UNIVERSIDAD DE GRANADA
E.T.S. DE INGENIERÍAS INFORMÁTICA Y DE TELECOMUNICACIÓN

Asignatura: Algorítmica

Grupo nº: 4

Integrantes: Raúl Rodríguez Pérez, Francisco Javier Gallardo Molina, Inés Nieto Sánchez, Antonio Lorenzo Gavilán Chacón

Grupo: C1

Resolución Práctica 1: Análisis de Eficiencia de Algoritmos

Curso 2019-2020

Grado en Ingeniería Informática

ÍNDICE:

1. Algoritmo para el cálculo una matriz transpuesta

El problema propuesto en común para todos los grupos de la asignatura ha sido el diseño de un algoritmo que devuelva la transpuesta de una matriz. Dicha matriz debe seguir la restricción de que su tamaño debe ser de la forma n=2^k, donde n es el número de elementos que posee la matriz. Calcular la transpuesta de una matriz no es otra cosa que modificar las filas de dicha matriz en columnas o viceversa, es decir:

Ejemplo: Matriz A -> 1 2 3 transpuesta de M -> 1 4 7
4 5 6 2 5 8
7 8 9 3 6 9

El algoritmo que hemos implementado siguiendo una técnica de fuerza bruta, básicamente realiza la tarea de recorrer el triángulo superior de la matriz (los valores que se encuentran por encima de la diagonal principal), intercambiando cada valor en (i,j) por (j,i).

Por otro lado, el algoritmo implementado por un método de divide y vencerás, se basa en un algoritmo recursivo en el que se irá dividiendo la matriz en pequeñas matrices hasta llegar al caso base de una matriz de orden 2, en donde simplemente se intercambiará los valores por encima y debajo de la diagonal principal. Tras resolver todas las submatrices de orden 2, se irán reagrupando formando la matriz original, con el pequeño matiz de que, se deberán intercambiar también las matrices que estén por encima de la diagonal de matrices (formada por las matrices que poseen los elementos de la diagonal principal) con las opuestas que están debajo de dicha diagonal.

2. Análisis teórico del algoritmo

En el caso del algoritmo de cálculo de la traspuesta por el método de fuerza bruta podemos deducir su eficiencia que es de O(n2). Esto lo hemos podido deducir ya que, sí apreciamos el código adjunto donde se encuentra el algoritmo, observamos que para realizar la transpuesta por dicho método, empleamos un doble bucle anidado junto al intercambio de valores. Analizando estos factores, desde un punto de vista de eficiencia, llegamos a la conclusión de que:

$$\sum_{i=0}^{n-1}\sum_{j=i+1}^{n-1} a = \sum_{i=0}^{n-1} a[(n-1)-(i+1)+1] = \sum_{i=0}^{n-1} a(n-i-1)$$

$$= a\left[\sum_{i=0}^{n-1} n - \sum_{i=0}^{n-1} i - \sum_{i=0}^{n-1} 1\right] = a\left(n^2 - n\frac{n+1}{2} - n\right) = a\left(\frac{n^2-3n}{2}\right)$$

$$Eficiencia \in \boldsymbol{O(n^2)}$$

Por otro lado, vamos a analizar el algoritmo que hemos implementado bajo el método divide y vencerás. Para analizar la eficiencia en este caso, al utilizar dicho método, debemos buscar cual es la ecuación recurrente que rige nuestro algoritmo. Dicha ecuación se trata de: T(n)=4T(n/2)+(n/2)^2. Hemos llegado a esta ecuación debido a que en el código de nuestro algoritmo se recurre cuatro veces a la función de recurrencia y en cada caso se calcula la traspuesta de cada una de las cuatro submatrices cuadradas en que se divide la matriz original. Como estas submatrices tienen de dimensión n/2, se obtiene el término 4T(n/2). Por otro lado el algoritmo también debe intercambiar las dos submatrices de la diagonal secundaria y esto lo hace con una función llamada swap que utiliza dos bucles for anidados y cada uno con un recorrido de n/2, por tanto se debe sumar al término anterior el factor n/2 elevado al cuadrado (ver código adjunto).

A continuación procederemos a calcular la recurrencia para calcular la eficiencia de nuestro algoritmo:

$$T(n) = 4T\left(\frac{n}{2}\right) + \left(\frac{n}{2}\right)^2$$

$$Cambio:\ n = 2^k \ \rightarrow T(2^k) = 4T\left(\frac{2^k}{2}\right) + \left(\frac{2^k}{2}\right)^2 = 4T(2^{k-1}) + \frac{1}{4}4^k$$

$$T_{(k)} = 4T_{(k-1)} + \frac{1}{4}4^k \ \rightarrow \ T_{(k)} - 4T_{(k-1)} = \frac{1}{4}4^k$$

$$(x-4)^2 = 0$$

$$T(n) = C_1 4^k + C_2 k 4^k = C_1 n^2 + C_2 n^2 \log_2 n$$

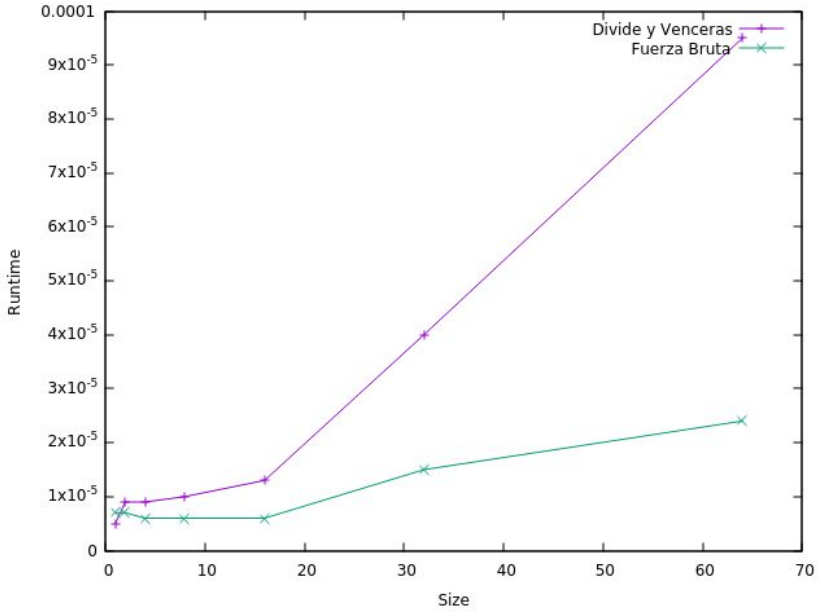
$$T(n) \in \boldsymbol{O(n^2 \log_2 n)}$$

