UNIVERSITAT POLITÈCNICA DE VALÈNCIA

CAMPUS D'ALCOI

Programación

Práctica 2: Complejidad

Josué Ferri Pascual

jferri@dsic.upv.es

Práctica 2

➔ **Parte 1 - Mejorar el coste de un algoritmo**

➔ **Parte 2 - Calcular coste temporal para tallas grandes**

➔ **Comandos Gnuplot**

Recordatorio

```
// metodo 1
int m = n * n;
m1.n_a_asignaciones++;
m1.n_op_multiplicacion++;
```

```
// metodo 2
m2.n_a_asignaciones++;
m = 0;

m2.n_a_asignaciones++;
m2.n_c_comparaciones++;
for (int i = 0; i < n; i++) {
    m2.n_c_comparaciones++;
    m2.n_op_suma++;

    m = m + n;
    m2.n_a_asignaciones++;
    m2.n_op_suma++;
}
```

```
// metodo 3

m3.n_a_asignaciones++;
m = 0;

m3.n_a_asignaciones++;
m3.n_c_comparaciones++;
for (int i = 0; i < n; i++) {
    m3.n_c_comparaciones++;
    m3.n_op_suma++;


    m3.n_a_asignaciones++;
    m3.n_c_comparaciones++;
    for (int j = 0; j < n; j++) {
        m3.n_c_comparaciones++;
        m3.n_op_suma++;

        m++;
        m3.n_a_asignaciones++;
        m3.n_op_suma++;
    }
}
```

Parte 1 - Mejorar el coste de un algoritmo

- ➔ 1.1 Búsqueda lineal en una matriz
- ➔ 1.2 Búsqueda lineal exitosa en la matriz
- ➔ 1.3 Búsqueda binaria en una matriz
- ➔ 1.4 Búsqueda binaria recursiva en una matriz

Búsqueda Lineal (matriz)

➔ Implementa un algoritmo iterativo que

- Dada una matriz de enteros y un valor
- Realice una búsqueda de ese valor, de manera lineal
- Iterando por **todos los elementos**, desde el valor en la posición [0,0] hasta el valor en la posición [n_rows-1, n_cols-1].

➔ Ayudas

- Función crea matriz creciente
- Función para imprimir matriz

```
public static int[] linearMatSearch(int[][] mat, int valor) {
        int counter = 0;
        int[] result = { -1, -1, counter };
        int x = mat[0].length; // filas
        int y = mat.length; // columnas
                ...
}
```

➔ Preguntas

- ¿Coste temporal del algoritmo para caso mejor y peor?
- Si elemento no se encuentra, ¿cuántas iteraciones?

Búsqueda Lineal Exitosa (matriz)

➔ Implementa una versión mejorada

- Cuando el valor es localizado devuelve la posición de las coordenadas
- Y **finaliza** la búsqueda

```
public static int[] successfulMatSearch(int[][] mat, int v) {
        int counter = 0;
        int[] result = { -1, -1, counter };
        int x = mat[0].length; // filas
        int y = mat.length; // columnas
        ...
}
```

➔ Cálculo promedio

- Dependerá de la posición

➔ Preguntas

- ¿Coste temporal del algoritmo para caso mejor y peor?
- ¿El valor promedio ha mejorado respecto al algoritmo anterior?

Búsqueda Binaria (matriz)

```
public static int[] getCoordinates(int index, int width) {...}
```

➔ Adaptación

- Sólo es válido valores ordenados
- Matriz (2 dimensiones) -> Array (1 dimensión)

➔ Funcionamiento búsqueda

1. Definir límites: dos índices, uno para el inicio del array y otro para el final.
2. Búsqueda del elemento medio
3. Comparación con el elemento medio:
   a) Si el elemento en la posición media del array -> Devolvemos posición
   b) Si no reajustamos los límites:
4. Repetición del proceso: Repetimos los pasos 2 a 4.
   a) Hasta encontrarlo
   b) O que no exista

➔ Preguntas

- ¿Coste temporal del algoritmo para caso mejor y peor?
- ¿Para una matriz de 50x50 que coste tendría, si el elemento está en la mitad? (Extrapola a m x n)
- ¿Para una matriz de 50x50 que coste tendría, si el elemento no existe? (Extrapola a m x n)

Búsqueda Binaria Recursiva (matriz)

➔ Implementación recursiva

```
public static int[] binaryMatRecursiveSearch (int[][] mat, int target, int
inicio, int fin){
	int[] result = { -1, -1, 0 };
	if(inicio <= fin) {
		...
		result[2]++;
	}
	return result;
}
```

➔ Preguntas

- ¿En qué se diferencia el coste temporal del algoritmo para caso mejor y peor?
- ¿Qué ocurre si empleamos matrices muy grandes?

