Estructura de Datos: Práctica 1

Alejandro Núñez Pérez

Ejercicio 1

La eficiencia teórica del algoritmo *Bubble Sort* es del órden *O(n^2)*

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

Al superponer la gráfica de la eficiencia teórica y de la eficiencia empírica, obtenemos que, aunque tienen el mismo órden de crecimiento, los valores son muy dispares: los datos empíricos no están ajustados a la función de la curva.

Ejercicio 2

Al ajustar los datos empíricos con la función de la curva, obtenemos una gráfica mucho más pareja al órden $O(n^2)$ del algoritmo *Bubble Sort*. *(Ver figura 1)*

Ejercicio 3

El código de `ejercicio_desc.cpp` representa una búsqueda binaria, una búsqueda de un elemento `x` que divide el conjunto en dos partes iguales y analiza el medio.

El problema de la medición es que es un algoritmo con eficiencia de órden *O(log(n))* y se ejecuta demasiado rápido para la biblioteca usada para medir el tiempo (`ctime`). Un *workaround* posible es ejecutar `m` veces el algoritmo y dividir el total por `m`, para obtener una unidad medible.

Una vez planteado todo, queda una gráfica que se asemeja a la eficiencia teórica, aunque con algún que otro pico (ver figura 2).

Ejercicio 4

El mejor caso del algoritmo es que los datos ya estén ordenados (no hay cambios en los datos), mientras que el peor de los casos es que estén inversamente ordenados (hay cambios en cada iteración). La forma de generar los datos cambia a...

```
// Mejor caso (ya ordenado)
for(int i = 0; i < tam; i++) {
    v[i] = i;
}

// Peor caso (órden inverso)
for(int i = 0; i < tam; i++) {
    v[i] = (tam-i);
}
```

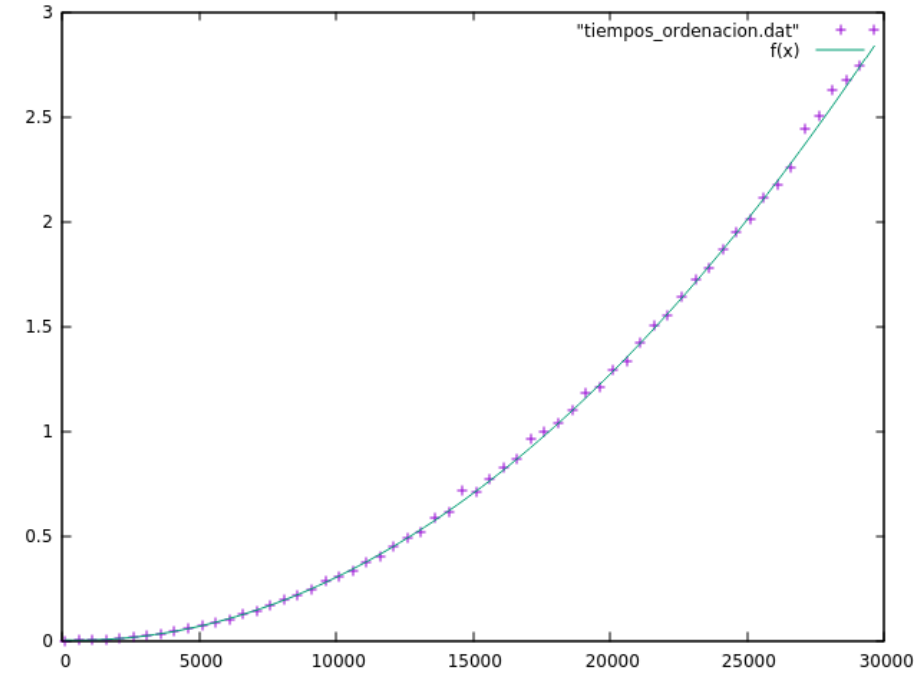
Podemos obtener una gráfica que represente los tres casos, que podemos usar para comparar los tres casos. *(Ver figura 3)*