Así que como conclusión tenemos que, teóricamente, nuestro algoritmo para el cálculo de la transpuesta de una matriz basado en la técnica de divide y vencerás tiene una eficiencia del orden **O(n^2logn)**.

2.1 Análisis del umbral entre los métodos

Pasamos ahora a analizar el umbral por el que, podemos establecer desde qué punto es más eficiente uno de los métodos por los que implementamos el algoritmo para el cálculo de la transpuesta. Al haber realizado el análisis teórico de ambos métodos, tenemos por un lado que, por el método de fuerza bruta, el algoritmo presenta una eficiencia de orden **O(n^2)**, y por otro lado, con el método divide y vencerás, observamos una eficiencia de orden **O(n^2logn)**. Se puede deducir a partir de la eficiencia para los dos casos que para **n = 2** su eficiencia es la misma, es decir para una matriz cuadrada. Para el caso de **n < 2**, es decir para una matriz con un solo elemento es más eficiente el método de divide y vencerás y para **n > 2** es más eficiente el método de la fuerza bruta(desarrollamos más acerca de esto en la conclusión al final del trabajo). Esto lo hemos sacado al igualar las eficiencias, quedándonos el 'umbral' o punto donde se cruzan las ecuaciones:

$$n^2 = n^2 \log_2 n \ \rightarrow \ 1 = \log_2 n \ \rightarrow \boldsymbol{n = 2}$$



3. Caso de ejecución de la transpuesta de una matriz

Diseñamos un algoritmo "divide y vencerás" que nos permita calcular la traspuesta de una matriz cuya dimensión sea una potencia de 2. Para diseñar la solución de este ejercicio, nos va a ser muy útil usar las siguientes notaciones:

M[i..j, k..l] va a representar la submatriz de M[1..n, 1..n] obtenida al considerar solo las filas que van desde la i hasta la j, y las columnas que van desde la k hasta la l. Por tanto, se ha de verificar que: 1 <= i <= j <= n y 1 <= k <= l <= n.

Caso base: j - 1 = i. Como hemos visto en el ejemplo, podemos considerar como caso base aquellas matrices cuyo tamaño sea 2 x 2, es decir, las matrices de la forma M[i..j,k..l] con j - 1 = i. En este caso:

$$M = M[i..j,\ k..l] = \begin{bmatrix} M[i,k] & M[i,l] \\ M[j,k] & M[j,l] \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

Su matriz traspuesta sería:

$$M^t = M[i..j,\ k..l] = \begin{bmatrix} M[i,k] & M[j,k] \\ M[i,l] & M[j,l] \end{bmatrix} = \begin{bmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \end{bmatrix}$$

Por tanto, en nuestro caso base sólo tenemos que intercambiar las componentes M[i,l] y M[j,k] Para hacer esto utilizamos la siguiente función:

```
void swap (int ** & M,int filIniA, int colIniA, int filIniB, int colIniB, int dimen) {
    for (int i = 0; i < dimen; i++) {

        for (int j = 0; j < dimen; j++) {
            int aux = M[filIniA + i][colIniA + j];
            M[filIniA + i][colIniA + j] = M[filIniB + i][colIniB + j];
            M[filIniB + i][colIniB + j] = aux;
        }
    }
}
```

Esta función intercambia en una matriz cuadrada (dividida en 4 submatrices cuadradas) la submatriz cuadrada inferior izquierda, por la submatriz cuadrada superior derecha, cuando la ejecuto con el siguiente código:

```
int filMedio = (filInicio + filFin) / 2;
int colMedio = (colInicio + colFin) / 2;

// Caso básico de matriz 2x2
if (filFin - 1 == filInicio)
  swap (M, filMedio + 1, colInicio, filInicio, colMedio + 1, filFin - filMedio);
```

En este caso las submatrices están formadas por un solo elemento.

Caso recursivo: j - 1 > i. En este caso podemos dividir siempre la matriz M de partida en cuatro submatrices Mij, en segundo lugar intercambiar la submatriz cuadrada inferior izquierda por la submatriz cuadrada superior derecha (esto se hace con la función de la imagen anterior) y calcular la traspuesta Mt a partir de las traspuestas Mijt de cada submatriz:

$$M = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} \quad \text{-----> } \quad M^t = \begin{bmatrix} M_{11}{}^t & M_{21}{}^t \\ M_{12}{}^t & M_{22}{}^t \end{bmatrix}$$

El cálculo de la traspuesta de cada submatriz se realiza aplicando la recursividad de la siguiente manera, en la que se calcula en orden la traspuesta de la submatriz M11, M21, M12 y M22:

```
void DyVtrasponer (int ** & M, int filInicio, int filFin, int colInicio, int colFin)
```

```
DyVtrasponer (M, filInicio, filMedio, colInicio, colMedio);
DyVtrasponer (M, filInicio, filMedio, colMedio + 1, colFin);
DyVtrasponer (M, filMedio + 1, filFin, colInicio, colMedio);
DyVtrasponer (M, filMedio + 1, filFin, colMedio + 1, colFin);
```

