Documentación Ejercicios Básicos
Máximo y Mínimo

Alumno: Jorge Valenzuela García

Código Fuente completo del algoritmo MAXMIN por Divide y Vencerás:

```
pair<int,int> Max_Min(const vector<int> & v){

        pair<int,int> par;          // Declaramos la variable pair
        int max;                    // Declaramos nuestra variable max para guardar el maximo
        int min;                    // Declaramos nuestra variable min para guardar el minimo
        vector<int> izquierda;      // Declaramos el vector que contendra una mitad
        pair<int,int> pderecha;     // Declaramos el par max-min del vector de la derecha
        vector<int> derecha;        // Declaramos el vector que contedra la otra mitad
        pair<int,int> pizquierda;   // Declaramos el par max-min del vector izquierda


        if (v.size() == 1){
                par.first = v[0];
                par.second = par.first;
                return par;
        }

        else{
                // Hallamos la mitad y dividimos el vector en dos
                int mitad = v.size()/2;

                for (int i = 0; i < mitad; i++)
                        izquierda.push_back(v[i]);

                for(int j = mitad; j < v.size() ; j++)
                        derecha.push_back(v[j]);


                // Recursivamente hallamos el par minimo y maximo de los vectores divididos
                pderecha = Max_Min(derecha);
                pizquierda = Max_Min(izquierda);


                // Para el maximo
                if (pderecha.first > pizquierda.first)
                        max = pderecha.first;
                else
                        max = pizquierda.first;

                // Para el minimo
                if (pderecha.second < pizquierda.second)
```

```
                min = pderecha.second;
            else
                min = pizquierda.second;

            par.first = max;
            par.second = min;

            return par;
        }
}
```

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: Para comenzar sabemos que dividimos el vector en 2 en cada llamada al método, hasta que finalmente nos encontramos en el caso base donde simplemente se nos devuelve el par max/min, por tanto, una vez se nos devuelve el par max/min del caso base, vamos comparando si ese mínimo es menor que el mínimo que se nos devuelve desde la otra llamada del método, de modo que simplemente comprobando los pares máximos y los mínimos que nos van devolviendo, encontramos la solución.

El mayor coste de esto, como podemos ver en el código, es el recorrer el vector en cada llamada al método, lo cual desde un comienzo tiene como mínimo un orden de eficiencia de O(n).
Ahora vemos que el tamaño de estos vectores es cada vez más pequeño, de hecho, es cada vez la mitad del anterior, lo cual, sabemos que como resultado tenemos un orden de eficiencia de $O(\log_2 n)$.

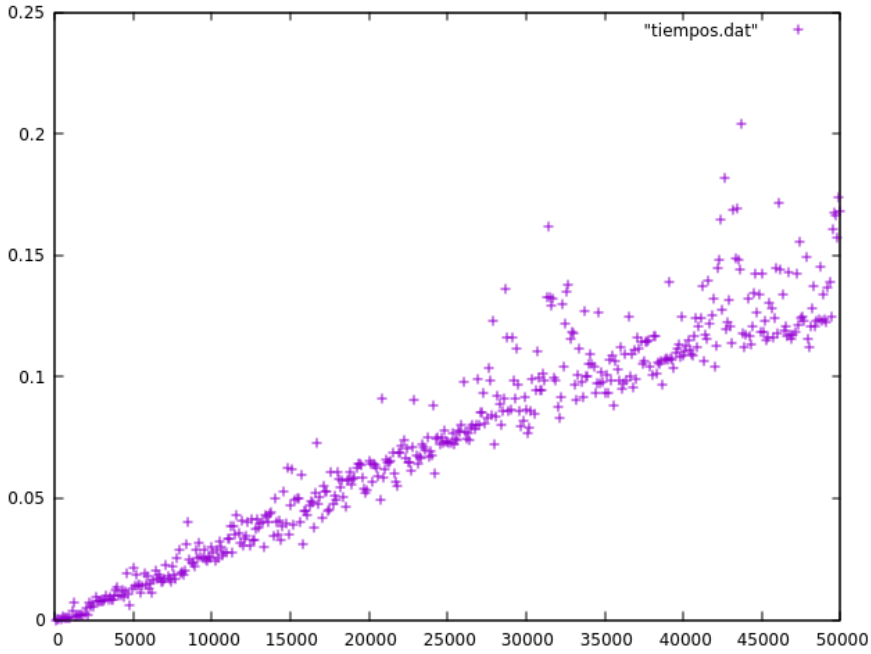
Así que como hemos visto, finalmente obtenemos una eficiencia en el peor caso de $\mathbf{O(n \log_2 n)}$.

