Práctica 2: Elemento en su posición

Alba Moreno Ontiveros, Sergio Díaz Rueda, Jesús Mesa González, Antonio Javier Rodríguez Pérez

20/04/2016

Resumen del algoritmo

Dado un vector ordenado (de forma no decreciente) de números enteros vector, todos distintos, el objetivo es determinar si existe un índice i tal que vector[i] = i, en caso de existir varios crear un vector soluciones, que contendrá los elementos que coinciden con su posición.

Algoritmos Usados

El **algoritmo usual** que nosotros hemos hecho es el siguiente:

```
int EnSuPosicionFBruta(int *vec, int num_elementos){
    int i = 0;
    int resultado;

    while(vec[i] != i && i < num_elementos){
        i++;
    }
    if(vec[i] == i)
        resultado = vec[i];
    else
        resultado = -1;
    return(resultado);
}
```

Como queremos comprobar si existe un elemento en el vector (con uno basta) que coincida con el índice de su posición en el vector usamos para ello un bucle while
La última comprobación es necesaria, ya que sin ella habría un gran fallo en el supuesto de haber llegado al final del vector sin encontrar ningún elemento que cumpla vec[i] = i, en cuyo caso nos devolvería el último elemento del vector.
Con el if evitamos eso, nos devuelve un elemento si cumple la condición y sino nos devuelve -1, “error”.

El **algoritmo Divide y Vencerás** que hemos aplicado es:

```
void busqueda(int v[],int inicio, int fin){

        if((fin-inicio)<2){
                if(v[inicio]==inicio){
                        cout<<"encontrado"<<endl;
                }
        }

        else{
                int medio=(fin+inicio)/2;
                if(medio==v[medio]){
                        cout<<"encontrado"<<endl;
                }
                else{
                        if(medio<v[medio]){
                                busqueda(v,inicio,medio);
                        }
                        else{
                                busqueda(v,medio,fin);
                        }
                }
        }

}
```

En este caso hemos implementado una solución recurrente, en la cual si no se da el caso base, dividimos el vector por la mitad. A continuación comprobamos si el elemento de la mitad es igual a su índice en el vector. Si lo es hemos acabo. Si no comprobamos si este elemento es mayor o menor que su índice.

Si es mayor nos olvidamos de la parte derecha de del vector, puesto que si ese elemento no está en su posición los siguientes elementos mayores que él, que no se pueden repetir, tampoco lo estarán. Si es menor estamos ante la misma situación de antes, pero esta vez nos olvidamos de la parte de la izquierda.

Finalmente aplicamos la misma función de manera recursiva para una de las mitades con la que nos quedamos.

Caso a

Suponemos que todos los elementos del vector son distintos, no hay ninguno repetido. Procedemos de dos formas;

1.- Usando el algoritmo usual, el cual está conformado por un ciclo while, y por tanto su eficiencia teórica es O(n).

$$T(n) = a0*n+a1$$

Para el cálculo de las constantes ocultas hacemos una tabla con 100 valores, desde 10 hasta 1000 con intervalos de 10.

N	tiempo(seg)
10	7.57e-07
1010	1.2038e-05
2010	1.3613e-05
3010	1.0645e-05
4010	9,40E-003
5010	1.1705e-05
6010	1.3973e-05
7010	1.6359e-05
8010	1.9421e-05
9010	2.1554e-05
90010	0.000213834
91010	0.000212663
92010	0.000220403
93010	0.000219728
94010	0.000223687
95010	0.000226403
96010	0.000293485
97010	0.000228904
98010	0.000232096
99010	0.000234513