Un ejemplo de ejecución es el cálculo de una matriz de 4x4, en la que se puede ver los pasos en las siguientes imágenes:

```
Matriz original:

23      44      44      39
31      4       18      21
12      44      48      41
30      40      39      25

Matriz [M11 M12
        M21 M22]
```

```
Paso 1: Interambio de M12 con M21

23      44      12      44
31      4       30      40
44      39      48      41
18      21      39      25
```

```
Paso 2: Traspuesta de M11

23      31      12      44
44      4       30      40
44      39      48      41
18      21      39      25
```

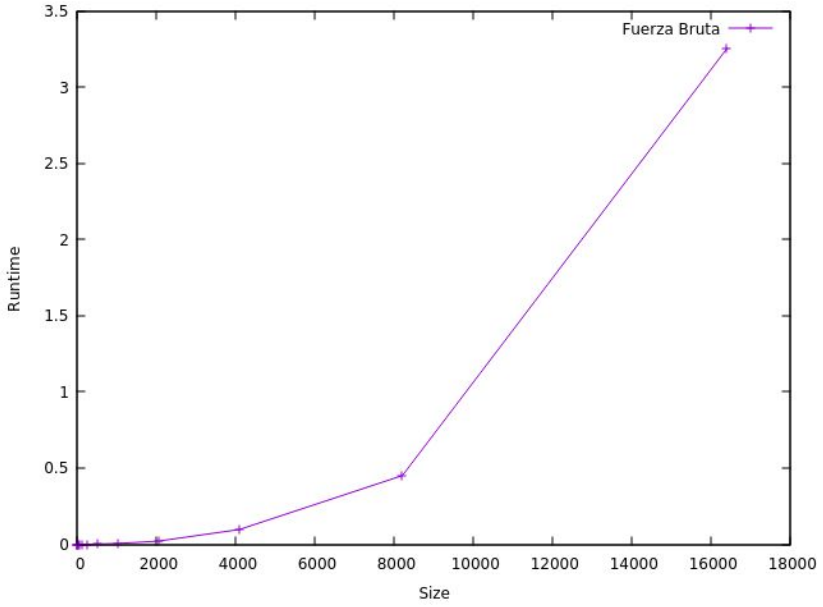
```
Paso 3: Traspuesta de M21

23      31      12      30
44      4       44      40
44      39      48      41
18      21      39      25
```

```
Paso 4: Traspuesta de M12

23      31      12      30
44      4       44      40
44      18      48      41
39      21      39      25
```

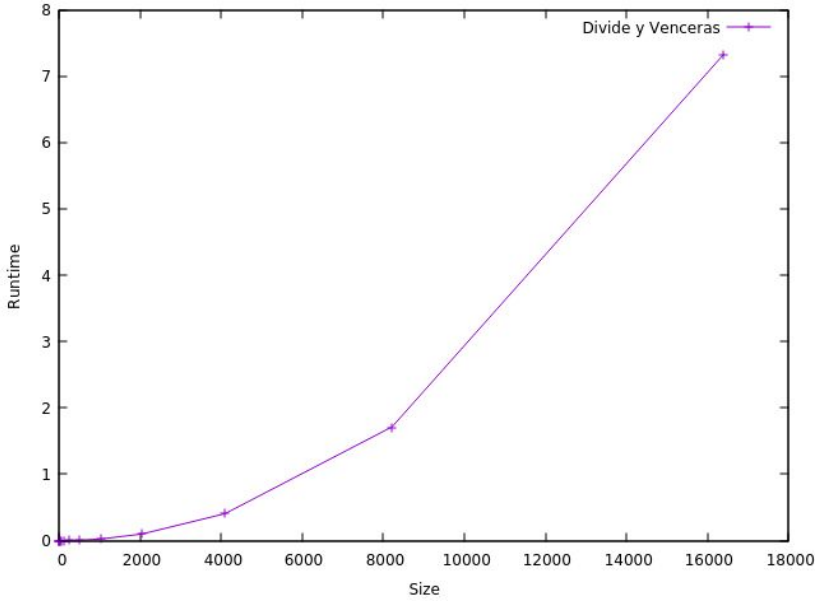
```
Paso 5: Traspuesta de M22

23      31      12      30
44      4       44      40
44      18      48      39
39      21      41      25
```

4. Análisis empírico

Para el estudio de la eficiencia empírica no hemos tenido ningún problema, realizamos un estudio normal de la eficiencia, tal cual lo habíamos hecho en la práctica anterior y los resultados fueron bastante buenos, siendo estos muy similares a lo esperado. Destacar que hemos ejecutado el algoritmo de la forma que nos han indicado,usando matrices cuadradas de la forma 2^k*2^k, por lo que, ampliando 1 a 1 la k, hemos ejecutado el algoritmo sucesivamente hasta 16000. La gráficas resultantes empleando los métodos de fuerza bruta y divide y vencerás han sido las siguientes:



Podemos apreciar, siendo el eje y el tiempo de ejecución; y el eje x el tamaño/orden de la matriz, que la tendencia que tiene la gráfica tiene la forma convexa característica de una función cuadrática, por lo que en primera instancia, nuestro análisis teórico es correcto.



Apreciamos en la gráfica realizada a partir del método divide y vencerás que la función en primera instancia tiene una tendencia muy similar a una cuadrática, siendo dicha gráfica muy parecida a la gráfica que hemos realizado anteriormente por el método de fuerza bruta. Es cierto que en nuestro análisis teórico, ambas algoritmos tienen eficiencias parecidas, pero por ahora no podemos afirmar nada con certeza.

