Documentación Ejercicios Básicos
Tornillos y Tuercas

Alumno: Jorge Valenzuela García

Código Fuente completo del algoritmo por Divide y Vencerás:

```
void ordenarTornillos( const vector<int> &tornillos, const vector<int> &tuercas, vector<int>
&ordenado){


	// Creamos los tres conjuntos de datos:
	// mayores que el pivote, menores que el pivote
	// iguales que el pivote
	// El pivote sera la tuerca y tornillo elegidos
	vector<int> tuercas_mayores;
	vector<int> tuercas_menores;
	vector<int> tornillos_mayores;
	vector<int> tornillos_menores;


	// Cogemos un tornillo al azar y creamos la variable tuerca
	// la cual tendra el valor que sea igual a tornillo (tamaño)
	int tornillo = tornillos[rand() % tornillos.size()];
	int tuerca;


// SI PENSAMOS EN QUE ESTE ALGORITMO ES RECURSIVO, SABEMOS QUE POR EL
ORDEN
// DE LLAMADAS QUE HEMOS PUESTO, LOS MINIMOS SE ASIGANARAN PRIMERO,
QUEDANDO
// POR ULTIMO LAS LLAMADAS A LOS MAXIMOS. SI PASAMOS POR REFERENCIA UN
VECTOR
// RESULTADO DENOMINADO TORNILLOSORDENADOS O TUERCAS ORDENADAS, IRA
MODIFICANDOSE
// DE LA FORMA ANTES DESCRITA, QUEDANDO FINALMENTE TOTALMENTE
ORDENADOS


	// Caso base
	if (tornillos.size() == 1){ // Sabemos que tuercas tendra el mismo tamaño que tornillos
		tuerca = tornillo;
		ordenado.push_back(tuerca); // y el mismo valor que tornillo
	}


	else{
	// Dividimos segun el pivote (tornillos al azar) las tuercas en los tres conjuntos
	// antes creados
```

```
		for(int i = 0; i < tuercas.size() ; i++ ){
			if (tornillo > tuercas[i])
				tuercas_menores.push_back(tuercas[i]);

			else if(tornillo < tuercas[i])
				tuercas_mayores.push_back(tuercas[i]);

			else
				tuerca = tuercas[i]; // Guardamos la tuerca que es igual al tornillo
		}


	// Ahora ordenamos los tornillos en mayores y menores a la tuerca
		for(int i = 0; i < tornillos.size() ; i++ ){
			if (tuerca > tornillos[i])
				tornillos_menores.push_back(tornillos[i]);

			else if(tuerca < tornillos[i])
				tornillos_mayores.push_back(tornillos[i]);

			else
				tornillo = tornillos[i]; // Guardamos finalmente el tornillo igual a la
tuerca
		}



	// INICIO RECURSIVIDAD

	// Si los vectores tienen mas de un elemento dentro volvemos a llamar a los metodos
	// Finalmente aqui tambien delvolvemos la pareja tuerca tornillo encontrada en esta iteracion
		if (tornillos_menores.size() > 0){ // solo pongo tornillos porque tuercas tiene el
mismo tamaño
		// Las menores las ponemos al principio del vector
			ordenarTornillos(tornillos_menores , tuercas_menores, ordenado);
		}

		// Una vez estan las mas pequeñas ya insertadas con la condicion y orden anterior
		// pasamos a insertar las propias de esta traza del metodo
		ordenado.push_back(tuerca);

		if ( tornillos_mayores.size() > 0 ){
		// Las mayores las ponemos al final del vector
			ordenarTornillos(tornillos_mayores , tuercas_mayores, ordenado);
		}

	}
	// FIN RECURSIVIDAD


}
```

Hardware usado: Intel i7-6700HQ 2,60 GHZ , 2 GB RAM DDR4 en Virtualbox (8GB reales DDR4).

Sistema operativo: Linux Ubuntu 16.04 (VirtualBox).

Eficiencia teórica:

Vamos a comenzar a analizar la eficiencia de este algoritmo Divide y Vencerás.

Peor caso: Vemos que lo que realiza el algoritmo es escoger un tornillo al azar y dividir las tuercas en tres conjuntos, las mayores que ese tornillos, las menores, y la tuerca que es igual. Solo habrá una tuerca igual debido a que todos los tornillos y tuercas tienen tamaños diferentes, pero para cada tornillos tenemos su tuerca.

Una vez dividas las tuercas en los dos conjuntos mayores y menores y la tuerca encontrada, cogemos esa tuerca, y dividimos los tornillos en dos conjuntos, los mayores y los menores a la tuerca.