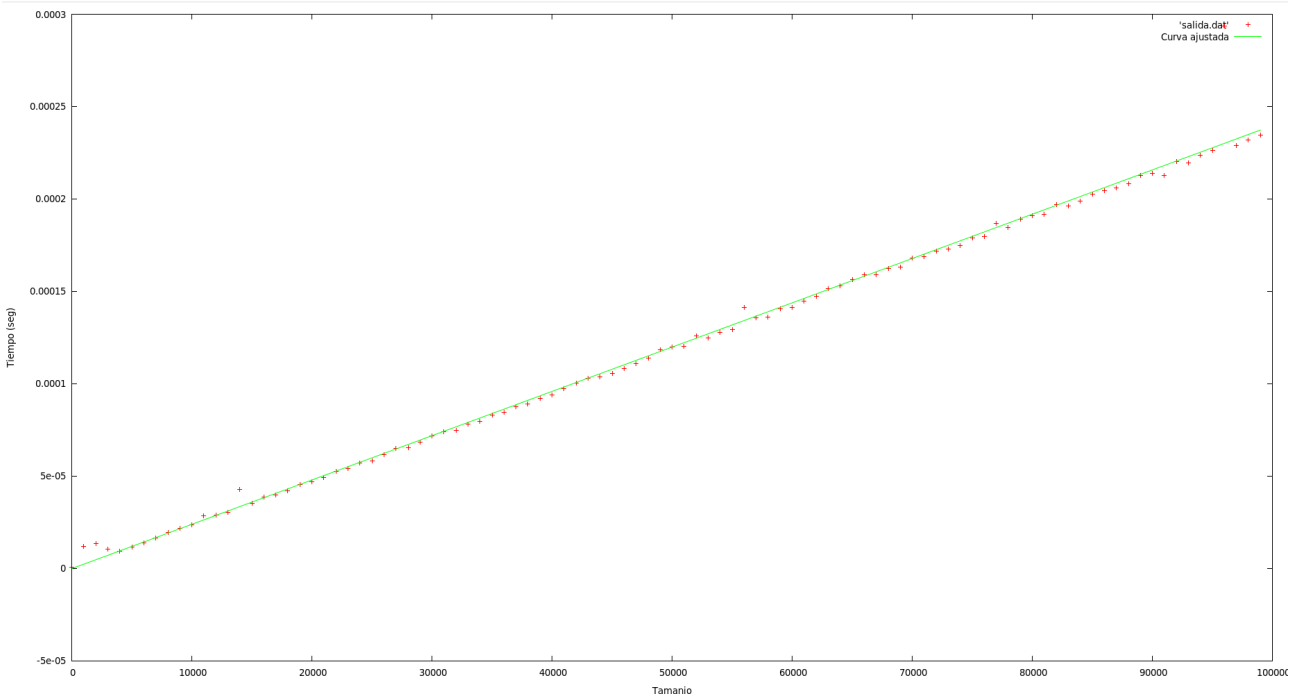
Por tanto:

Final set of parameters Asymptotic Standard Error
========== ==========

a0 = 2.40191e-09 +/- 3.938e-11 (31.05%)
a1 = -3.62322e-07 +/- 2.268e-08 (11.93%)

$$T(n) = 2.40191e\text{-}09 * n - 3.62322e\text{-}07$$

Una vez realizado el ajuste y la recta obtenemos la siguiente gráfica:



2.-Usando el algoritmo DV

La eficiencia teórica viene dada por la siguiente recurrencia:

$T(n) = T(n/2) + T(1) + T(1)$

$T(n) = C * n$

Por tanto la eficiencia teórica es O(n). Hacemos una tabla con el valor de n y los distintos tiempos de ejecución.

N	tiempo(seg)
10	3.25e-07
1010	3.9e-07
2010	3.95e-07
3010	2.45e-07
4010	2.45e-07
5010	2.17e-07
6010	2.67e-07
7010	2.69e-07
8010	2.65e-07
9010	2.13e-07
90010	7.63e-07
91010	6.03e-07
92010	6.07e-07
93010	6.06e-07
94010	5.81e-07
95010	2,75E-003
96010	6.05e-07
97010	7.18e-07
98010	6,00E-007
99010	1.04e-06

Hacemos una tabla con los valores de n y su correspondiente tiempo.