5. Análisis híbrido

Para calcular las eficiencia híbridas de ambos algoritmos hemos realizado, primeramente, un ajuste para obtener las constantes ocultas de la función f(x) = a*x*x+b*x+c, en el caso del algoritmo empleado en el método de fuerza bruta, y f(x)=a*x^2*log(x)+b, en el caso del algoritmo empleado con la técnica divide y vencerás. Después de realizar ambos ajustes los valores de las constantes son:

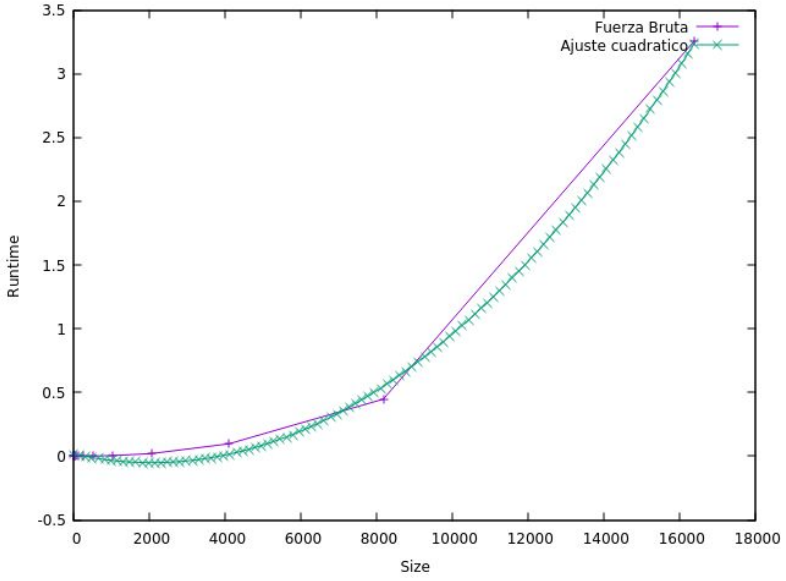
```
Final set of parameters
=======================
a                = 1.60252e-08
b                = -6.61124e-05
c                = 0.0171797
```

Fuerza bruta

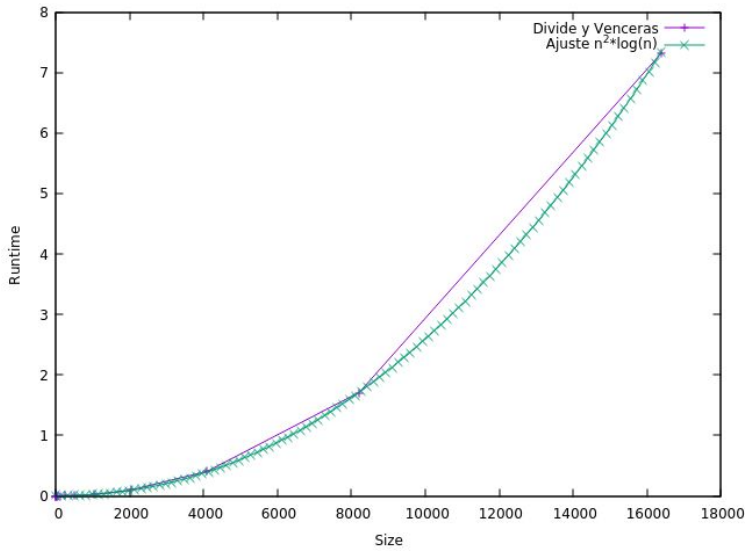
```
Final set of parameters
=======================
a                = 2.81391e-09
b                = 0.00185356
```

Divide y vencerás

Ahora al haber calculado las constantes, vamos a proceder a representar y comparar dicha función con los valores empíricos anteriormente calculados:



Apreciamos que, en la gráfica del algoritmo por fuerza bruta, el ajuste es muy bueno, pudiendo afirmar que nuestro análisis teórico era el correcto y nuestro algoritmo se rige por una eficiencia de orden cuadrático.



Observamos que, en la gráfica del algoritmo por divide y vencerás, se disipan las dudas que teníamos a primera vista con la eficiencia empírica, puesto que, al hacer el ajuste cuadrático para saber si este se parecía tanto como se apreciaba en primera instancia, nos dimos cuenta de que el ajuste era muy malo. Pero, al emplear el ajuste con las conclusiones que habíamos sacado en el análisis teórico, nos encontramos con un ajuste bastante bueno, pudiendo afirmar que la eficiencia del algoritmo es de orden O(n^2*log(n)).

6. Algoritmo para una serie unimodal de números

El problema propuesto para nuestro grupo ha sido el diseño de un algoritmo que nos permita determinar el pivote en una serie unimodal de números. Más concretamente se nos ha pedido determinar ‘p’ (pivote) en un vector ‘v’ de tamaño ‘n’, en el que a la izquierda de ‘p’ los números se encuentran ordenados de forma creciente y a la derecha de ‘p’, de forma decreciente:

Ejemplo: vector[] = (1 2 3 4 **10** 9 8 7 6 5)
pivote = 10

Nuestro profesor nos ha dado ya implementado la forma de rellenar aleatoriamente el vector. Cabe destacar que aunque los números se añaden ‘aleatoriamente’, la forma de rellenar el vector sigue un patrón. Dicho patrón se basa en que, siguiendo el ejemplo puesto anteriormente, la primera secuencia de números se encuentra ordenados de forma creciente hasta un número que denotaremos por ‘x’ (en el caso del ejemplo x=4). Tras finalizar la secuencia se encuentra el pivote y a continuación, estará la secuencia de números ordenados de forma descendiente que comenzarán con el número ‘p-1’(siendo p el pivote) y finalizará por ‘x+1’ (siendo x el último elemento de la secuencia creciente).

El algoritmo implementado con una técnica de fuerza bruta, básicamente realiza la tarea de recorrer todo el vector comparando todas sus componentes para comprobar finalmente, cual de todas posee el mayor valor, que en defecto, es el pivote que buscamos.

