Programación

Práctica 2

Complejidad

Contenido

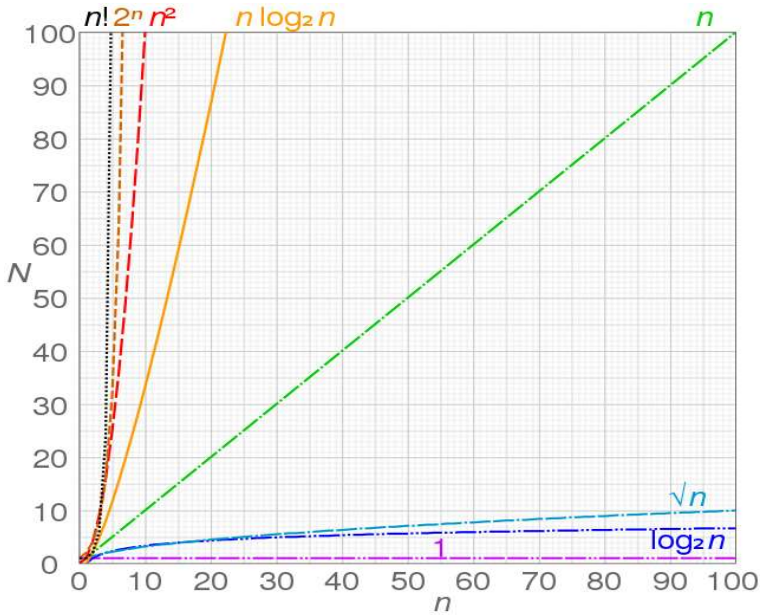
Introducción y Objetivos

La práctica pretende ser un complemento a la parte teórica del curso y también guiar, paso a paso, a la hora de considerar la eficiencia de cualquier solución.

- En esta práctica se llevará a cabo una introducción a la complejidad y eficiencia a la hora de desarrollar algoritmos
- La eficacia y eficiencia deben ser siempre dos de los principales objetivos de un programador
- El alumno debe ser capaz de analizar las diferentes soluciones a un problema dado, y ser capaz de dar una respuesta formal a qué solución es la mejor dado un contexto específico
- e.g., Elegir una implementación específica de una estructura de datos, con múltiples implementaciones, según el tipo de operaciones que se vayan a realizar sobre ella

Objetivos

1. Reflexionar sobre costes y optimización antes, durante y después del desarrollo de algoritmos
2. Verificar empíricamente que las demostraciones teóricas respecto al coste temporal de los algoritmos se cumplen (e.g., ejecutar y comparar tiempos)
3. Inferir funciones aproximadas que definan el comportamiento temporal de un algoritmo, para comparar y predecir su coste frente a tallas grandes



src: https://en.wikipedia.org/wiki/Analysis_of_algorithms

Parte 1 - Mejorar el coste de un algoritmo

1.1 Búsqueda lineal en una 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].

- La matriz puede no ser cuadrada (mxn),
- En este caso los valores estarán ordenados, pero podrían no estarlo,
- No habrá valores repetidos,
- La búsqueda finalizará cuando se encuentre el elemento.

El algoritmo devolverá el par [fila, columna] con la posición, o el par [-1,-1] si tal elemento no se encuentra. Se adjunta la firma de la función:

```
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
            ...
}
```

Para calcular la complejidad añade un contador que se incremente dentro de la instrucción crítica. Cuando la función devuelva el resultado añade el valor del contador.

Puedes usar la siguiente función para crear matrices de tamaño nxm:

```
    public static int[][] generateOrderedMatrix(int m, int n) {
        Random rand = new Random();
        int[][] matrix = new int[m][n];
        int value = 0;
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                value +=rand.nextInt(9)+1; //Inc. aleatorio entre 1 y 10
                matrix[i][j] = valor;
            }
        }
        return matrix;
    }
```

Para presentar los resultados se recomienda una primera línea con los identificadores

```
System.out.printf("%s;%s;%s;%s\n", "Descripcion", "Talla", "Iteraciones",
"Resultado");
```

y a continuación los resultados, todo separado por ";" para facilitar su exportación que se realizará posteriormente.

```
System.out.printf("%s;%d;%d;%s\n", "linearMatSearchMejor", m*n, result[2],
"["+result[0]+","+result[1]+"]");
```

Puedes imprimir la matriz de la siguiente forma para verificar el funcionamiento:

```
for (int i = 0; i < m; i++) {
	for (int j = 0; j < n; j++) {
		System.out.print(matriz[i][j] + " ");
	}
	System.out.println(",");
}
```

Pregunta 1.1.1

¿Cuál es el coste temporal del algoritmo para los casos mejor y peor? Justifícalo mediante el número de iteraciones.

Pregunta 1.1.2

Si la matriz es de tamaño 50x50, y el elemento no se encuentra en ella, ¿cuántas iteraciones realiza el algoritmo? Demuéstralo también contando iteraciones.

1.2 Búsqueda lineal exitosa en la matriz

Implementa una versión mejorada del algoritmo anterior iterativo. Para ello considera que cuando el valor es localizado devuelva la posición de las coordenadas y finalice 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
	...
}
```

En este caso vamos a calcular también el valor promedio, para ello deberás calcular las iteraciones necesarias para cada uno de los valores y realizar la media de los resultados.

