Recurrencia

PRÁCTICA 2

Autor: Pablo Gómez Rivas

Materia: Lógica y Algorítmica

Grupo de Prácticas: GTA2

Fecha: Viernes, 1 de abril de 2022

Índice

Resumen

En esta práctica vamos a resolver un problema usando dos algoritmos: uno iterativo y otro recursivo, y compararemos el tiempo de ejecución de ambos para poder elegir el más adecuado. Los pasos a seguir son similares a los realizados en la práctica anterior:

1. Implementar los algoritmos en Java y pasar los juegos de pruebas JUNIT.
2. Realizar diferentes experimentos partiendo de un valor inicial para el tamaño de la entrada n, doblando el tamaño de la entrada en cada nuevo experimento, y obtener los tiempos medios de 10 ejecuciones de cada método con los mismos datos de entrada, del mismo modo que en la práctica A1.
3. Análisis de datos: elaborar tablas comparativas con los resultados de los tiempos medios, obtención de las gráficas y las tasas de crecimiento.

Se incluye un apartado final con una introducción al paradigma de Programación Dinámica que ha de usarse en la sesión 04.

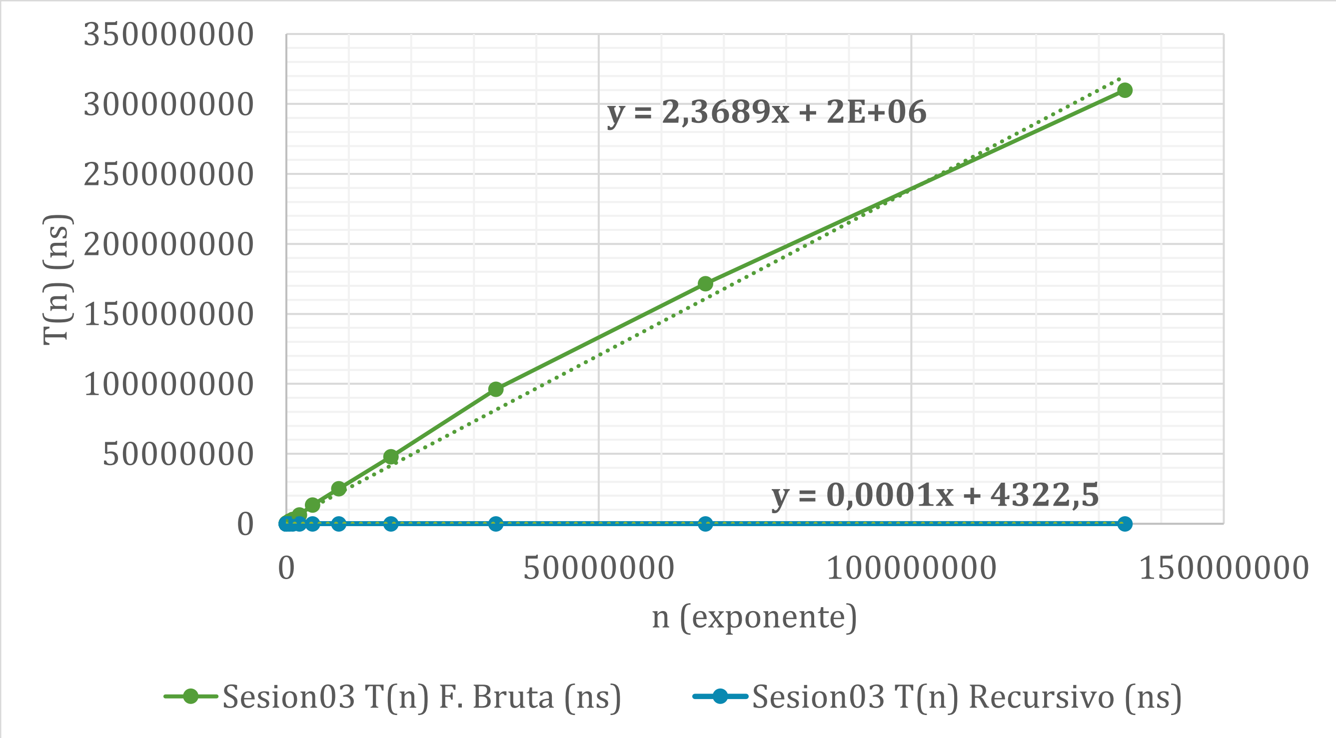
Sesión 01. Ordenar una matriz por filas

Ejercicio 1: Tabla tiempo de ejecución VS Tamaño exponente

n (exponente)	$\overline{T}$(n) F. Bruta (ns)	$\overline{T}$(n) Recursivo (ns)
64	3533,3333	1400
128	5011,1111	1233,3333
256	11522,2222	1677,7778
512	21544,4444	1733,3333
1024	31677,7778	1633,3333
2048	61200	1711,1111
4096	128777,7778	2577,7778
8192	227388,8889	2355,5556
16384	337944,4444	4011,1111
32768	493666,6667	5100
65536	670422,2222	5366,6667
131072	640788,8889	2733,3333
262144	1165822,222	4433,3333
524288	1926711,111	4655,5556
1048576	3285388,889	5366,6667
2097152	6277677,778	6500
4194304	13447411,11	8833,3333
8388608	25110366,67	9944,4444
16777216	47895055,56	9877,7778
33554432	96198444,44	15888,8889
67108864	171696644,4	15200
134217728	309.885.311	18566,6667

Tiempo de Ejecución VS Tamaño de la Entrada

Ejercicio 2: Gráfica Tiempo de ejecución VS Tamaño del exponente