Por otro lado, el algoritmo implementado por el método de divide y vencerás, se basa en un algoritmo recursivo en el que destacamos 3 posibles casos bases; en primer lugar, si el vector solo posee un elemento, dicho elemento será nuestro pivote. En segundo lugar, si el vector posee solo 2 elementos, el pivote será el mayor de dichos elementos. Y por último, si al dividir el vector en 2 y tomar el valor medio, comprobamos que dicho valor es el mayor, entonces será nuestro pivote. Por otro lado, si no se han cumplido ninguno de los casos propuestos se pasa a analizar las dos posibles vertientes; una vez separado en dos mitades el vector, en qué lado se encuentra el pivote, izquierda o derecha. Tras conocer en cuál de los lados está, se vuelve a llamar recursivamente a la función pasándole los parámetros inicio y fin correspondientes según el caso.

7. Análisis teórico del algoritmo

En primer lugar vamos a analizar el algoritmo unimodal que hemos desarrollado por el método de fuerza bruta. En dicho algoritmo, se puede deducir a partir del código adjunto que su eficiencia es lineal **O(n)**. Esto es así ya que para obtener el elemento pivote, se recorre el vector desde el principio hasta que se encuentra este elemento, comparando elementos contiguos de dos en dos. En el peor caso, en donde el pivote se encuentra al final del vector, se recorrería esté completamente y como tiene “n” elementos se deduce que la eficiencia es lineal, como queríamos demostrar.

Por otro lado, vamos a analizar el algoritmo unimodal que hemos implementado usando, esta vez, el método divide y vencerás. Para analizar la eficiencia en este caso, al utilizar el método comentado anteriormente, en el cual se sigue la política de dividir el problema principal en subproblemas, resolver dichos subproblemas y luego juntar todas las soluciones para conseguir el resultado del problema principal. El primer paso que debemos realizar es conocer la ecuación recurrente que rige nuestro algoritmo. Dicha ecuación es: T(n)=T(n/2) + a, hemos llegado a esta ecuación debido a que en el código hemos dividido a la mitad el vector, comprobando si el pivote se encuentra a la izquierda o a la derecha de la mitad del vector (de ahi el T(n/2)), y además la ‘a’ sería la constante que se daría en el caso de que el pivote se encuentre justo en la mitad del vector.

A continuación procederemos a calcular la recurrencia para calcular la eficiencia de nuestro algoritmo:

$$T(n) = T\left(\frac{n}{2}\right) + a$$

$$Cambio:\ n = 2^k \ \rightarrow\ T(2^k) = T\left(\frac{2^k}{2}\right) + a = T(2^{k-1}) + a$$

$$T_{(k)} = T_{(k-1)} + a \ \rightarrow\ T_{(k)} - T_{(k-1)} = a$$

$$(x-1)^2 = 0$$

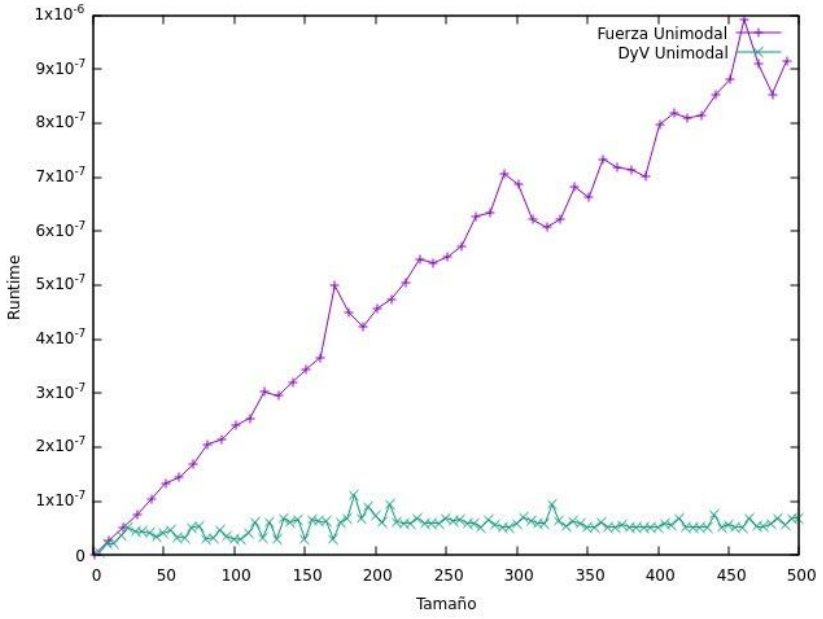
$$T(n) = C_1 1^k + C_2 k 1^k = C_1 + C_2 \log_2 n$$

$$T(n) \in \boldsymbol{O(\log_2 n)}$$

Así que como conclusión tenemos que, teóricamente, nuestro algoritmo unimodal basado en la técnica de divide y vencerás tiene una eficiencia del orden O(logn).

7.1 Análisis del umbral entre los métodos

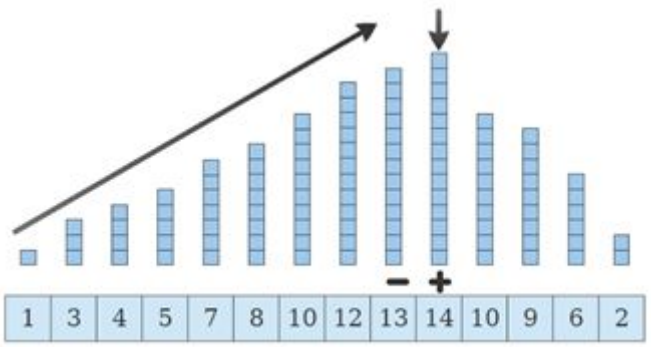
Pasamos ahora a analizar el umbral por el que, podemos establecer desde qué punto es más eficiente uno de los métodos por los que implementamos el algoritmo para el cálculo del pivote en una serie unimodal de números. Al haber realizado el análisis teórico de ambos métodos, tenemos por un lado que, por el método de fuerza bruta, el algoritmo presenta una eficiencia de orden O(n), y por otro lado, con el método divide y vencerás, observamos una eficiencia de orden O(logn). Se puede deducir a partir de la eficiencia para los dos casos que, no existe ningún valor 'umbral' en el cual las ecuaciones dadas por los dos algoritmos lleguen a cruzarse. Por lo que, representando gráficamente para apoyar estas palabras, podemos observar que el algoritmo implementado por el método dyv siempre será más eficiente.



