ALGORITMIA

Grado en Ingeniería Informática del Software - Curso 2025-2026

GUION DE LA PRÁCTICA 3 (directorio p3)

0)INTRODUCCIÓN

Se abordará el estudio de la complejidad temporal y el tiempo de ejecución de algoritmos recursivos Divide y Vencerás (**DyV**), tanto para el esquema de **Sustracción** como para el de **División**.

La segunda parte de la práctica consiste en dar solución a un problema concreto mediante la técnica DyV, medir los tiempos de ejecución del algoritmo que se implemente y proceder al análisis final de los tiempos obtenidos.

Seguiremos en esta práctica midiendo los tiempos SIN_OPTIMIZACIÓN (excepto en el problema planteado al final), ya que incluso en este caso de modelos recursivos el optimizador JIT parece que produce mayor distorsión de tiempos que el caso de modelos iterativos (vistos en las sesiones anteriores). En las tablas de tiempos que nos pidan, si cualquier tiempo se alarga más de 1 minuto pondremos Fuera de Tiempo (FdT).

1)DyV por SUSTRACCIÓN

Las clases **Sustraccion1.java** y **Sustraccion2.java** tienen un esquema DyV por *SUSTRACCIÓN* con a**=1**, lo que lleva consigo un **gran gasto de pila**. Afortunadamente, los problemas así solucionados, van a tener en la práctica una mejor solución iterativa (con bucles) que la solución recursiva de este tipo (a=1).

La clase **Sustraccion3.java** tiene un esquema por *SUSTRACCIÓN* con a**>1**, lo que lleva consigo un **gran tiempo de ejecución** (exponencial o no polinómico). Esto supone que para un tamaño del problema de algunas decenas el algoritmo no acaba (*tiempo intratable, NP*). La consecuencia es que debemos procurar no plantear soluciones del tipo *SUSTRACCIÓN* con varias llamadas (a>1).

Procede:

```
javac *.java
dir
java -Xint Sustraccion1 1 //10,100,…
java -Xint Sustraccion2 1 //10,100,…
java -Xint Sustraccion3 1 //10,100,…
```

SE LE PIDE:

Tras analizar la complejidad de las tres clases anteriores, no se le pide hacer sus tablas de tiempos, pero sí razonar si los tiempos coinciden o no la complejidad temporal de cada algoritmo.

Contestar: ¿para qué valor de n dejan de dar tiempo (abortan) las clases Sustracción1 y Sustracción2? ¿por qué sucede eso?

Calcular: ¿cuántos años tardaría en finalizar la ejecución Sustracción3 para n=80? Razonar la respuesta.

Implementar una clase Sustracción4.java con una complejidad $O(n^3)$ (con su pertinente toma de tiempos) y después rellenar una tabla en la que se muestre el tiempo (en milisg.) para n=100, 200, 400, 800, … (hasta FdT).

Implementar una clase Sustracción5.java con una complejidad $O(3^{n/2})$ (con su pertinente toma de tiempos) y después rellenar una tabla en la que se muestre el tiempo (en milisg.) para n=30, 32, 34, 36, … (hasta FdT).

Calcular: ¿cuántos años tardaría en finalizar la ejecución Sustracción5 para n=80? Razonar la respuesta.

2)DyV por DIVISIÓN

Las clases **Division1.java, Division2.java y Division3.java** tienen un esquema por DyV por División, respectivamente del tipo $a<b^k$, $a=b^k$ y $a>b^k$ (ver cómo se definen estas constantes en teoría).

Procede:

```
dir
java -Xint Division1 1 //10,100,…
java -Xint Division2 1 //10,100,…
java -Xint Division3 1 //10,100,…
```

SE LE PIDE:

Tras analizar la complejidad de las tres clases anteriores, no se le pide hacer sus tablas de tiempos, pero sí razonar si los tiempos coinciden o no la complejidad temporal de cada algoritmo.

Implementar una clase Division4.java con una complejidad $O(n^2)$ (con $a<b^k$) y la pertinente toma de tiempos. Después rellenar una tabla en la que se muestre el tiempo (en milisg.) para n=1000, 2000, 4000, 8000, … (hasta FdT).

Implementar una clase Division5.java con una complejidad $O(n^2)$ (con $a>b^k$) y la pertinente toma de tiempos. Después rellenar una tabla en la que se muestre el tiempo (en milisg.) para n=1000, 2000, 4000, 8000, … (hasta FdT).

3)DOS EJEMPLOS BÁSICOS

Cada una de las clases **VectorSuma.java y Fibonacci.java** resuelven un problema básico de diversas formas, la primera sumar los elementos de un vector de n componentes y la segunda calcular el número de Fibonacci de un orden n dado.

SE LE PIDE:

Tras analizar la complejidad de los diversos algoritmos que hay dentro de las dos clases, ejecutarlas y tras poner en una tabla los tiempos obtenidos, comparar la eficiencia de cada algoritmo.