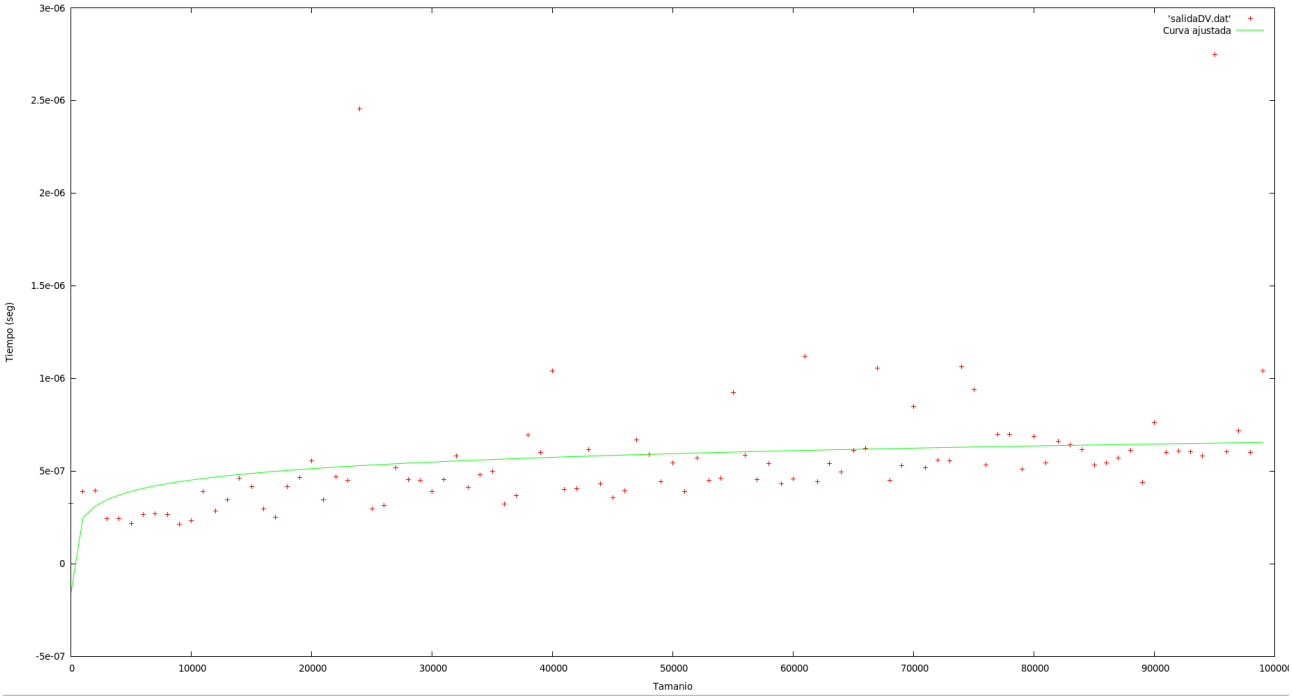
Con gnuplot hacemos un ajuste logaritmico y calculamos las constante ocultas.

