PRÁCTICA 2

ALGORITMOS DIVIDE Y VENCERÁS

Antonio Jiménez Rodríguez 2D (D2)

Problema asignado

Ejercicio 7 >>

Dado un vector v[1..n] de valores numéricos enteros, se dice que los elementos v[i] y v[j] realizan una inversión si i<j pero v[i]>v[j]. Proporcionar un algoritmo que devuelva todas las posibles inversiones del vector de entrada v.

Para la resolución de nuestro problema debemos dar el número de inversiones que hay en el vector si un elemento posterior a otro tiene un valor menor que el anterior. Es decir debemos dar el número de inversiones que tendríamos que realizar para que un vector de entrada saliera ordenado ascendentemente..

1. Algoritmo básico para la resolución del problema

Vamos a mostrar el pseudocódigo de un algoritmo básico para la resolución del problema.

```
FUNCION NUM_INVERSIONES(vector, n)
INICIO
    i: ENTERO
    j: ENTERO
    inversiones: ENTERO

    inversiones<-0

    PARA i<-0 HASTA i<n-1 HACER
    INICIO
        PARA j<-(i+1) HASTA j<n HACER
        INICIO
            SI vector[i] > vector[j]
                INTERCAMBIAR (vector[i], vector[j])
                inversiones <- inversiones+1
            FIN SI
        FIN PARA
    FIN PARA
    RETURN inversiones
FIN FUNCION
```

Como podemos ver es un algoritmo de fuerza bruta muy sencillo que lo que hace es por cada posición del vector ir buscando el elemento más pequeño desde la posición en la que está colocado hacia delante, poniendo este en esa posición. Cada vez que realiza un intercambio incrementa la variable inversiones que es la que se devolverá como solución al problema.

2. Comprobar si el problema es resoluble por el método Divide y Vencerás

Nos hacemos las siguientes preguntas sobre nuestro problema::

1) **¿El problema se puede resolver por un método básico, sin usar Divide y vencerás?** Como hemos visto en el apartado anterior, si lo podemos resolver.
2) **¿El problema puede dividirse en problemas de igual naturaleza más pequeños?** Sí, se podría dividir en subvectores de menor tamaño.
3) **¿Los subproblemas son equivalentes de menor tamaño e independientes?** Sí, por ejemplo dos subvectores de igual tamaño donde en uno se encuentran elementos menores a los del otro.
4) **¿La solución de los subproblemas se pueden combinar?** Sí, aunque en este caso no sería necesario combinar la solución ya que trabajaremos sobre el mismo vector.
5) **¿Existe un caso indivisible?** Si, cuando el tamaño del subproblema es uno

Por lo tanto podemos plantearnos una resolución del problema por el método Divide y Vencerás.

3. Diseño del algoritmo Divide y Vencerás

A continuación voy a mostrar el pseudocódigo del método Divide y Vencerás para la resolución de este problema.

```
FUNCION NUM_INVERSIONES(vector, ini, fin)
INICIO
    pospivote: ENTERO
    num_inversiones: ENTERO
    num_invers_izq: ENTERO
    num_invers_der: ENTERO

    SI ini >= fin
        RETURN 0
    FIN SI
    EN OTRO CASO
        [num_inversiones, pospivote, vector] = PIVOTAR(vector, ini, fin)
        num_invers_izq = NUM_INVERSIONES(vector, ini, pospivote)
        num_invers_der = NUM_INVERSIONES(vector, pospivote+1, fin)
        num_inversiones = num_inversiones + num_invers_izq + num_invers_der
    FIN EN OTRO CASO

    RETURN num_inversiones
FIN FUNCION
```

El funcionamiento del algoritmo es sencillo.

Si ini >= fin, es decir, si el vector de entrada tiene una o cero componentes devuelve que ha habido 0 inversiones (una componente ya está ordenada.

Si no, se le aplica la función PIVOTAR() al vector. Esta función elige un elemento del vector el cual será el pivote, que tendrá todos los elementos mayores o iguales que él a su derecha y todos los elementos mayores que él a su izquierda. Posteriormente se divide el vector en 2 mitades partiendo por el pivote y se llama recursivamente a la función NUM_INVERSIONES().

