordenada en orden inverso (peor caso) de 20.000 números , cada 1000 (se ordenan 20 arrays). Cuyo código es el siguiente:

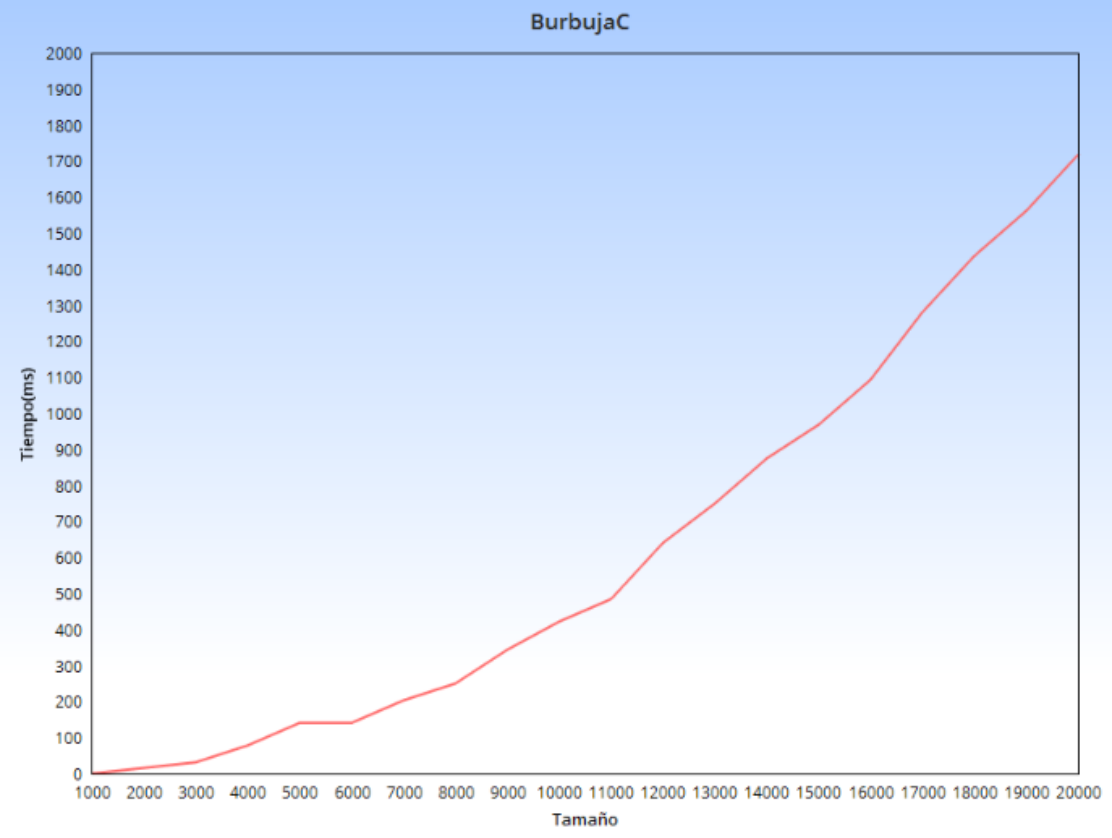

Figura 1.1: Grafica BurbujaC

```
public class BurbujaJava {

        public static void Burbuja (double [] v, int posini, int posfin)
            {
                int i, j;
                double aux;
                boolean haycambios= true;
                i= posini;

                while (haycambios) {
                        haycambios=false; // Suponemos vector ya ordenado

                        // Recorremos vector de final a i
                        for (j= posfin; j>i; j--) {
                                // Dos elementos consecutivos mal
                                    ordenados
                                if (v[j-1]>v[j]) {
```

```
                                        aux= v[j]; // Los intercambiamos
                                        v[j]= v[j-1];
                                        v[j-1]= aux;

                                        // Al intercambiar, hay cambio
                                        haycambios= true;
                                }
                        }

                        i++;
                }
}

public static void main(String[] args) {
        final int SIZE= 20000;
        double []vect= new double[SIZE];
        long tini, tfin;

        for (int TAM= 1000; TAM<=SIZE; TAM+= 1000) {
                // Ejemplo: Vector al reves
                for (int i= 0; i<TAM; i++)
                        vect[i]= TAM-i;

                        tini= System.currentTimeMillis();
                        Burbuja(vect, 0, TAM-1);
                        tfin= System.currentTimeMillis();

                        System.out.println("N: "+TAM+" T (ms): "+(tfin-
                            tini));
                }
        }

}
```