Una vez tenemos los dos conjuntos de mayores y menores de tornillos y de tuercas, vemos que tenemos una eficiencia de n*2 de base.

Sabemos que vamos a dividir los conjuntos de tornillos y tuercas hasta que solo tengamos un tornillo y una tuerca por conjunto.

Para resolver esto y ponerlos en orden, mi solución ha sido la siguiente:
Vamos a ir dividiendo como hemos dicho los conjuntos, una vez llegamos al caso base, añadimos la tuerca a un nuevo vector llamado ordenado, (partimos de que tornillos está ordenado, pero sino fuese así simplemente bastaría con añadir a un nuevo vector ordenado_tornillos los tornillos en el mismo momento que añadimos a ordenado las tuercas).

Bien, ahora hay que controlar las llamadas a los métodos y como se acumulan en la pila. Para esto lo realizado ha sido llamar primero a los menores, lo que finalmente me dará el tornillo y tuerca más pequeños, y añadirlos al vector ordenado, seguidamente, antes de llamar a los métodos de los conjuntos mayores, añado la tuerca y tornillo encontrados en el método en sí, y finalmente se llama a los conjuntos mayores, lo que hará que lo último en ser añadido será el tornillo y tuerca más grades. Todo esto finalmente hace que queden ordenados dentro de un vector ordenado todas las tuercas. Las llamadas si las viésemos tendrían la forma de un ABB, y para almacenarlos ordenadamente es como si lo recorriésemos en inorden.

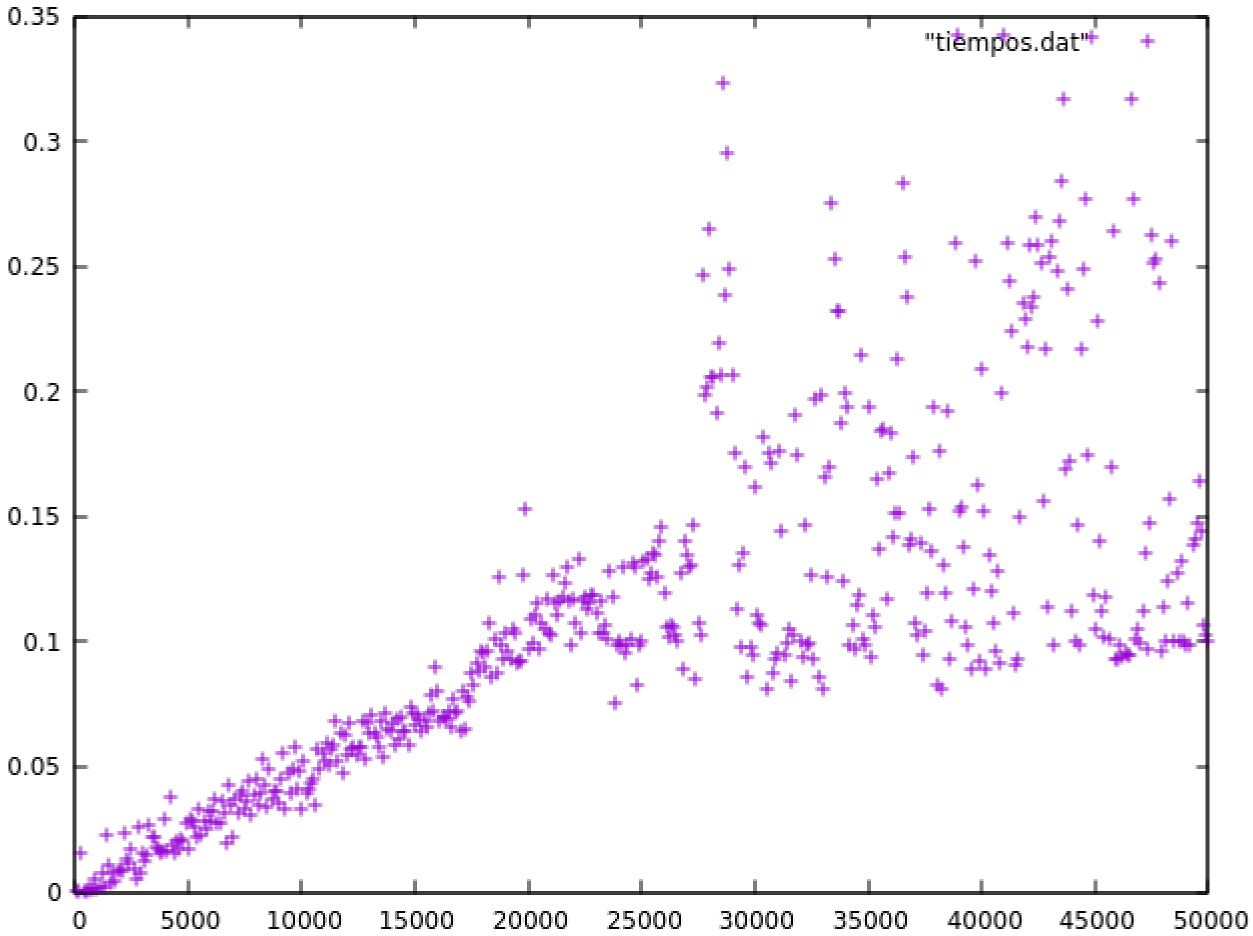
Dicho esto, el peor caso sabemos que sería aquel en el que siempre cogemos el tornillo de mayor o menor tamaño, de modo que la recursividad no valdría casi de nada, quedándonos una eficiencia de n*n/2, lo que se traduce en un orden de eficiencia de **$O(n^2)$**.