Por último se suman todas las inversiones de cada partes y se devuelve como solución.

```
FUNCION PIVOTAR(vector, ini, fin)
    num_inversiones : ENTERO
    pospivote: ENTERO

    pospivote<-ini

    MIENTRAS QUE ini<=fin HACER
        ini<-ini+1
        MIENTRAS QUE ini<=fin Y vector[pospivote]>vector[ini] HACER
            ini<-ini+1
        FIN MIENTRAS QUE
        MIENTRAS QUE ini<=fin Y vector[pospivote]<=vector[fin] HACER
            fin<-fin-1
        FIN MIENTRAS QUE

        SI ini <= fin
            INTERCAMBIAR(vector[ini], vector[fin])
            fin<-fin-1
            inversiones<-inversiones+1
        FIN SI
    FIN MIENTRAS QUE

    INTERCAMBIAR(vector[pospivote], vector[fin])
FIN FUNCION
```

Pseudocódigo de la función pivotar explicada en el párrafo anterior

4. Estudio de la eficiencia de los algoritmos

Vamos a pasar ahora con el estudio de la eficiencia de cada algoritmo.

Comenzamos por el algoritmo básico que, como podemos ver en el pseudocódigo tiene una eficiencia en el peor caso de $O(n^2)$ y en el mejor caso también de $\Omega(n^2)$. Esto se da porque ejecuta dos bucles for, uno dentro de otro, con tamaño del caso igual a n ya que aunque el vector esté ordenado correctamente las condiciones de salida son i < n-1 para un bucle y j < n para el otro.

Para el algoritmo Divide y Vencerás la cosa cambia. Al ser un algoritmo recursivo debemos hallar y resolver su ecuación recurrente.
En el cuerpo del algoritmo tenemos varias sentencias O(1) y dos llamadas recursivas junto con una llamada a la función pivotar. La función pivotar no siempre divide el vector en dos partes iguales por lo que tendremos que estudiarla en el mejor y en el peor caso.

❑ Mejor caso

En el mejor caso tenemos que la función pivotar deja al pivote en medio del vector dividiéndolo así en dos mitades, por lo que la ecuación recurrente sería:

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

Vamos resolver la ecuación recurrente no homogénea:

1) Realizamos un cambio de variable

$$n = 2^m \rightarrow m = \log(n)$$

2) Pasamos todos los términos de T a la izquierda con el cambio de variable aplicado

$$T(2^m) - 2*T(2^{m-1}) = 2^m$$

3) Resolvemos la parte de la izquierda (sacamos el polinomio característico homogéneo)

$$T(2^m) - T(2^{m-1}) = 0$$

$$t_m - t_{m-1} = 0$$

$$x^m - x^{m-1} = 0$$

$$x^{m-1}(x - 1) = 0$$

$$p_h(x) = x - 2$$

4) Resolvemos la parte de la derecha (parte no homogénea)

$$2^m = b_1^m \bullet q_1(m)$$

$$b_1 = 2$$

$$q_1(m) = 1 \rightarrow d_1 = 0$$

5) Construimos el polinomio característico y calculamos t_n

$$p(x) = p_h(x) \bullet (x - b_1)^{d1+1} = (x - 2)^2$$

$$t_m = c_1\, 2^m\, m^0 + c_2\, 2^m\, m^1$$

6) Deshacemos el cambio de variable

m = log(n) $\rightarrow$ $t_n = c_1$ n* $log(n)^0 + c_2\ n * log(n)^1$ $\rightarrow$ La función es Ω (n*log(n))

❑ Peor caso

En el peor caso tenemos que la función pivotar deja al pivote en uno de los extremos del vector por lo que no lo dividiría, haría una llamada recursiva con el resto del vector sin el pivote y otra con un único elemento, el pivote cuyo orden de eficiencia de respuesta es O(1). La ecuación recurrente quedaría de la siguiente forma:

$$T(n) = T(n - 1) + n$$

1) Pasamos todos los términos de T a la izquierda con el cambio de variable aplicado

$$T(n) - T(n - 1) = n$$

2) Resolvemos la parte de la izquierda (sacamos el polinomio característico homogéneo)

$$T(n) - T(n - 1) = 0$$
$$t_n - t_{n-1} = 0$$
$$x^n - x^{n-1} = 0$$
$$x^{n-1} (x - 1) = 0$$
$$p_h(x) = x - 1$$

3) Resolvemos la parte de la derecha (parte no homogénea)