8. Caso de ejecución de la serie unimodal de números

Diseñamos el algoritmo de fuerza bruta para la serie unimodal de números con el siguiente método:

- Avanzamos desde la primera posición del vector de una en una hasta que encontremos que el valor de la posición actual es menor que el de la anterior.
- Devolvemos el valor del vector en la posición anterior, que es el pivote.



El código es fácil de entender con la explicación anterior (ver imagen de abajo).

```
int unimodal (const int * v, int inicio, int fin) {

    int max = inicio;

    for (int i = inicio + 1; i <= fin; i++) {

        if (v[i] > v[max])       //Todavia no hemos llegado al pico
            max = i;             //Guardamos la posición

        else                  //Ya he encontrado el pico
            return v[max] ;  //Devuelvo la posición anterior
    }
}
```

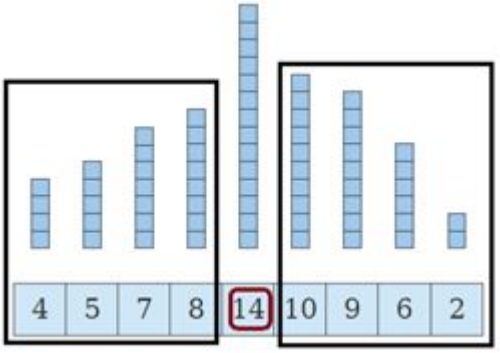
Diseñamos el algoritmo de divide y vencerás para la serie unimodal de números con el siguiente método. Nos posicionamos en la mitad del vector y comparamos los valores de las posiciones a ambos lados de la mitad:

1) Si el vector dado contiene un solo elemento, lo devolvemos y si tiene dos, devolvemos el mayor de ellos.

```
if (tam == 1) {  //Caso base 1. Solo tenemos 1 elemento
    return vp;
}

else if (tam == 2) {  //Caso base 2. Tenemos 2 elementos
    return v[posMax (v, inicio, fin)];
}
```

2) En el caso de que no se cumpla ninguna de las anteriores condiciones:

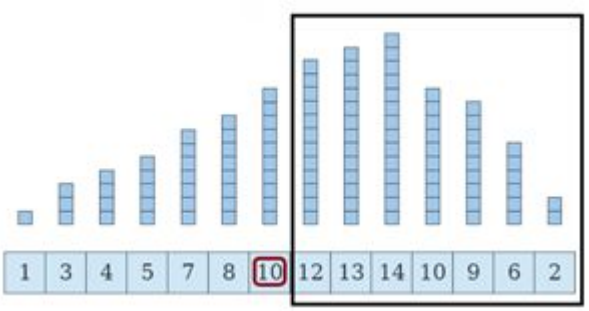
2-a) Si el punto medio es mayor que el elemento anterior y posterior del vector, es el pico y lo devuelve.



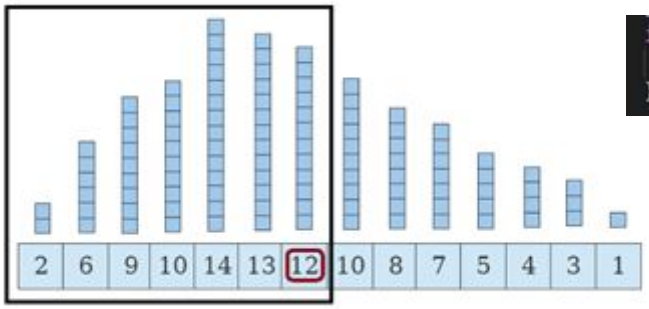

```
else {

    if(vp > a && vp > b) return vp; //Caso base 3. El pico es el medio
```

2-b) Si el valor en la posición medio es menor que el de la derecha, el pico se encuentra en esa dirección, así que llamamos al algoritmo entre la mitad + 1 y la posición final devolviendo la posición del valor máximo que haya entre esas dos partes.

```
else {

    if(vp < b) {//El pico está a la derecha del medio
        return dyvunimodal(v, medio + 1, fin);
    }
```



2-c) Si el valor en la posición medio es mayor que el de la derecha, el pico se encuentra en la parte izquierda del vector, así que llamamos al algoritmo entre el inicio y la posición medio, devolviendo la posición del valor máximo que haya entre esas dos partes.




```
if(vp > b) {//El pico está a la izquierda del medio
    return dyvunimodal(v, inicio, medio);
}
```

Se ha procedido a ejecutar el código paso por paso para comprobar el funcionamiento del algoritmo de divide y vencerás. Se puede ver en las imágenes siguientes dos ejemplos de su ejecución:

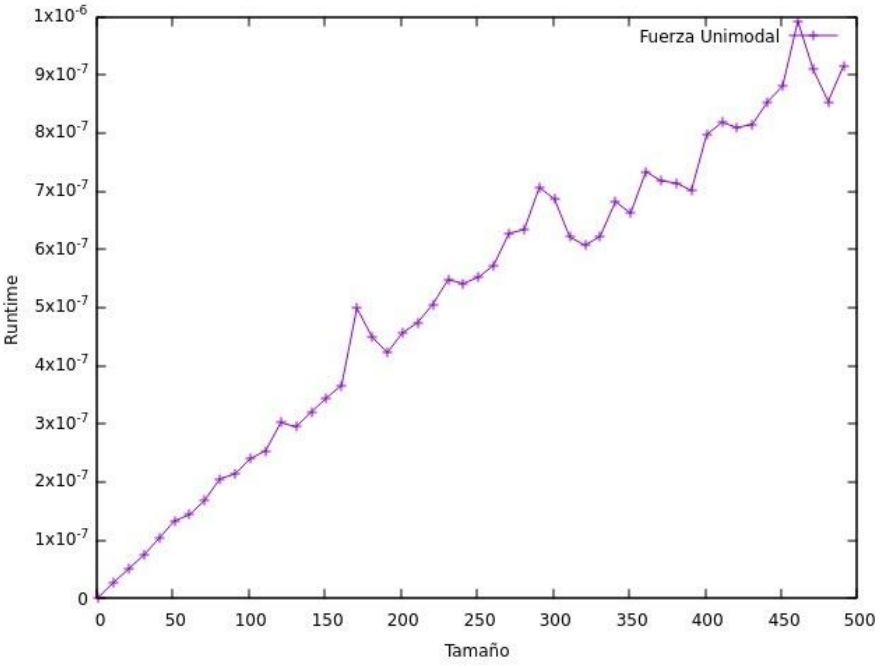
```
antonio@ubuntu-VirtualBox:~/Documentos/2C/ALG/PRACTICAS/
LTA/C1--Equipo4--P2/ejer3.3$ ./DyVunimodal_pasos 10
0 9 8 7 6 5 4 3 2 1   ----> Pico = 5

0 9 8 7 6 5   ----> Pico = 7

0 9 8 7   ----> Pico = 8

0 9 8   ----> Pico = 9

El pico es 9
```

```
antonio@ubuntu-VirtualBox:~/Documentos/2C/ALG/PRACTICAS/
LTA/C1--Equipo4--P2/ejer3.3$ ./DyVunimodal_pasos 10
0 1 2 3 4 5 9 8 7 6   ----> Pico = 5

9 8 7 6   ----> Pico = 7

9 8 7   ----> Pico = 8

9 8   ----> Pico = 9

El pico es 9
```

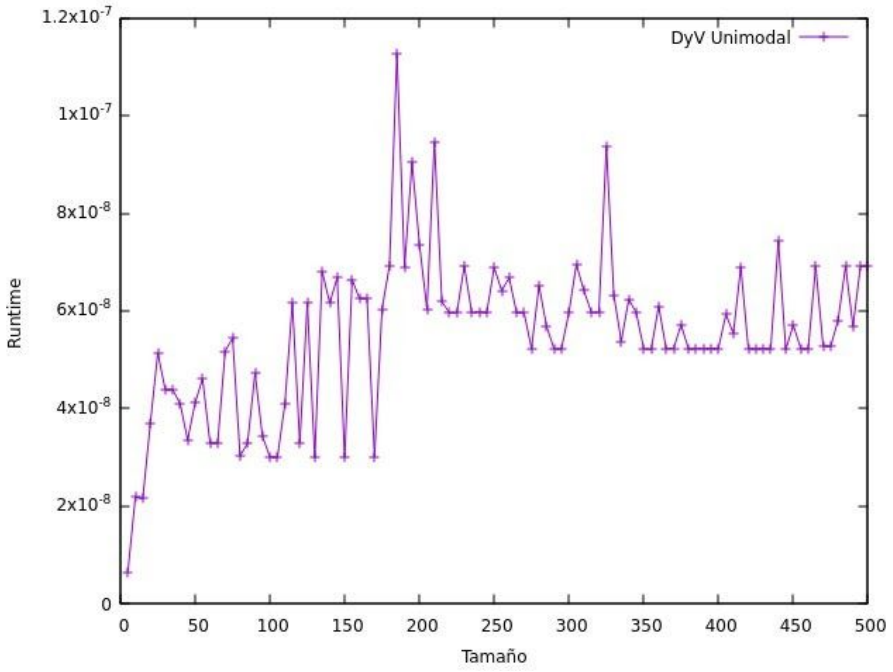
9. Análisis de la eficiencia empírica

Para el estudio de la eficiencia empírica hemos tenido algunos problemas debido a la naturaleza del problema y a los tiempos tan ínfimos que presenta.

Comenzamos realizando un estudio normal como el que empleamos en la práctica anterior, pero los resultados no eran óptimos, las gráficas hacían unos recorridos que no se asemejaba en ningún momento a lo esperado. Tras analizar más a fondo nuestro algoritmo, y los resultados de dichas gráficas nos dimos cuenta de que al crearse vectores con números aleatorios, podría darse la casualidad de que, aun siendo vectores con más elementos, se encontrara el pivote de manera más veloz que en casos con menos elementos. Por lo que, cambiamos nuestra manera de hacer el estudio, y empleamos la técnica de ejecutar cada punto un número determinado de veces, y calcular el tiempo medio de ejecución del algoritmo en dicho punto. Al poner en práctica esta técnica obtuvimos mejores resultados que los anteriores. Destacar que hemos ejecutado de 10 en 10, desde 1 elemento en el vector hasta 500 elementos, ejecutando cada uno de los elementos un total de 10000 veces. La gráficas resultantes empleando los métodos de fuerza bruta y divide y vencerás han sido las siguientes:



Apreciamos en la gráfica del algoritmo que, siendo el eje Y el tiempo de ejecución (en segundos) y el eje X el número de elementos del vector, la gráfica presenta una forma muy parecida a una recta, coincidiendo con la forma lineal que calculamos en el análisis teórico y que debe tener nuestra función.



Por otro lado la gráfica con el método de divide y vencerás, aparentemente no podemos apreciar de manera certera que posee una tendencia logarítmica (siendo esta la eficiencia calculada en el análisis teórico), pero sí se asemeja bastante a dicha tendencia.