Parte 2 - Calcular coste temporal para tallas grandes

- ➔ 2.1 Implementando la búsqueda lineal exitosa
- ➔ 2.2 Toma de tiempos y verificación empírica
- ➔ 2.3 Obteniendo una función de aproximación para predicciones
- ➔ Comandos Gnuplot

Búsqueda Lineal Exitosa (array)

➔ **Implementación búsqueda lineal array**

```java
public static int[] linealSearchIterative(int[] array, int valor) {
	int pos = -1;
	int iterations= 0;
	int[] result = { pos, iterations};
	boolean found = false;
	for (int i = 0; i < array.length && !found; i++) {
		iterations++;
		if (valor == array[i]) {
			pos = i;
			found = true;
		}
	}
	result[0]=pos;
	result[1]=iterations;
	return result;
}
```

Toma de tiempos algoritmo anterior (2.2.1)

➔ Pasos

1. Tomar el tiempo actual timeStart
2. **Varias veces** ejecutar el algoritmo (e.g., búsqueda lineal)
3. Tomar el tiempo actual timeEnd
4. Calcular la diferencia entre timeEnd y timeStart
5. Realizarlo para varias tallas

```
final int INIT_SIZE = 100000,
          MAX_SIZE = 1000000,
          STEP = 100000,    // A
          REPS = 1000;
```

➔ Resultado esperado por consola

Talla;	Mejor;	Peor;	Promedio
100000;	12;	48659;	31798
200000;	14;	72013;	54752
300000;	28;	106481;	80571
400000;	18;	135909;	107609
500000;	17;	176938;	134735
600000;	50;	209083;	165895
700000;	30;	240212;	193659
800000;	20;	284051;	228191
900000;	18;	317829;	265926
1000000;	21;	377093;	293272

Toma de tiempos algoritmo anterior (2.2.2)

➔ **Exportar a fichero**

```
PrintStream csvPrintStream = null;
try {
	PrintStream csvPrintStream = new PrintStream(new
FileOutputStream("output.csv"));
} catch (FileNotFoundException e) {
	System.err.println("Error: No se pudo abrir archivo CSV" + e);
}
```

```
csvPrintStream.printf("%10s;%10s;%10s;%10s\n", "Talla", "Mejor", "Peor",
"Promedio");
	...
csvPrintStream.printf("%10d;%10d;%10d;%10d\n", size, tBest, tWorst, tAvr);
```

Función de aproximación para predicciones

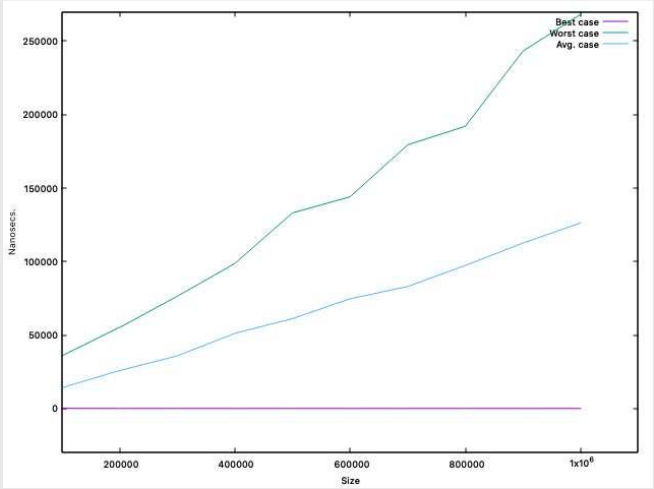
➔ Objetivo

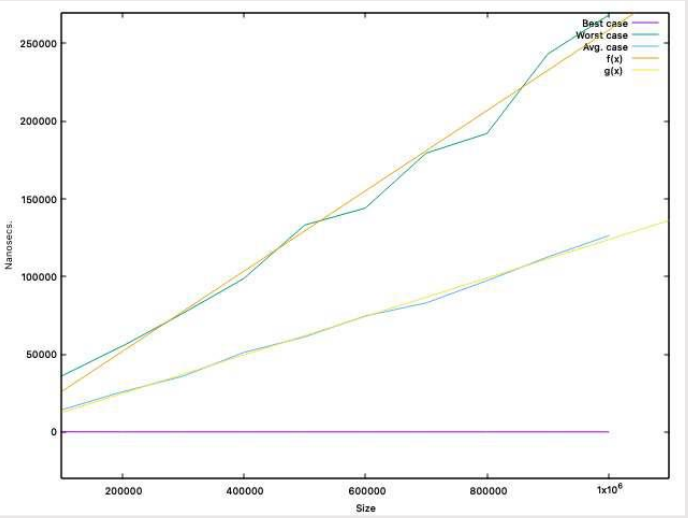
- Predecir el tiempo de ejecución

➔ Dibujar gráfica (x=talla, y=tiempo)

➔ Calcular f(x)=ax+b

```
f(x) = a*x+b
g(x) = c*x+d
fit f(x) "output.csv" using 1:3 via a, b
# ... some output
fit g(x) "output.csv" using 1:4 via c, d
# ... some output
```





Comandos GNUPLOT

➔ **Seleccionar carpeta**

➔ **Imprimir gráficos**

```
plot "output.csv" using 1:3 title "Worst case" with lines

replot "output.csv" using 1:2 title "Best case" with linespoints
```

➔ **Calculo funciones**

```
f(x) = a*x+b
g(x) = c*x+d
fit f(x) "output.csv" using 1:3 via a, b
# ... some output
fit g(x) "output.csv" using 1:4 via c, d
# ... some output
```

Práctica 2: Complejidad

Josué Ferri Pascual

jferri@dsic.upv.es