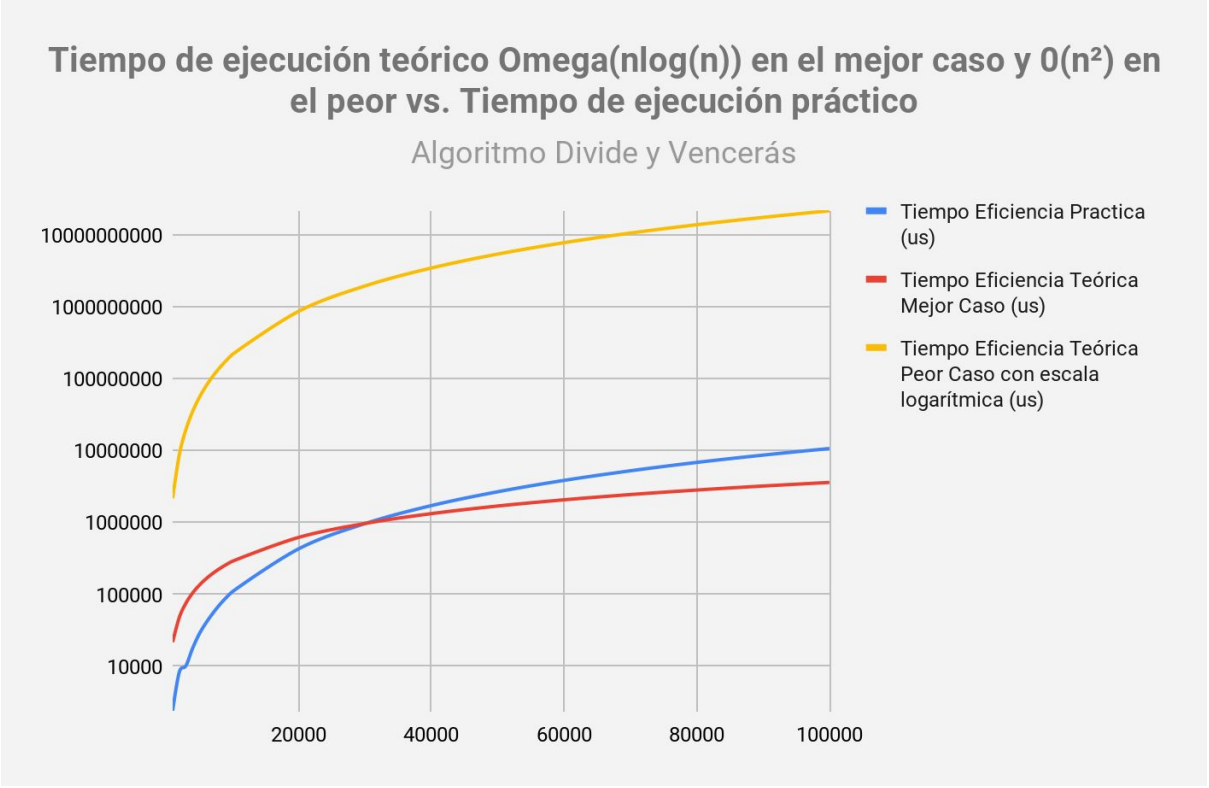
$$n = b_1^n \bullet q_1(n)$$
$$b_1 = 1$$
$$q_1(m) = n \rightarrow d_1 = 1$$

4) Construimos el polinomio característico y calculamos t_n

$$p(x) = p_h(x) \bullet (x - b_1)^{d1+1} = (x - 1)^3$$
$$t_n = c_1\ 1^n\ n^0 + c_1\ 1^n\ n^1 + c_1\ 1^n\ n^2 \rightarrow \text{La función es } O(n^2)$$

5. Análisis del problema del umbral

Vamos a analizar ahora si este algoritmo Divide y Vencerás mejora al básico propuesto. Para ello vamos a ejecutar los dos con una serie de valores y vamos a representarlos en una gráfica midiendo el tiempo de la función que resuelve el problema.



Como podemos ver el algoritmo no ronda siempre el mejor caso, pero es mucho mejor que una eficiencia n^2 que se encuentra muy por encima de los resultados prácticos. La curva de nuestra ejecución tiende a una constante cuando el tamaño del caso es muy grande. (A la curva de la eficiencia en el peor caso se le ha aplicado la escala logarítmica para poder compararla en la misma gráfica con la eficiencia en el mejor caso y la eficiencia práctica).

Sin embargo el algoritmo básico hace lo esperado, cuanto más grande es el tamaño del caso mayor es el tiempo de ejecución, tanto que cuando el número de casos tiende a infinito el tiempo también tiende a infinito.

El problema de la constante oculta en este caso no tendríamos que resolverlo. Como veremos en la siguiente gráfica el algoritmo Divide y Vencerás siempre se encuentra por debajo en tiempo de ejecución.