10. Análisis de la eficiencia híbrida

Para calcular las eficiencia híbridas de ambos algoritmos hemos realizado, primeramente, un ajuste para obtener las constantes ocultas de la función f(x) = a*x+b, en el caso del algoritmo empleado en el método de fuerza bruta, y f(x)=a*log(x)+b, en el caso del algoritmo empleado con la técnica divide y vencerás. Después de realizar ambos ajustes los valores de las constantes son:

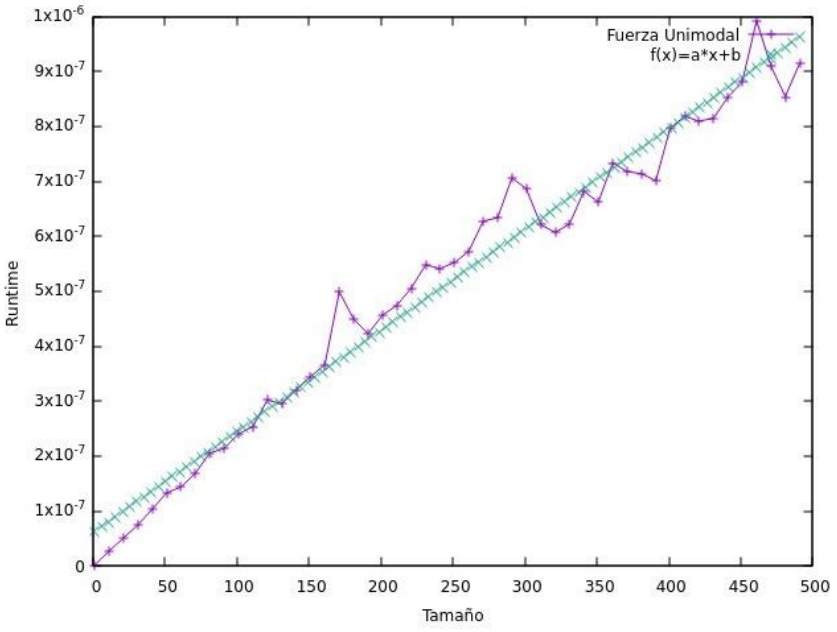
```
Final set of parameters
======================
a               = 1.8344e-09
b               = 6.13678e-08
```

Fuerza bruta

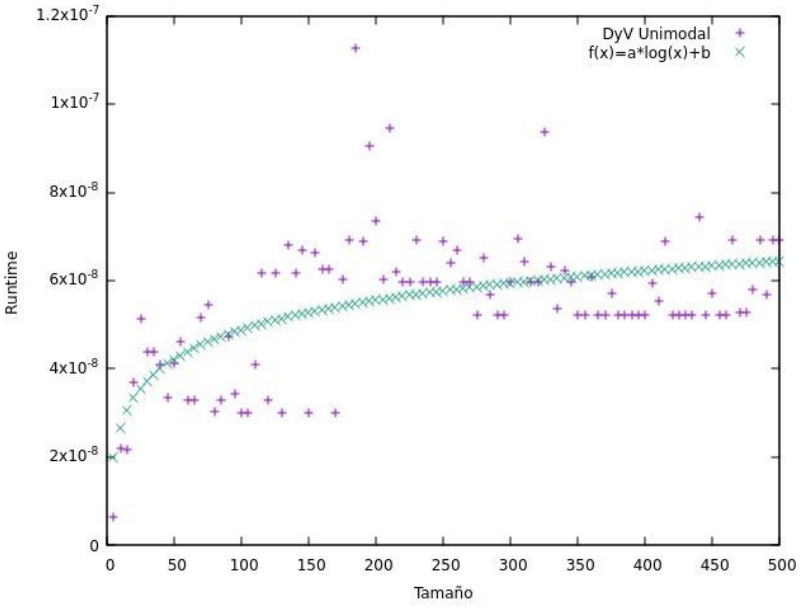
```
Final set of parameters
======================
a               = 9.6298e-09
b               = 4.56509e-09
```

Divide y vencerás

Ahora al haber calculado las constantes, vamos a proceder a representar y comparar dicha función con los valores empíricos anteriormente calculados:



Podemos apreciar que, aunque el ajuste en algunas zonas no coincida exactamente, podemos afirmar que el ajuste es bueno, quedando reflejado que nuestro estudio del análisis teórico es correcto, y dicho algoritmo presenta un orden lineal.



Por otro lado, en el caso del algoritmo por divide y vencerás, hemos optado por representar simplemente los puntos, sin líneas que los unan, ya que pensamos que se observa mejor la dispersión de los puntos y se aprecia mejor la tendencia de los resultados empíricos. Como podemos apreciar, existen puntos que no coinciden con exactitud con el ajuste, pero podemos afirmar que nuestra función posee una tendencia muy parecida, y por tanto, podemos afirmar que dicho algoritmo posee una eficiencia del orden logarítmico tal y como habíamos calculado anteriormente.

11. Conclusión

Como conclusión queremos destacar los puntos más importantes que hemos ido desarrollando y revelando a la hora de resolver esta práctica. En primer lugar nos gustaría destacar, en el caso del ejercicio de la transpuesta, que el método divide y vencerás nos da un peor resultado desde el punto de vista de la eficiencia que por el método de fuerza bruta. Esto aunque en primer momento nos pareció chocante y pensamos que nos habíamos equivocado en alguna cosa, tal y como hemos visto y estudiado en teoría, el método divide y vencerás no es más que eso, un método por el cual podemos conseguir mejores resultados para un algoritmo en concreto, pero esto no significa que para todos los algoritmos en donde empleemos dicho método vamos a tener un resultado mejor que el inicial. El caso es que nosotros conozcamos diversos métodos por los cuales podemos atacar un algoritmo, y gracias a la práctica poder conocer a qué tipo de algoritmos atacar con según qué tipos de métodos. Por otro lado destacamos, en el ejercicio que se asignó a nuestro grupo, que es muy importante conocer diversos métodos para poder analizar la eficiencia de los diferentes algoritmos. En este ejercicio nos hemos preocupado puesto que al principio los resultados que hemos obtenido han sido bastante alejados de lo que esperábamos, pero gracias a que en la anterior prácticas nos han enseñado a emplear otras tácticas hemos conseguido sacarlo adelante. Teniendo como conclusión que cada algoritmo tiene su propia naturaleza, y se debe abordar su análisis de manera específica.