Algoritmos Divide y Vencerás

El entero en su misma posición

0.Introducción

Los procesadores de cada uno de los integrantes son los siguientes:

- Ana García Muñoz (intel core i-7 8ª Gen)
- José María Ramírez González (intel core i-7 8ª Gen)
- Manuel Sánchez Pérez (intel core i-5 8ª Gen)
- Víctor Gutiérrez Rubiño. (AMD Ryzen 5 2600X)

El trabajo realizado ha sido repartido equitativamente entre los cuatro miembros del equipo (25% cada uno).

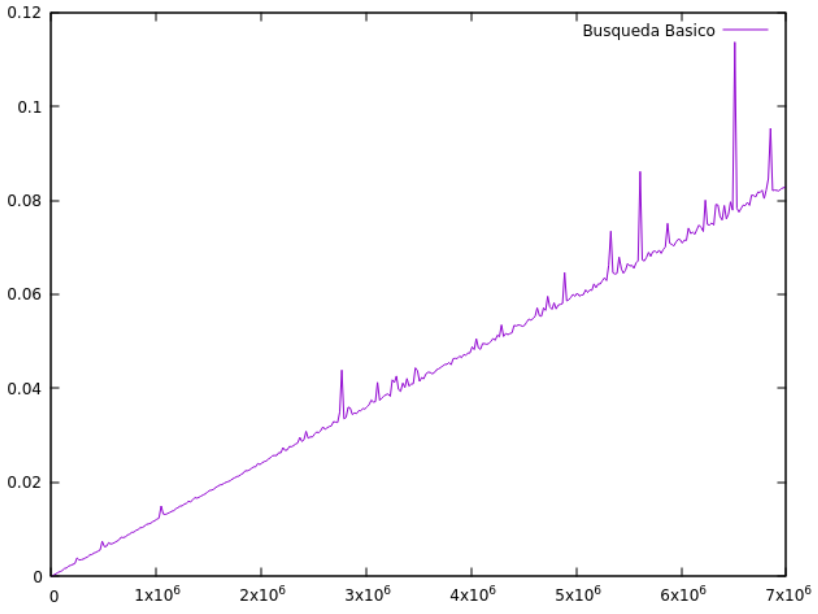
1.Estudio del algoritmo básico

```
for (it = myvector.begin(); it != myvector.end(); ++it)
{
  if (cont == *it)
  {
    encontrado = true;
    pos = cont;
  }
  cont++;
}
```

Eficiencia teórica

Como se puede ver en la imagen anterior, este algoritmo consta tan sólo de un bucle for que va recorriendo los elementos de un vector hasta encontrar un elemento que coincida con el índice de su posición.
En el peor de los casos, no habrá ninguna coincidencia (no existe), y el algoritmo habrá recorrido el vector entero. Por ello, podemos concluir que es un algoritmo O(n).

Eficiencia empírica




```
6470000 0.079648
6490000 0.077883
6510000 0.113627
6530000 0.078314
6550000 0.077481
6570000 0.078256
6590000 0.078972
6610000 0.078858
6630000 0.079562
6650000 0.078907
6670000 0.081097
6690000 0.081023
6710000 0.080717
6730000 0.081759
6750000 0.081644
6770000 0.082143
6790000 0.080381
6810000 0.082195
6830000 0.084548
6850000 0.095299
6870000 0.082068
6890000 0.082172
6910000 0.08208
6930000 0.082004
6950000 0.082432
6970000 0.08264
```

Hemos ido tomando datos de 20.000 en 20.000, hasta un tamaño de vector de siete millones como podemos denotar los tiempos son muy cortos, y además también podemos resaltar que no es completamente lineal en parte práctica ya que hay en diferentes zonas de la gráfica donde hay pequeños picos.

Eficiencia Híbrida

Vamos a comprobar que el algoritmo es O(n). Para ello, definiremos una función para ajustar a los datos. En este caso,la función es lineal, así que la definiremos de la siguiente manera:

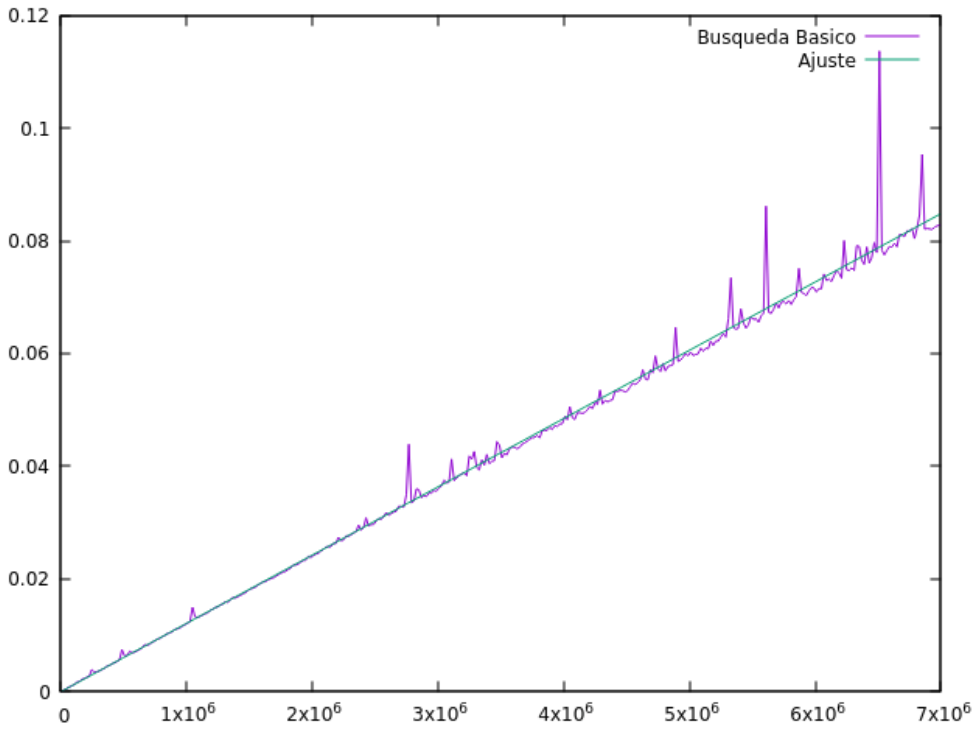
$$T(n) = a_0 \times n$$

Para obtener las constantes ocultas, utilizaremos gnuplot para realizar la regresión, obteniendo los siguientes datos:

```
Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a0                 = 1.21135e-08   +/- 3.319e-11    (0.274%)
```

Por lo que la función ajustada, que podemos usar para calcular el tiempo de ejecución del algoritmo para una entrada de tamaño n, nos queda:

$$T(n) = 1.21135 \times 10^{-8} \times n$$



Esta gráfica nos muestra el ajuste de datos.

2.Estudio del algoritmo Divide y Vencerás

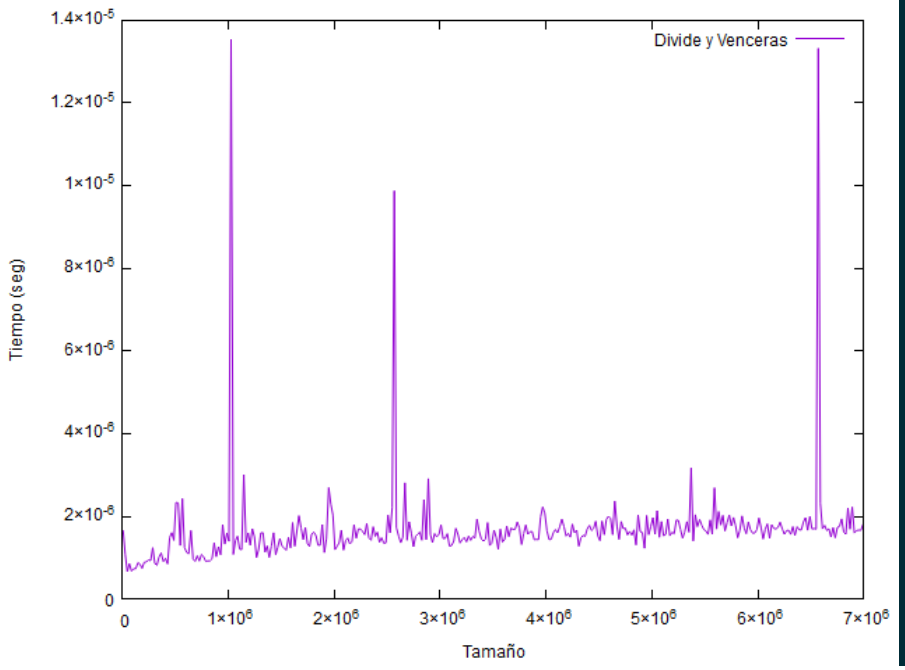
```
void divide_venceras(vector<int> &vect, int inicio, int fin) {
    int tam = fin - inicio + 1;
    if (tam > 10 && !encontrado) {
        int half = tam/2;
        half += inicio;
        if (vect.at(half) == half) {
            encontrado = true;
            posicion = half;
        } else if(vect.at(half) > half){
            divide_venceras(vect, inicio, half);
        } else{
            divide_venceras(vect, half, fin);
        }

    } else if(!encontrado){
        lineal(vect, inicio, fin);
    }
}
```

Eficiencia teórica

Como se puede ver en la imagen anterior, este algoritmo consta de una función recursiva que va quedándose con mitades del vector, teniendo en cuenta que está ordenado en orden ascendente, en caso de que queden pocos elementos se llama a una función auxiliar que hace una búsqueda básica.
En el peor de los casos, no habrá ninguna coincidencia (no existe), y el algoritmo habrá recorrido un fragmento de log(n). Por ello, podemos concluir que es un algoritmo O(log (n)).