Vamos a demostrarlo también numéricamente. Tenemos que $K*n^2 = K'*n*\log(n)$ donde K y K' son las constantes ocultas del algoritmo básico y el algoritmo DyV respectivamente. Atendiendo a los resultados de la ejecución práctica tenemos que:

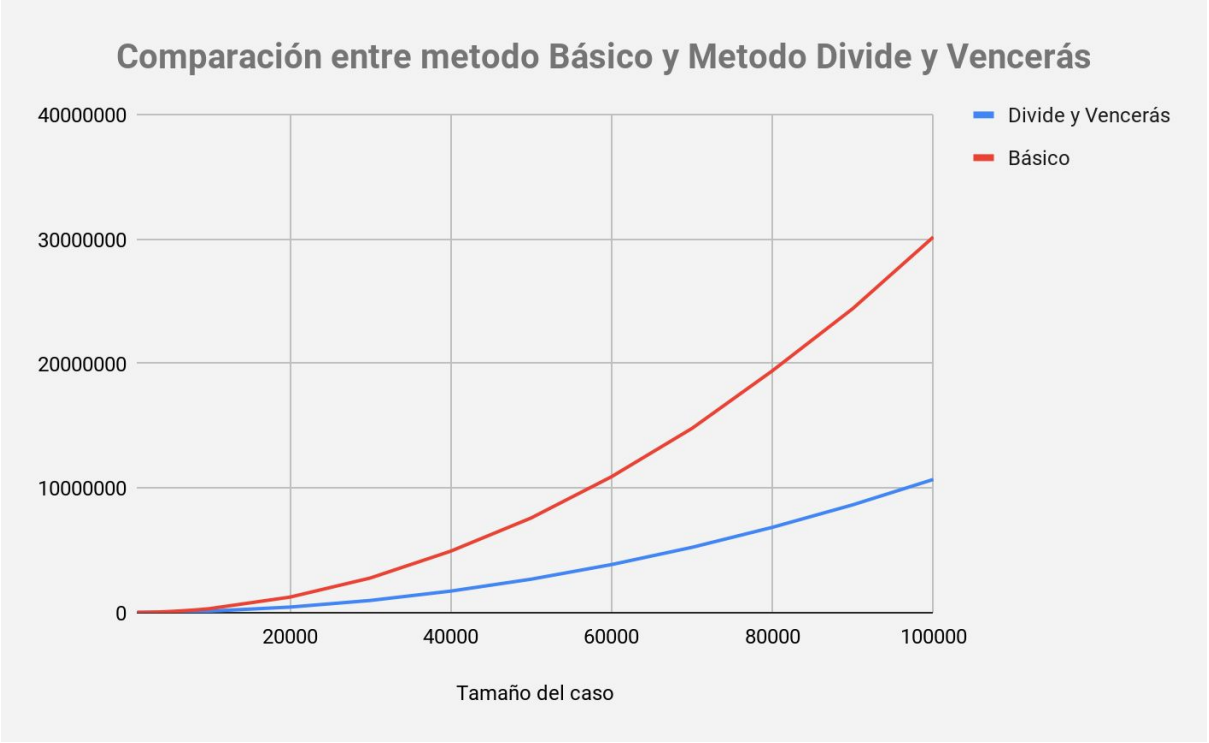
$$K = 0{,}00321474189 \text{ y } K' = 2{,}16998498$$

$$n/\log(n) = 2{,}16998498 / 0{,}00321474189$$

Esta ecuación no tiene solución para $n \in \Re^+$. Es demuestra pues que las gráficas no se cortan en ningún punto del espacio positivo.

Como conclusión es obvio que el mejor algoritmo Divide y Vencerás es mucho mejor en tiempo de ejecución, llegando a ser incluso 2,8 veces más rápido que el básico propuesto.

En la siguiente tabla donde comparamos los dos algoritmos podemos verlo visualmente.



6. Instrucciones para la compilación y ejecución del código fuente

Disponemos de dos archivos fuente, metodo-basico.cpp y metodo-dyv.cpp.

La compilación se realizará de la siguiente manera:

```
g++ metodo-basico.cpp -o metodo-basico
g++ metodo-dyv.cpp -o metodo-dyv
```

Y la ejecución:

```
./metodo-basico salida-basico.txt 1000 2000 3000 ... 100000
./metodo-dyv salida-dyv.txt 1000 2000 3000 ... 100000
```

Al inicio de cada .cpp tambíen se encuentra esta información.