4)PRÁCTICA DyV

Inicialmente existen n puntos (n es potencia de 2: 4, 8, 16, 32, …) en un espacio bidimensional de tamaño 100*100, en el que cada punto queda definido por sus coordenadas reales (x,y), cada una de ellas con un precisión de 6 decimales.

El problema que se nos pide resolver es calcular los dos puntos más próximos y la distancia que los separa (evidentemente se trata de distancias en "línea recta" y en el "caso de empate" de distancias mínimas entre pares de puntos nos vale una cualquiera de las soluciones empatadas).

Nos dan un juego de datos de entrada de 5 ficheros:

"**datos32.txt**" : para este caso la solución ha de ser:

PUNTOS MÁS CERCANOS: [27.278446, 90.131051] [27.536642, 89.236468]

SU DISTANCIA MÍNIMA= 0.931098

"**datos128.txt**" : para este caso la solución ha de ser:

PUNTOS MÁS CERCANOS: [27.278446, 90.131051] [27.536642, 89.236468]

SU DISTANCIA MÍNIMA= 0.931098

“**datos512.txt**”

PUNTOS MÁS CERCANOS: [68.474715, 94.638993] [68.511846, 94.653169]

SU DISTANCIA MÍNIMA= 0.039745

“**datos2048.txt**”

PUNTOS MÁS CERCANOS: [27.692774, 55.78312] [27.686908, 55.818825]

SU DISTANCIA MÍNIMA= 0.036184

“**datos8192.txt**”

PUNTOS MÁS CERCANOS: [83.681527, 10.289108] [83.691485, 10.28876]

SU DISTANCIA MÍNIMA= 0.009964

SE LE PIDE:

PRIMERA PARTE

Resolver el problema mediante un algoritmo trivial con una clase **p3p.PuntosTrivial.java**, que calcule la solución calculando la distancia entre todo par de puntos diferentes, para así saber al final el par de puntos buscado y su distancia mínima.

Ejecutar la clase metiendo como parámetro cada uno de los nombres de los ficheros aportados, para comprobar el buen funcionamiento del algoritmo implementado:

java p3p.PuntosTrivial nombreFichero // [datos32.txt .. datos8192.txt]

Hacer una clase **p3p.PuntosTrivialTiempos.java** que tras generar de forma aleatoria los n puntos, posteriormente calcule el tiempo de ejecución del algoritmo trivial:

El tiempo en milisegundos, con optimizador activado y se pone FdT cuando pasa del minuto

n	***tiempo***
1024	…
2048	…
4096	…
(hasta que tarde más de un minuto)	

Razonar si sintonizan los tiempos obtenidos con la complejidad del algoritmo.

SEGUNDA PARTE

Resolver el problema mediante un algoritmo Divide y Vencerás con una clase **p3p.PuntosDyV.java** que permita, dividiendo el problema en subproblemas, saber al final el par de puntos buscado y su distancia mínima.

Ejecutar la clase metiendo como parámetro cada uno de los nombres de los ficheros aportados para comprobar el buen funcionamiento del algoritmo implementado:

java p3p.PuntosDyV nombreFichero // datos32.txt, …, datos2048.txt

Hacer una clase **p3p.PuntosDyVTiempos.java** que tras generar de forma aleatoria los n puntos, posteriormente calcule el tiempo de ejecución del algoritmo DyV:

El tiempo en milisegundos, con optimizador activado y se pone FdT cuando pasa del minuto

n	***tiempo***
1024	…
2048	…
4096	…
(hasta que tarde más de un minuto)	

Razone si sintonizan los tiempos obtenidos con la complejidad del algoritmo.

De forma opcional, puede hacer una clase p3p.PuntosTodosDyV.java que resuelva el problema planteado para cualquier número de puntos n>=3.

Pruebe el buen funcionamiento de **p3p.PuntosTodosDyV.java** para:

"**datos100.txt**" : para este caso la solución ha de ser:

PUNTOS MÁS CERCANOS: [40.203173, 17.809812] [39.518531, 18.864816]

SU DISTANCIA MÍNIMA= 1.257684

"**datos1000.txt**" : para este caso la solución ha de ser:

PUNTOS MÁS CERCANOS: [8.036015, 65.015139] [8.005602, 64.992083]

SU DISTANCIA MÍNIMA= 0.038165

"**datos10000.txt**" : para este caso la solución ha de ser:

PUNTOS MÁS CERCANOS: [73.085395, 31.480901] [73.085113, 31.479733]

SU DISTANCIA MÍNIMA= 0.001202

Se ha de entregar en un **.pdf** la memoria del trabajo que se le pide y además las clases **.java** que ha programado. Todo ello lo organizará en una carpeta, que es la que entregará comprimida en un fichero **p3ApellidosNombre.zip**.

La entrega de esta práctica se realizará, en tiempo y forma, según las indicaciones dadas por el profesor de práctica.