Mejor caso: En el mejor de los casos, tendríamos el mismo orden de eficiencia, porque pese a que encontremos el par max/min desde un comienzo, tenemos que seguir recorriendo y comprobando los resultados que nos llegan desde las otras llamadas de los métodos llamados en la recursividad. Finalmente en el mejor de los casos también tenemos un orden de $\mathbf{\Omega(n \log_2 n)}$.

Caso promedio: Viendo lo anterior sabemos que el caso promedio es igual, $\mathbf{\Theta(n \log_2 n)}$.

Eficiencia Empírica y Gráfica:

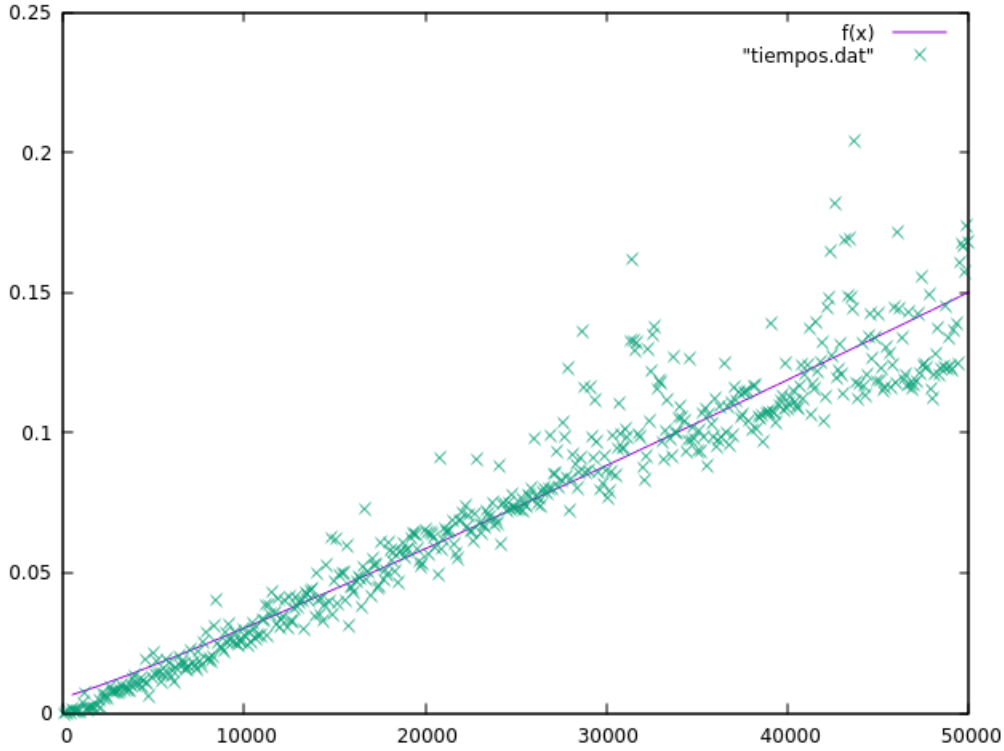
Los resultados obtenidos de forma empírica son los siguientes:



Los ajustes son:

a = 1.85215e-07
b = 0.0056495

El resultado obtenido con la función superpuesta es la siguiente:



Moda

Código Fuente completo del algoritmo MAXMIN por Divide y Vencerás:

```
// Metodo Max_Min. Devuelve al par de elementos Max/Min del vector pasado como parametro
pair <int,int> Moda(const vector<int> & v){

	vector<int> heterogeneosmenor;
	vector<int> homogeneos;
	vector<int> heterogeneosmayor;
	int pivote;
	pair<int,int> moda;
	pair<int,int> modamenor;
	pair<int,int> modamayor;
	bool encontrada = false;	// Para indicar si ya hemos encontrado la moda

	if (v.size() <= 0){
		moda.first = -1;
		moda.second = -1;
		return moda;
	}
	else
		pivote = v[0];


	// Creamos los tres conjuntos
	for (int i=0; i < v.size(); i++){
		if(v[i] == pivote){
			homogeneos.push_back(v[i]);
		}
		else{
			if(v[i] < pivote)
				heterogeneosmenor.push_back(v[i]);
			else
				heterogeneosmayor.push_back(v[i]);
		}
	}

	// Una vez dividido el conjunto vamos a ver que conjunto es el mas grande

	if(heterogeneosmenor.size() >= heterogeneosmayor.size())
		if(homogeneos.size() >= heterogeneosmenor.size())
			encontrada = true;
	else
		if(homogeneos.size() >= heterogeneosmayor.size())
			encontrada = true;


	// RECURSIVIDAD
```

```
        if (encontrada){
                moda.first = pivote;
                moda.second = homogeneos.size();
                return moda;   // Si tenemos la moda devolvemos el pivote (moda.first)
        }                      // y su cardinalidad (moda.second), es decir,
                               // el elemento de homogeneos y su cardinalidad
        else{
                if (heterogeneosmayor.size() > 0)
                        modamayor = Moda(heterogeneosmayor);
                if (heterogeneosmenor.size() > 0)
                        modamenor = Moda(heterogeneosmenor);
        }

        // Fin RECURSIVIDAD

        if (modamayor.second > modamenor.second) // Comprobamos que moda es mas comun, es
                return modamayor;                // decir, comprobamos las cardinalidades
        else
                return modamenor;

}
```

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 y explicación del método:

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

Peor caso: Partimos de que sabemos que tenemos un conjunto de valores representados mediante un vector y que tenemos que hallar la moda, es decir, el elemento que mas se repite.