Eficiencia empírica




```
6130000 1.801e-06
6150000 1.713e-06
6170000 1.699e-06
6190000 1.735e-06
6210000 1.864e-06
6230000 1.698e-06
6250000 1.57e-06
6270000 1.657e-06
6290000 1.668e-06
6310000 1.568e-06
6330000 1.756e-06
6350000 1.543e-06
6370000 1.752e-06
6390000 1.679e-06
6410000 1.676e-06
6430000 1.869e-06
6450000 1.972e-06
6470000 1.669e-06
6490000 2e-06
6510000 1.699e-06
6530000 1.708e-06
6550000 1.695e-06
6570000 1.331e-05
```

Ejecutando el programa hemos cogido muestras de 20.000 en 20.000 hasta siete millones

Eficiencia Híbrida

Vamos a comprobar que el algoritmo es O(log(n)). Para ello, definiremos una función para ajustar a los datos. En este caso,la función es lineal, así que la definiremos de la siguiente manera:

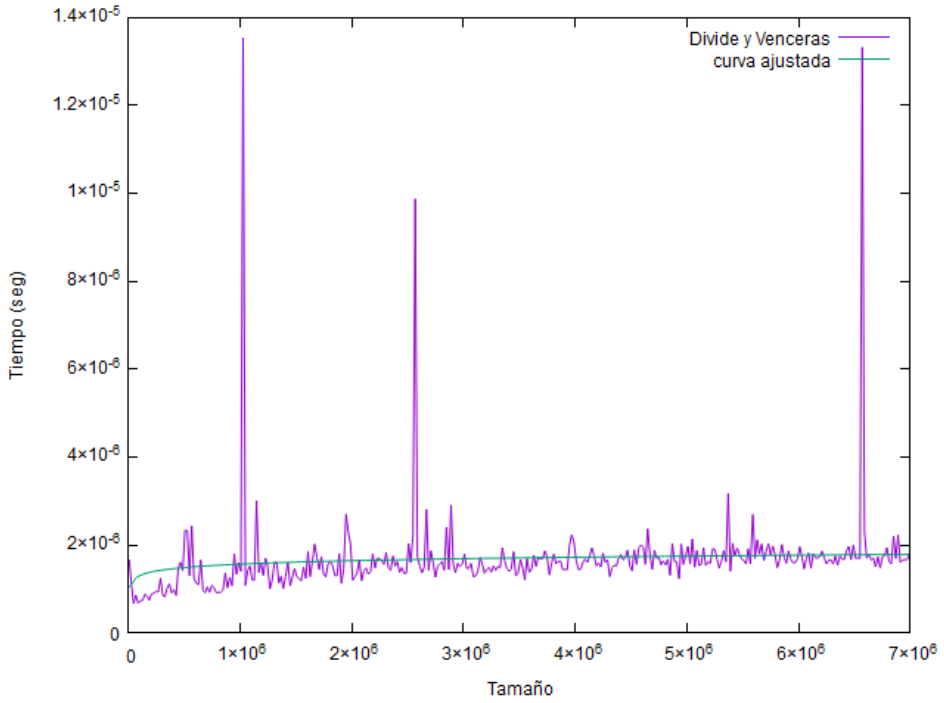
$$T(n) = a_0 \times log(n)$$

Para obtener las constantes ocultas, utilizaremos gnuplot para realizar la regresión, obteniendo los siguientes datos:

```
Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a0              = 1.13383e-07      +/- 3.774e-09    (3.329%)
```

Por lo que la función ajustada, que podemos usar para calcular el tiempo de ejecución del algoritmo para una entrada de tamaño n, nos queda:

$$T(n) = 1.13383 \times 10^{-7} \times log(n)$$



3.Comparativa de ambas ejecuciones

Como vemos, el tiempo del algoritmo básico es mucho mayor al tiempo del algoritmo de divide y vencerás, hasta el punto de que no se aprecia en la gráfica de comparación (está en el 0, ya que el orden es 3 veces menor).