Ejercicio 3: Orden de complejidad del algoritmo iterativo

Realizamos un análisis teórico del algoritmo estudiando sus estructuras de control.

```
public double exponenFuerzaBruta() {

        double resultado = 1;

        for (int i = 0; i < exponente; i++)

            resultado *= base;

        return resultado;

    }
```

- Las líneas 1, 3 y 4 son operaciones elementales de tiempo constante, por lo que su orden de complejidad es de $O(1)$.
  Bucle FOR de la línea 2:

$$T_{bucle} \leq \sum_{i=0}^{n-1} c = nc \in O(n)$$

$$O\big(max\ (1\ , n)\big) = O(n)$$

$$T_{F.bruta} \in O(n)$$

Ejercicio 4: Orden de complejidad del algoritmo recursivo

Un algoritmo recursivo se basa en la técnica de Divide y Vencerás, es decir, resuelve un problema resolviendo una o más instancias más pequeñas del mismo problema.

En este caso, el problema que se nos plantea es calcular a^n:

$$a^n = \begin{cases} a & \text{si } n = 1 \\ (a^{\frac{n}{2}})^2 & \text{si } n \text{ es par} \\ a.a^{n-1} & \text{si } n \text{ es impar} \end{cases}$$

Si calculamos a^n recursivamente y medimos la eficiencia del algoritmo por el número de multiplicaciones deberíamos esperar que el algoritmo esté en $O(log\ n)$ porque , en cada iteración , el tamaño se reduce casi a la mitad.

Conclusiones de ambos algoritmos:

Una vez analizado cada algoritmo por separado, podemos afirmar que el algoritmo recursivo tiene un orden de complejidad logarítmico, menor (crece más lento) que el orden lineal que tiene el algoritmo fuerza bruta, ya que:

$$\lim_{n\to\infty} \left(\frac{\log\ n}{n}\right) = 0$$

$$\log n \in O(n)$$

Por lo tanto, si tomamos un tamaño del exponente lo suficientemente grande, es preferible optar por el método recursivo.

Ejemplo: a^7

Método Fuerza Bruta: realiza 7 multiplicaciones.

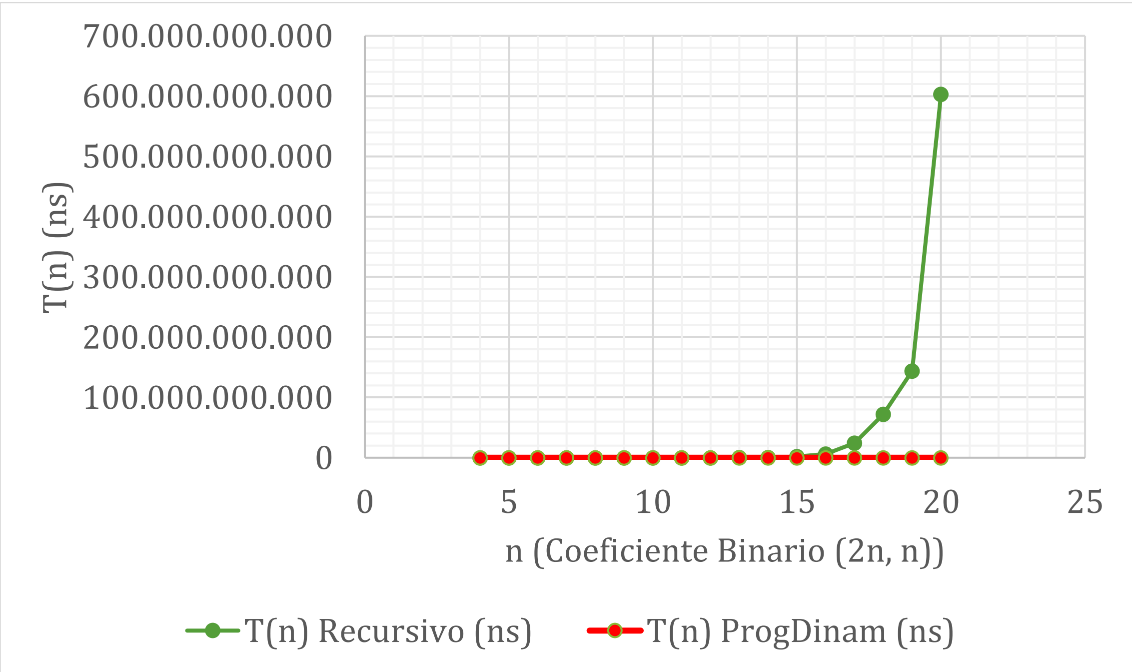
Método Recursivo: $a^7 = a \cdot a^6 = a(a^3)^2 = a(a \cdot a^2)^2 \rightarrow$ 4 multiplicaciones.