Final set of parameters Asymptotic Standard Error
========== ==========

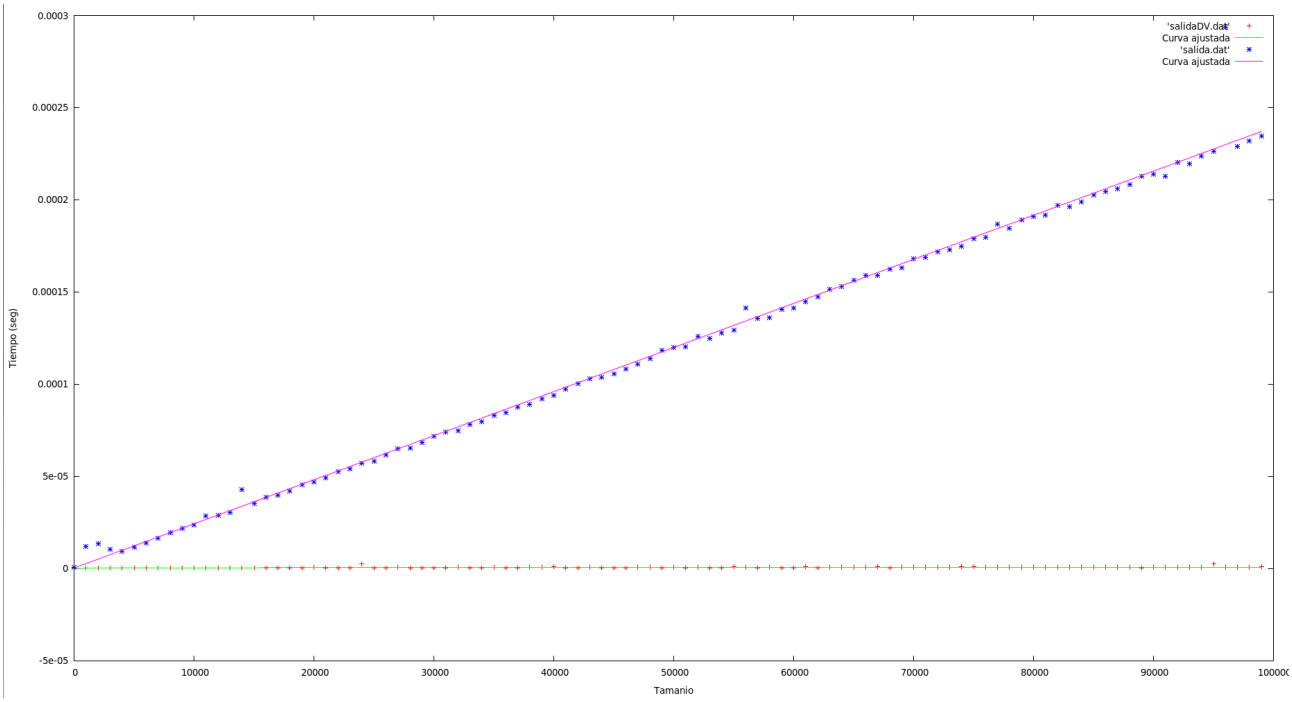
a0 = 4.66714e-12 +/- 2.226e-12 (7.854%)
a1 = 3.3097e-07 +/- 2.488e-11 (7.27%)

$$T(n) = 4.66714e\text{-}12 * \log(n) + 3.3097e\text{-}07$$

Hacemos la gráfica correspondiente con el ajuste:



3.- Comparación de gráficas:



Caso b

El vector contiene elementos repetidos.

Procedemos como en el apartado a. Para este caso, hemos usado una clase externa, Random.h y MyRandom.cpp, que hemos diseñado, lo que hace es generar números aleatorios entre un mínimo y un máximo.

1.- Usando el algoritmo usual, el cual está conformado por un ciclo for, y por tanto su eficiencia teórica es O(n).
Queremos comprobar si que haya elementos repetidos supone algo a nivel de eficiencia.

$$T(n) = a0*n+a1$$

Hemos hecho una tabla con 100 elementos, empezando en 10 hasta 1000 de 10 en 10.

N	tiempo(seg)
10	5.38e-07
1010	8,85E-003
2010	7,18E-003
3010	7,09E-003
4010	9,37E-003
5010	1.1704e-05
6010	1.4002e-05
7010	1.6416e-05
8010	1.8666e-05
9010	2.0935e-05
90010	0.00021299
91010	0.000216527
92010	0.000218996
93010	0.000221531
94010	0.000220143
95010	0.000224754
96010	0.000227073
97010	0.000228354
98010	0.000232438
99010	0.000235563

Realizamos el cálculo de las constantes ocultas;