Mejor caso: En el mejore de los casos vemos que siempre escogeríamos el tornillo de la mitad, creando así dos conjuntos mayores y menores justo de la mitad de tamaño, lo cual nos da un orden de eficiencia de **$\Omega(n \log_2 n)$.**

Caso promedio: Viendo lo anterior sabemos que el caso promedio será, **$\Theta(n \log_2 n)$.**

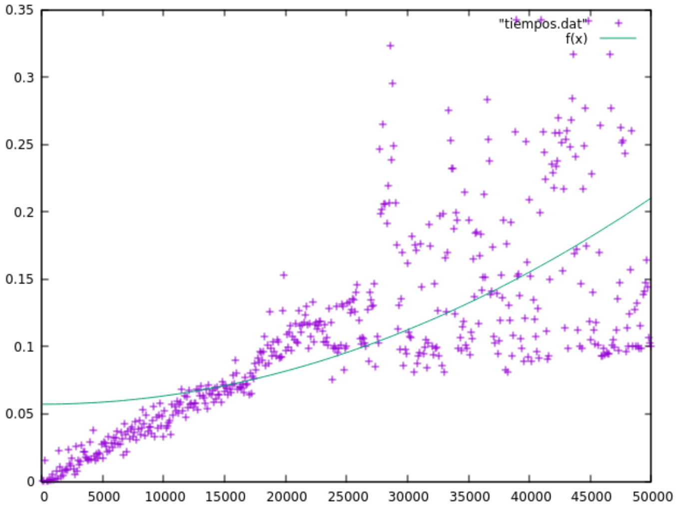
Eficiencia Empírica y Gráfica:

Los resultados obtenidos de forma empírica son los siguientes:

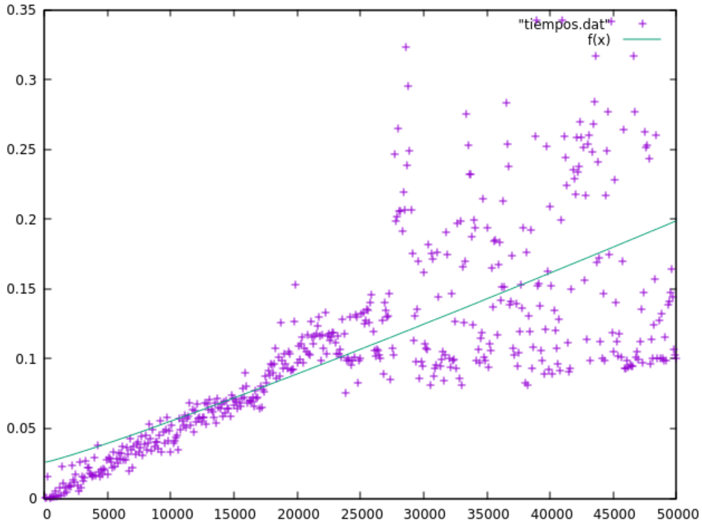


Los ajustes son:

Para cuadrática:
a = 6.12569e-11
b = 0.0572634



Para logarítmica:
a = 2.21806e-07
b = 0.025733



Como ocurrió con el caso del ejercicio de la moda, los ajustes no se aplican correctamente a los datos obtenidos mediante gnuplot, aun así los adjunto por añadir algo de información visual al documento.