Sesión 04. Calcular coeficiente binomial

Ejercicio 1: Tabla tiempo medio para cada algoritmo

n (Coef Binario (2n, n))	$\overline{T}$(n) Recursivo (ns)	$\overline{T}$(n) ProgDinam (ns)
4	20.244	7.556
5	42277,7778	8411,1111
6	95188,8889	11900
7	113955,5556	14711,1111
8	355688,8889	24777,7778
9	569666,6667	22522,2222
10	1795988,889	26788,8889
11	6809900	35611,1111
12	25766677,78	45477,7778
13	94243422,22	64177,7778
14	279.231.133	54277,7778
15	1738291311	90200
16	6277085367	98911,1111
17	24.287.439.211	94933,3333
18	72192579056	89311,1111
19	1,43874E+11	68922,2222
20	6,03218E+11	75800

Ejercicio 2: Gráfica Tiempo ejecución VS Tamaño de entrada



Ejercicio 3: Orden de complejidad de cada método

- Algoritmo recursivo: es menos eficiente que algoritmo dinámico, ya que el recursivo realiza muchos cálculos repetidos.

  Tomando la entrada $\binom{2n}{n}$, el algoritmo tiene un orden exponencial de $O(4^n)$.
- Algoritmo dinámico: en general, sería de orden $O(n \cdot k)$, pero en nuestro caso k = 2n, por lo que sería de orden de complejidad $O(n^2)$.

```
public long coefBinomialProgDinam() {

        int n, k;

        int[][] matriz = new int[cbN + 1][cbK + 1];

        for (n = 0; n <= cbN; n++)

              matriz[n][0] = 1;

        for (k = 1; k <= cbK; k++)

              matriz[0][k] = 0;

        for (n = 1; n <= cbN; n++) {

              for (k = 1; k <= cbK; k++)

                    matriz[n][k] = matriz[n - 1][k - 1] +
matriz[n - 1][k];

        }

return matriz[cbN][cbK];

}
```

Los bloques de fondo blanco tienen orden de complejidad de $O(1)$, ya que son operaciones elementales de tiempo constante.

El bloque verde tiene orden de complejidad de $O(n)$, ya que:

$$T\ bucle \le \sum_{n=0}^{cbN=2n} c = (2n+1) \cdot c \in O(n)$$

El bloque azul tiene orden de complejidad de O(n), ya que:

$$T\ bucle\ \leq \sum_{k=1}^{cbK=n} c\ =\ n\ \cdot\ c\ \epsilon\ O(n)$$

El bloque rojo tiene orden de complejidad de $O(n^2)$, ya que:

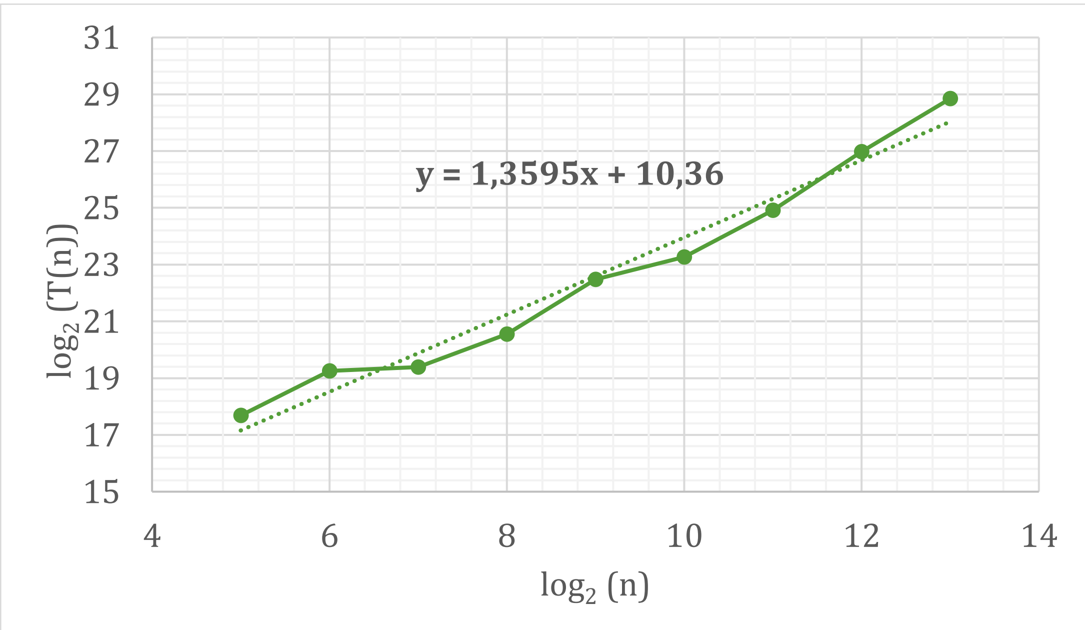
$$T\ bucle\ \leq \sum_{n=1}^{cbN=2n} \sum_{k=1}^{cbK=n} d\ =\ 2n\ \cdot\ nd\ \epsilon\ O(n^2\ )$$

El orden de complejidad del método de programación dinámica es

$O(máx\ (1, n\ , n^2))$ = $\boldsymbol{O(n^2)}$

Ejercicio 4: Comparación resultados experimentales y teóricos del algoritmo dinámico

n (Coef Binario (2n, n))	$\overline{T}$(n) ProgDinam (ns)	$\log_2$(n)	$\log_2$(T(n))
32	210455,5556	5	17,68315607
64	625722,2222	6	19,25516282
128	688766,6667	7	19,3936558
256	1537700	8	20,55234264
512	5829544,444	9	22,47495172
1024	10089666,67	10	23,26637518
2048	31576700	11	24,91235707
4096	132393355,6	12	26,98025548
8192	482.030.578	13	28,84454943



En la gráfica podemos observar que la ecuación de la recta es: ***y = 1, 3595x + 10,36***, por lo que su pendiente es k = 1,3595 y el algoritmo tendría orden. $O(n^{1,3595})$.

Teóricamente, el orden de complejidad de dicho algoritmo sería $O(n^2)$, pero no coincide porque con nuestros computadores personales no podemos tomar valores lo suficientemente grandes al hacer un análisis empírico.

Ejercicio 5: Constante de implementación para el algoritmo recursivo

$$T_{Recursivo} \in O(4^n) \rightarrow \bar{T}(20) \leq c \cdot 4^n \leftrightarrow 603217841322{,}222 \leq c \cdot 4^{20}$$

$$c = 0{,}5486 \rightarrow \boldsymbol{\bar{T}(n) \leq 0,5486 \cdot 4^n}$$

Ejercicio 6: Tiempo teórico de cada algoritmo para n = 500

- Algoritmo Recursivo: $\bar{T}(n) \leq 0{,}5486 \cdot 4^n$

$$\bar{T}(500) \leq 0{,}5486 \cdot 4^{500}\ ns \cdot \frac{1\ s}{10^9\ ns} \cdot \frac{1\ h}{3600\ s} \cdot \frac{1\ día}{24\ h} \cdot \frac{1\ año}{365\ días} =$$

$$= \boldsymbol{1.8641 \cdot 10^{284}\ años}$$

- Algoritmo Dinámico: $\bar{T}(n) \leq c \cdot n^{1,3595}$

$$\bar{T}(8192) \leq c \cdot 8192^{1,3595}$$

$$482030577{,}7778 \leq c \cdot 8192^{1,3595}$$

$$c = 2305{,}7964$$

$$\bar{T}(n) \leq 2305{,}7964 \cdot n^{1,3595}$$

$$\bar{T}(500) \leq 2305{,}7964 \cdot 500^{1,3595}$$

$$\bar{T}(500) \leq 1{,}0766 \cdot 10^7\ ns$$

Ejercicio 7: Conclusiones sobre la eficiencia de ambos algoritmos

Como hemos podido observar en las tablas y gráficas anteriores, existe una gran diferencia entre el tiempo de ejecución de un algoritmo y otro.

En este problema en concreto, la solución recursiva es completamente ineficiente puesto que repite muchos cálculos. El algoritmo de programación dinámica es más eficiente, ya que almacena los coeficientes que se van calculando en una matriz y combinando las soluciones para subproblemas más pequeños. Así se evita el problema de la técnica divide y vencerás de calcular varias veces las soluciones de mismos problemas pequeños.