Si no usáramos la recursividad (Divide y Vencerás) sabemos que tardaría $O(n^2)$, ya que por cada elemento del vector hay que compararlo con el resto de elementos del vector, eso mediante codificación sabemos que en el peor caso serían dos bucles anidados.

Ahora bien, mediante la recursividad lo que hacemos es los siguiente, cogemos el conjunto y escogemos un elemento al azar, denominado pivote (en mi caso el elegido es siempre el primer elemento del conjunto), y seguidamente dividimos el conjunto en otros tres, los mayores al pivote, los menores al pivote y los iguales al pivote. Si llega el momento en que el conjunto de elementos iguales al pivote es mayor que alguno de los otros dos significa que ya tenemos la moda del conjunto.

Bien, una vez dicho esto, analizando el código vemos que lo primero que hacemos es crear los tres conjuntos y comprobar si ya hemos encontrado la moda, sino mandamos a buscar la moda en los otros dos conjuntos creados (mayores y menores).

Una vez hecho esto supongamos que nos llegan de ambos conjuntos un elemento moda. Ahora necesitamos saber cual de esos dos elementos es la moda de verdad, por lo que he tomado lo decisión de que siempre se devuelva el valor de la moda junto con el número de veces que ha aparecido ese elemento en el conjunto. Por lo que finalmente nos quedamos con aquella moda con mayor número de apariciones.

Si llegamos al caso base, en el cual solo tenemos un elemento, y que seguro que coincide con el pivote porque es el mismo, se devolverá ese elemento junto con una el número de veces que aparece, es decir, 1, por lo que frente a cualquier otra llamada al método que nos devuelva otro número optando a ser la moda, pero que haya aparecido mas de una vez, hará que el valor del caso base se desprecie. Aun así es necesario poner el caso base, porque en caso de que no tengamos moda (todos los elementos aparecen una vez) se nos devolverá un elemento del conjunto, y, por el orden de llamadas, será el número mayor del conjunto, debido a que se le pregunta antes al conjunto mayor que al menor.

Explicado esto vemos que en el peor de los casos tenemos un conjunto con todos los elementos diferentes. Para la primera llamada al método sabemos que se ejecuta n veces. Ahora toca ver que ocurre con la recursividad.

Dejo claro que el problema que tiene mi solución, es que si por un casual, todos los números generados aleatoriamente quedan en orden de menor a mayor o de mayor a menor, lo que tiene probabilidad que tiende a 0, el orden de eficiencia sería de $O(n^2)$. Esto es debido a que el pivote que yo escojo es el primer elemento de conjunto, pero simplemente cambiando esa sentencia por pivote = v[rand() % v.size()] tenemos la certeza de que aunque estén ordenados, vamos a escoger un elemento con el que los conjuntos mayor y menor van a tener más elementos. Aclaro esto aquí porque no me he dado cuenta antes y ya tengo los resultados obtenidos, pero creo correcto aclararlo.

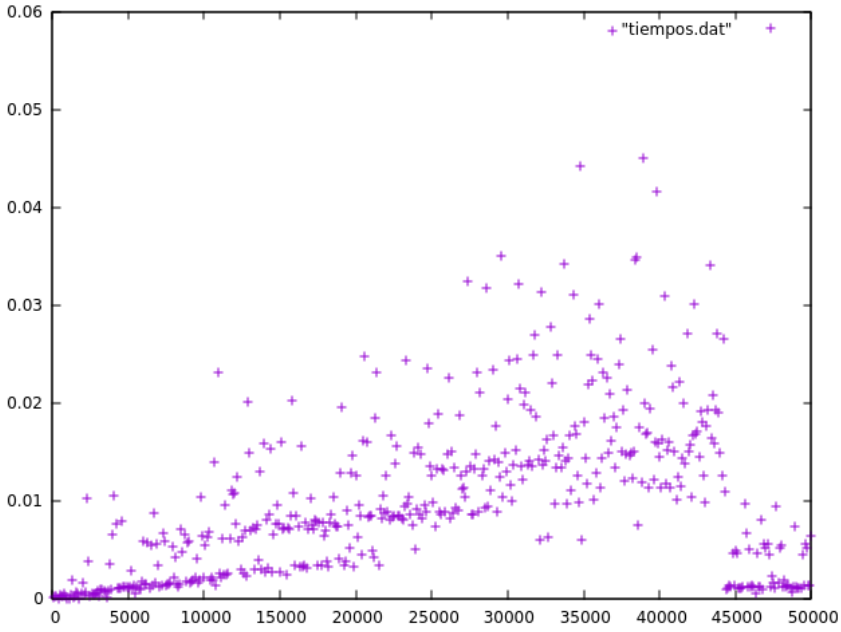
Una vez aclarado esto, teóricamente el peor de los casos es el antes visto, pese a que la probabilidad tienda a 0, (finalmente veremos que tiende más a un orden de **$O(n \log_2 n)$).**
El caso antes visto, para aclararlo, es aquel en el que, en cada iteración, escogemos el mayor o menor elemento del conjunto. Sabiendo que todos los elementos son diferentes, si esto ocurre, se ejecutará n*n/2, lo cual finalmente nos da un orden de **$O(n^2)$**.