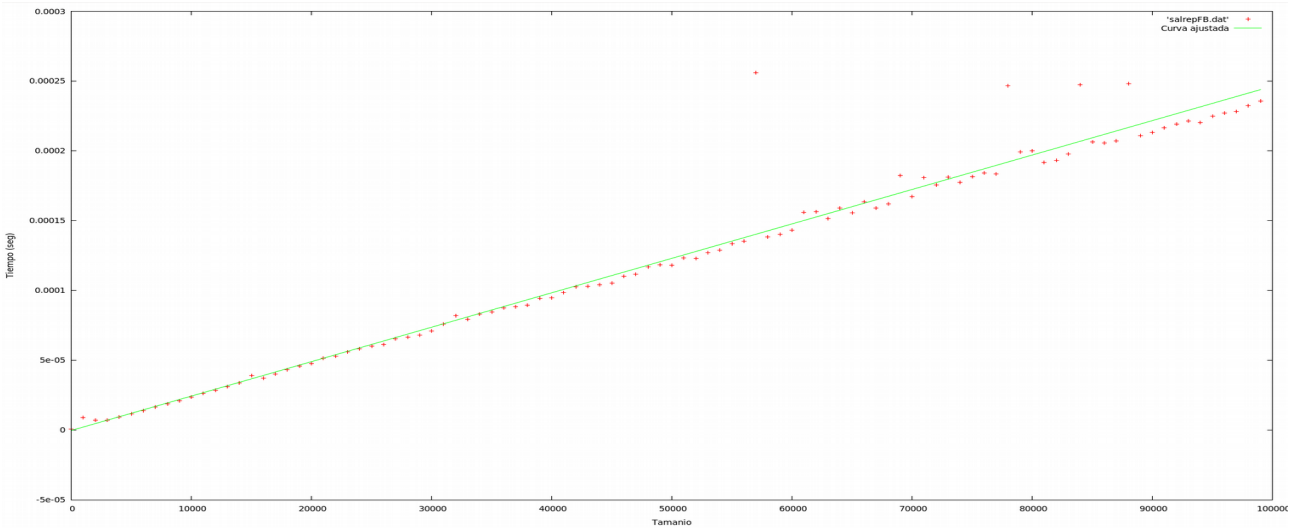
Final set of parameters	Asymptotic Standard Error
==========	==========

a0 = 2.46666e-09 +/- 3.511e-11 (46.68%)
a1 = -3.62322e-07 +/- 2.022e-08 (9.486%)

Y obtenemos que la nueva ecuación es:

$$T(n) = 2.46666e\text{-}09*n - 3.62322e\text{-}07$$

Una vez realizado el ajuste y la recta obtenemos la siguiente gráfica:



2.-Usando un algoritmo DV

En este caso hemos implementado una solución recurrente, en la cual si no se da el caso base, dividimos el vector en dos mitades y llamamos a la misma función de forma recursiva para cada una de las dos mitades.

Siempre antes de dividir el vector, comprobamos si el primer elemento del vector que nos han enviado es igual a inicio que es su indice en dicho vector. Si coinciden ya hemos encontrado un elemento en el vector cuyo valor coincide con su índice y ponemos la variable encontrado a true para evitar seguir comprobando el vector, puesto que ya hemos alcanzado una solución.

Usamos el algoritmo DV con un vector con elementos repetidos.

```
void busqueda2(int v[],int inicio, int fin){

        if(!encontrado){

                if((fin-inicio)<2){
                        if(v[inicio]==inicio){
                                cout<<"encontrado"<<endl;
                                encontrado=true;
                        }
                }

                else{

                        busqueda2(v,inicio,(fin+inicio)/2);
                        busqueda2(v,(fin+inicio)/2,fin);
                }
        }
}
```

La eficiencia teórica viene dada por la siguiente recurrencia:

$T(n) = T(n/2) + T(1) + T(1)$

$T(n) = C * n$

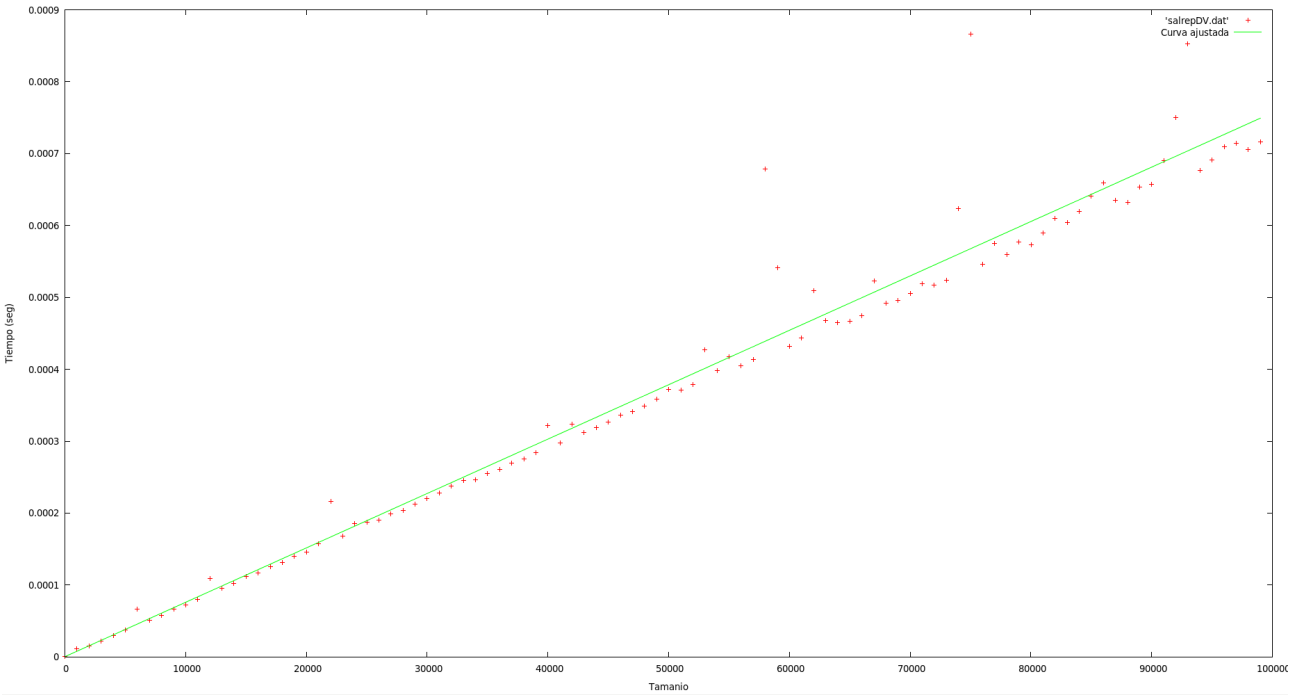
Por tanto la eficiencia teórica es O(n). Hacemos una tabla con el valor de n y los distintos tiempos de ejecución.

N	tiempo(seg)
10	3.29e-07
1010	1.1117e-05
2010	1.5515e-05
3010	2,23E-002
4010	2.9718e-05
5010	3.7132e-05
6010	6.6842e-05
7010	5.1083e-05
8010	5.8283e-05
9010	6.6556e-05
90010	0.000656907
91010	0.000689816
92010	0.000750199
93010	0.000852759
94010	0.000676624
95010	0.000691473
96010	0.0007091
97010	0.000713933
98010	0.000705891
99010	0.00071598

Con gnuplot hacemos un ajuste logaritmico y calculamos las constante ocultas.

Final set of parameters Asymptotic Standard Error
========== ==========

a0 = 7.56415e-09 +/- 2.226e-12 (7.854%)
a1 = 3.30905e-07 +/- 2.488e-11 (7.27%)



T(n) = 7.56415e-09*n + 3.30905e-07

3.- Comparación de gráficas: