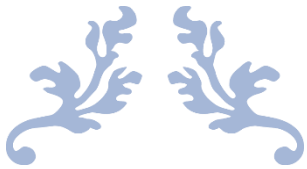


PRACTICA 1: EFICIENCIA

Ejercicio 5: Dependencia de la implementación

FRANCISCO RUIZ ADÁN

Hardware

CPU: Intel Core i7-7700HQ @ 2.80 GHz 8 cores, Intel HD Graphics 630

RAM: 8GB

Sistema operativo: Ubuntu 18.04 LTS 64-bit

Compilador: g++ versión 7.4.0

Opciones de compilación: -o

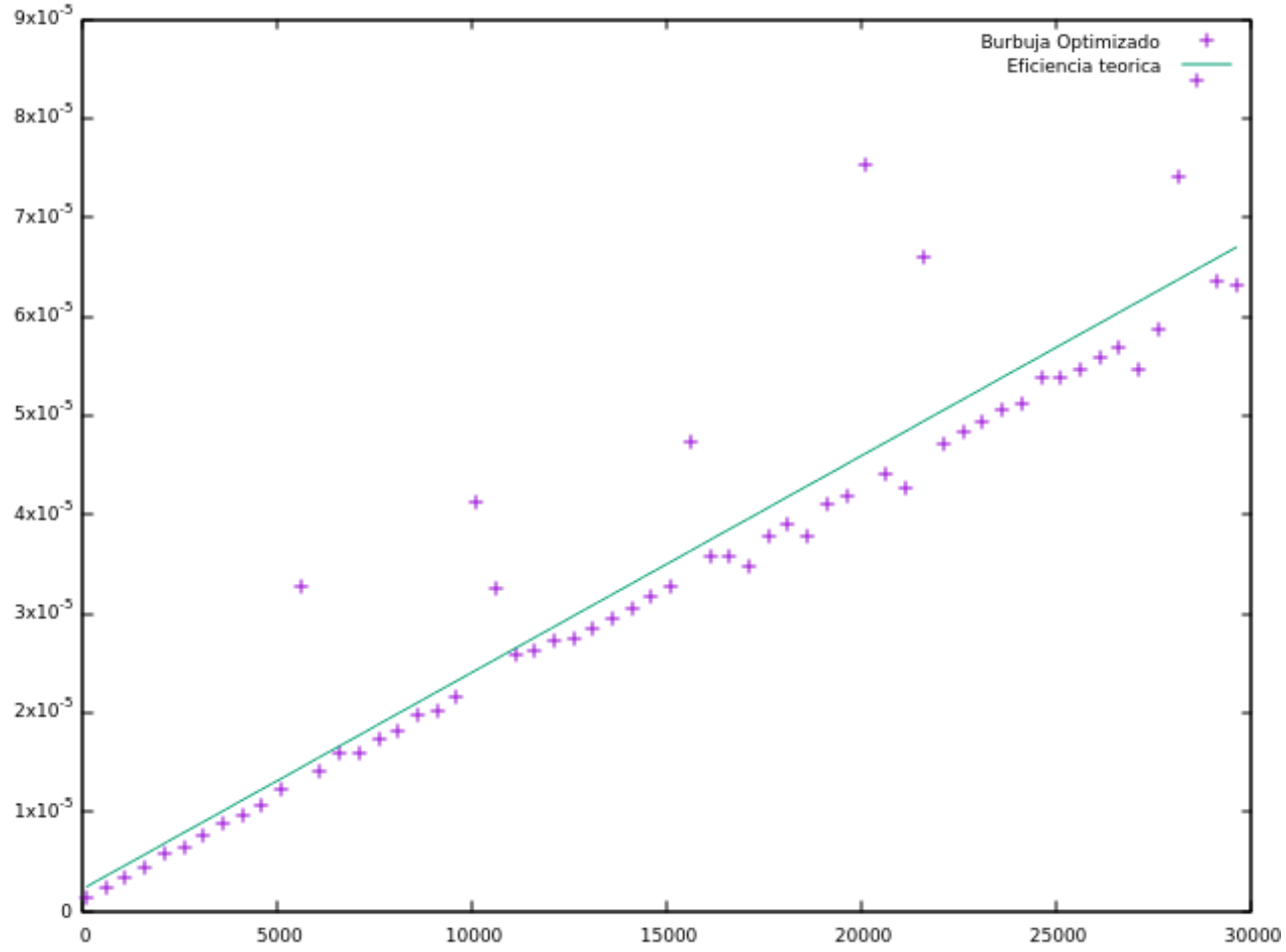
En este ejercicio se nos proporciona una implementación diferente del algoritmo de ordenación burbuja:

```
void ordenar(int *v, int n)
{
    bool cambio = true;
    for (int i = 0; i < n - 1 && cambio; i++)
    {
        cambio = false;
        for (int j = 0; j < n - i - 1; j++)
            if (v[j] > v[j + 1])
            {
                cambio = true;
                int aux = v[j];
                v[j] = v[j + 1];
                v[j + 1] = aux;
            }
    }
}
```

En ella se ha añadido una variable que permite saber si en una de las iteraciones del bucle externo no se ha modificado el vector. Si esto ocurre significa que el vector está ordenado y no hace falta continuar.

Si consideramos ahora la mejor situación posible donde el algoritmo recibe un vector ordenado y analizamos su eficiencia vemos que ésta ya no es de orden $O(n^2)$ sino de orden $O(n)$. Esto es así debido a que el algoritmo solamente recorrería una única vez el vector, hasta darse cuenta de que esta ordenado y parar.

Si analizamos la eficiencia empírica y realizamos el ajuste obtenemos una gráfica parecida a la siguiente:



Como podemos observar, la eficiencia es de orden lineal, con una función de ajuste $f(x)=2.18886 \cdot 10^{-9}x+2.18204 \cdot 10^{-6}$

Nota: para realizar las mediciones de tiempo se ha empleado la librería chrono de C++, la cual cuenta con relojes de muy alta precisión. Se ha optado por este método debido a que haciendo uso de la función clock, marcaba como tiempo para un tamaño entre 100 y 30000 de 0. La forma de hacerlo es la siguiente:

```cpp
int main(int argc, char *argv[])
{
    using namespace std::chrono;

    //steady_clock es un reloj especialmente diseñado para calcular
intervalos
    steady_clock::time_point inicio, fin;
    duration<double> tiempo;

    if (argc != 2)
        sintaxis();

    int tamanio = atoi(argv[1]);

    int *vec = new int[tamanio];

    crearMejorCaso(vec,tamanio);

    inicio = steady_clock::now();

    ordenar(vec, tamanio); //  Ordenamos el vector

    fin = steady_clock::now();

    tiempo = duration_cast<duration<double>>(fin - inicio);

    cout << tamanio << "\t" << tiempo.count() << endl;

    delete[] vec;
}
```