Las ordenes de compilación y ejecucion han sido:

javac BurbujaJava.java
java BurbujaJava

Al ejecutar el programa BurbujaJava la salida es la siguiente:

```
N: 1000 T (ms): 9
N: 2000 T (ms): 3
N: 3000 T (ms): 22
N: 4000 T (ms): 11
N: 5000 T (ms): 18
N: 6000 T (ms): 24
N: 7000 T (ms): 34
N: 8000 T (ms): 68
N: 9000 T (ms): 71
N: 10000 T (ms): 84
N: 11000 T (ms): 109
```

```
N: 12000 T (ms): 105
N: 13000 T (ms): 125
N: 14000 T (ms): 144
N: 15000 T (ms): 183
N: 16000 T (ms): 215
N: 17000 T (ms): 237
N: 18000 T (ms): 291
N: 19000 T (ms): 304
N: 20000 T (ms): 359
```

Y para una mejor visualización representamos los resultados junto con los obtenidos en BurbujaC (figura 1.2).

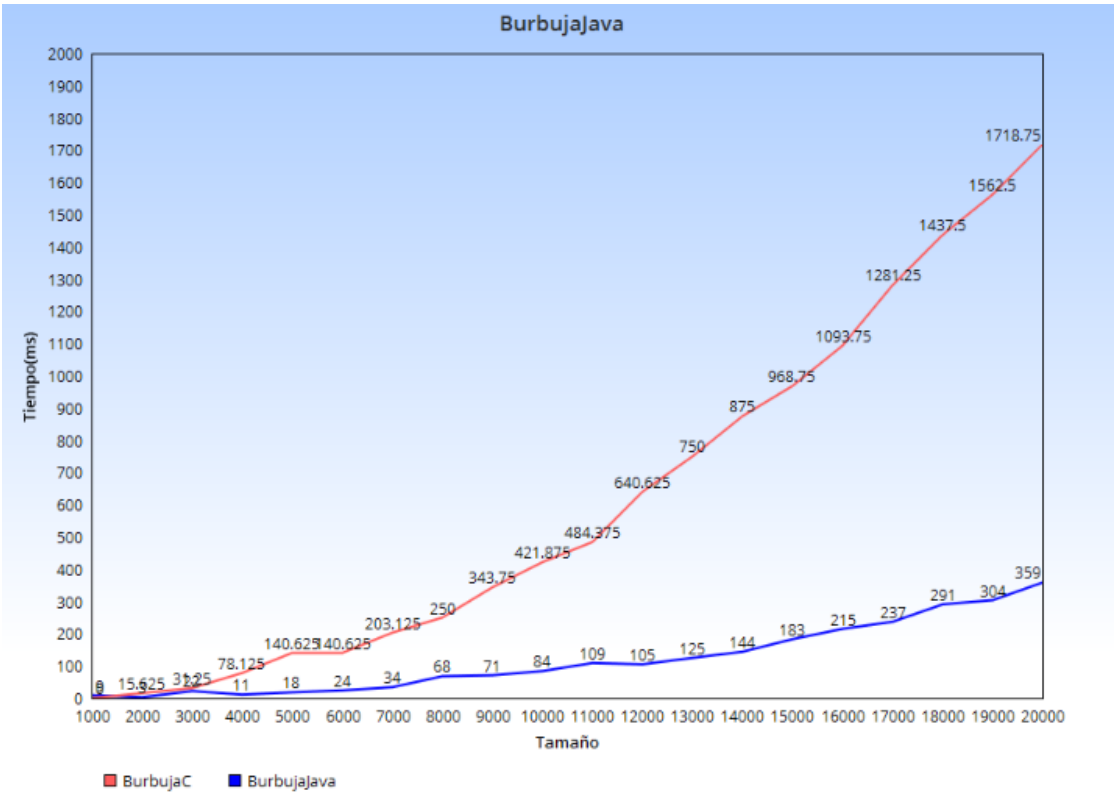

Figura 1.2: Grafica BurbujaJava y BurbujaC

Como se puede apreciar en la gráfica, el programa BurbujaC es mas lento que el programa BurbujaJava, esto se debe a que la constante multiplicativa del termino con mayor exponente (grado 2 en este caso) de la funcion de BurbujaC es mayor que la de BurbujaJava. Para intentar representar este hecho mas fácilmente se ha realizado en la siguiente sección una ampliación del problema.

2. Ampliacion del problema

Para terminar vamos a realizar lo mismo que para el apartado anterior pero ahora con un tamaño a ordenar mucho mayor. Para ello se han vuelto a ejecutar ambos programas pero ahora con un tamaño de 200.000 ejecutados cada 10.000 (Un total de 20 ordenaciones).

Después de ejecutar ambos programas se han anotado los resultados y se han introducido en la figura 2.1 .