Mejor caso: Podemos ver que en la primera iteración, podemos encontrar la moda, por tanto solo necesitamos una ejecución del bucle de búsqueda de la moda. Esto directamente nos da un orden de eficiencia de **$\Omega(n)$.**

Caso promedio: Viendo lo anterior sabemos que el caso promedio es igual, **$\Theta(n \log_2 n)$.** Esto es debido a que como hemos visto el caso intermedio rondará el que siempre se escoja como pivote un elemento que esté por el medio, y aunque no sea justo la mitad, vemos que siempre vamos a ir creando dos conjuntos donde buscar la moda.

Eficiencia Empírica y Gráfica:

Los resultados obtenidos de forma empírica son los siguientes:

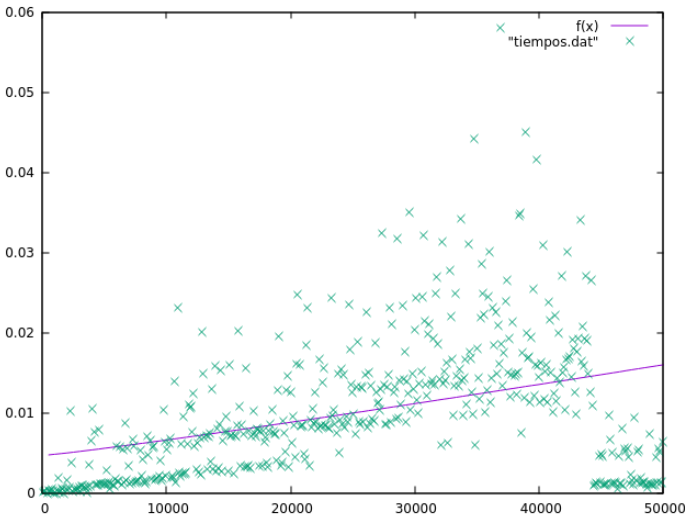


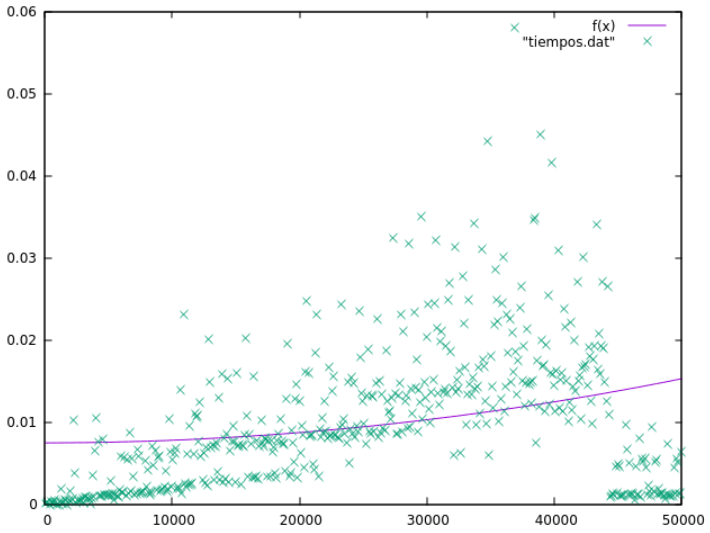
Podemos ver que la función es bastante dispersa, y que describe con ciertos valores una función cuadrática, y con otros una logarítmica.

Los ajustes son:

Para n * $\log_2$ n:
a = 1.45027e-08
b = 0.00465864

Para n2:
a = 3.12363e-12
b = 0.00745714





Los ajustes no se están realizando correctamente con gnuplot, pero aún así los adjunto por añadir información visual al análisis.