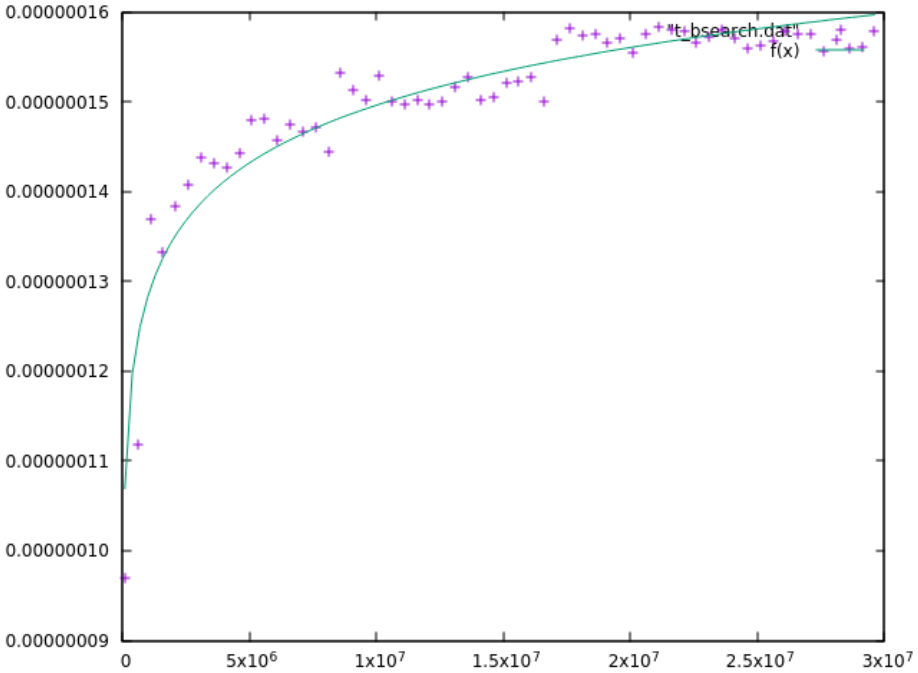
NOTA: *Aparentemente, a partir de* n *iteraciones, el "peor" de los casos resulta ser más rápido que el de un array con elementos al azar. ¿Quizás es una optimización por parte del SO, de GCC o del mismo procesador?*

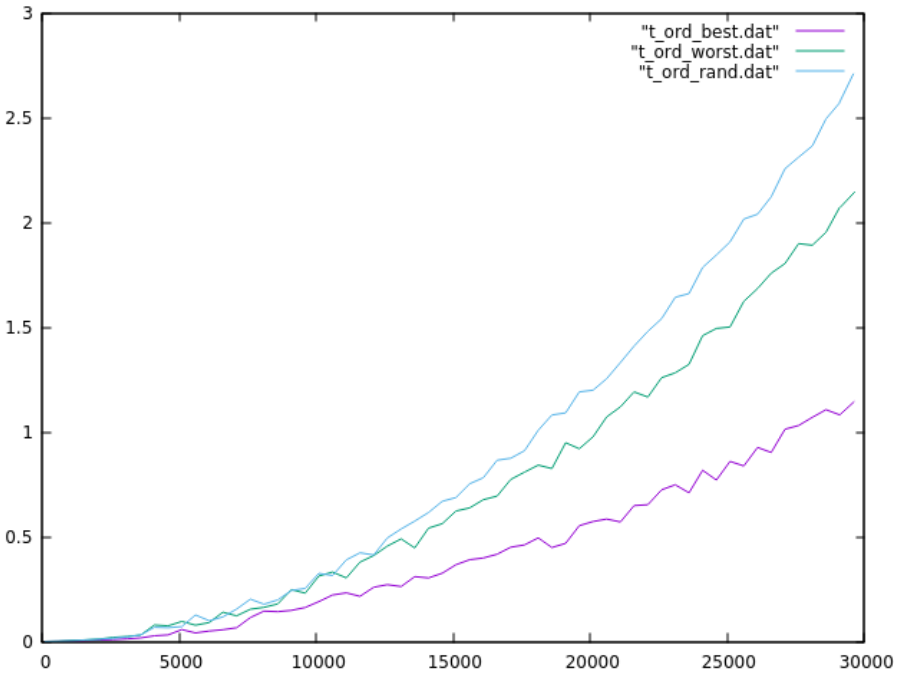
Ejercicio 5

Considerando que el array está ordenado, y que el algoritmo recorrería el array exactamente una vez para comprobar que ya esté ordenado o no, podemos decir que es de órden $O(n)$, ya que recorre todos los elementos exactamente una vez.

Figuras e Ilustraciones






"t_ord_best.dat"
"t_ord_worst.dat"
"t_ord_rand.dat"
3
2.5
2
1.5
1
0.5
0
0
5000
10000
15000
20000
25000
30000