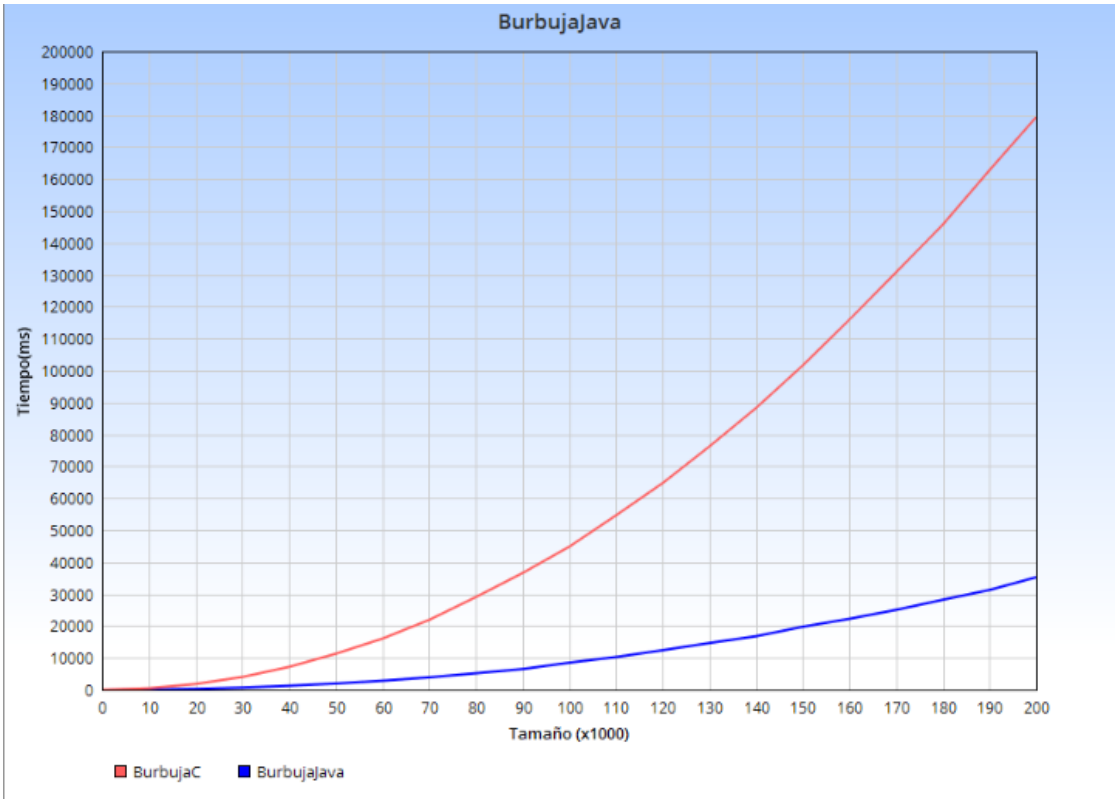

Figura 2.1: Grafica BurbujaJava y BurbujaC con mayor tamaño N

Por ultimo , partiendo de que ambas gráficas son parábolas y su eficiencia que pertenece a O(n*n) viene sujeta a la anidación entre los blucles while y for, vamos a intentar calcular dos funciones aproximadas a las dos lineas de la gráfica. Para ello vamos a seleccionar tres puntos y a calcular la función.

De la linea obtenida por el programa BurbujaC (color rojo) se han seleccionado los puntos:

P1 = (0 , 0)
P2 = (80.000 , 30.000)

P3 = (140.000 , 90.000)

Por lo que el sistema de ecuaciones para calcular la curva utilizando la función general de una parábola (y = a*x*x + b*x + c) sera:

0 = a*0*0 + b*0 + c
30.000 = a*80.000*80.000 + b*80.000 + c
90.000 = a*140.000*140.000 + b*140.000 + c

Por lo que al resolver el sistema de ecuaciones obtenemos que la funcion que representa la curva es :

f(x) = (1/224000)*x*x + (1/56)x

De la linea obtenida por el programa BurbujaJava (color azul) se han seleccionado los puntos:

P1 = (0 , 0)
P2 = (110.000 , 10.000)
P3 = (180.000 , 30.000)

Por lo que el sistema de ecuaciones para calcular la curva utilizando la función general de una parábola (y = a*x*x + b*x + c) sera:

0 = a*0*0 + b*0 + c
10.000 = a*110.000*110.000 + b*110.000 + c
30.000 = a*180.000*180.000 + b*180.000 + c

Por lo que al resolver el sistema de ecuaciones obtenemos que la función que representa la curva es :

f(x) = 0.0000109*x*x - 0.03x

Como conclusión obtenemos que la constante multiplicativa (aproximada) de BurbujaC (0.0000446) es mayor que la de BurbujaJava (0.0000109), por lo que los tiempos de ejecucion también son mayores.