```
int iterationSum=0;
for (int i = 0; i < m; i++) {
	for (int j = 0; j < n; j++) {
		result=successfulMatSearch(matrix,matrix[i][j]);
		iterationSum+=result[2];
	}
}
System.out.printf("%s;%d;%d;%s\n", "successfulMatSearchPromedio", m*n,
iterationSum/((m*n)-1), "");
```

Pregunta 1.2.1

¿Cuál es el coste temporal del algoritmo para los casos mejor y peor? Justifícalo mediante el número de iteraciones.

Pregunta 1.2.2

¿En el caso del valor promedio, ha mejorado respecto al algoritmo anterior? Justifícalo.

1.3 Búsqueda binaria en una matriz

Implementar una versión mejorada del algoritmo anterior. En este caso se implementará el algoritmo de búsqueda binaria de una matriz. Este algoritmo sólo funciona si la matriz se encuentra ordenada.

Pasos para implementar una propuesta de solución al problema

1. El primer paso será considerar la matriz 2D a un array 1D.
2. Para ello implementar una función `getCoordinates` capaz de, a partir de un índice y el ancho de la matriz cuadrada, sea capaz de obtener las coordenadas de ese índice en un par [row, col]. e.g., índice = 7, ancho = 3, devolvería [2,1]
    - Dividiendo el índice entre el ancho obtendríamos la fila
    - El resto de la división entero del índice entre el ancho nos devolvería la columna
3. Utilizando este algoritmo, implementa una pseudo-búsqueda binaria, es decir, buscando en la posición central de la matriz "aplanada" o "estirada", y buscando en la parte izquierda o derecha, según la comparación del valor en el centro respecto al valor que queremos buscar

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

public static int[] binaryMatSearch(int[][] matrix, int target) {

	// 1. Calculate number of rows
	// 2. Calculate number of columns
	// 3. Set the start index as 0
	// 4. Set the last index as the position of the last element

	// 5. While the start index is lower or equals than the last index
		// 5.1 Calculate the index for the mid position, as an integer
		// 5.2 Translate this index to 2D coordinates (using your function)
		// 5.3 Get the value from the matrix in the mid position

		// 5.4 If the value at the mid position is the same as target
			// 5.5 Return the 2D coordinates where it is located

		// 5.6 Else-If the value at the mid position is lower than the target
			// 5.7 Set the start index to the mid index + 1

		// 5.8 Else (the value at the mid position is higher than the target)
			// 5.9 Set the last index to mid - 1

	// 6. Executing this code means target is not in the array, return [-1,-1]
}
```

Pregunta 1.3.1

Para este algoritmo, ¿existe caso mejor y caso peor respecto al coste temporal? Explica tu respuesta y compara con los resultados de los algoritmos anteriores.

Pregunta 1.3.2

Si la matriz es de tamaño 50x50, y el elemento se encuentra en el centro, ¿cuántas iteraciones realiza el algoritmo? Demuéstralo con un ejemplo en el código.

Pregunta 1.3.3

Si la matriz es de tamaño 50x50, y el elemento no encuentra en ella, ¿cuántas iteraciones realiza el algoritmo? Demuéstralo con un ejemplo en el código.

1.4 Búsqueda binaria recursiva en una matriz

Implementar la búsqueda binaria de una matriz anterior, pero de forma 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;
}
```

Pregunta 1.4.1

Para este algoritmo, Compara los resultados obtenidos de caso mejor, caso peor y caso promedio con el algoritmo anterior. ¿En qué se diferencia?

Pregunta 1.4.2

¿Qué ocurriría si empleamos matrices mucho más grandes? Compruébalo.

Parte 2 - Calcular coste temporal para tallas grandes

2.1 Implementando la búsqueda lineal exitosa

En este ejercicio, se enseñará al alumno a verificar empíricamente el coste temporal de alguno de los algoritmos vistos en clase, utilizando como ejemplo didáctico el **algoritmo de búsqueda.**

Este algoritmo es utilizado habitualmente cuando **no se tiene información sobre el orden de los elementos** en el array. Este algoritmo consiste en iterar por todas las posiciones del array, desde la posición 0 hasta la posición length-1, parando cuando el valor es encontrado por primera vez.

Como bien sabemos, este algoritmo tiene un coste temporal diferente para los casos mejor y peor:

- Caso mejor: Ω(1) //el valor buscado está en la primera posición
- Caso peor: O(N) //el valor buscado está en la última posición del array

Implementa este algoritmo de búsqueda lineal exitosa sobre un vector siguiendo este código.

```
	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;
	}
```

Emplea una función de creación de un vector con número ordenados similar al empleado en la matriz del ejercicio anterior. A continuación, se presenta la firma del método:

```
public static int[] generateOrderedArray(int n) { ... }
```

2.2 Toma de tiempos y verificación empírica

En esta sección queremos demostrar empíricamente que el coste temporal del algoritmo de búsqueda lineal se comporta como esperamos frente a diferentes tallas (grandes), como se ha demostrado en la parte teórica de la asignatura.

Para poder evaluar empíricamente el coste temporal del algoritmo debemos realizar **experimentos** y **tomar tiempos**.

Ejercicio 2.2.1

De manera genérica, para medir el tiempo de un algoritmo seguimos los siguientes pasos:

1. Tomar el tiempo actual `timeStart`
2. Ejecutar el algoritmo (e.g., búsqueda lineal)
3. Tomar el tiempo actual `timeEnd`
4. Calcular la diferencia entre `timeEnd` y `timeStart`

En Java, podemos obtener el tiempo actual con **`System.nanoTime()`** (DOC)

Para este ejercicio, deberás realizar la **toma de tiempos** del algoritmo **búsqueda lineal iterativa**:

- La medida del tiempo deberá ser calculada para los 3 casos (*mejor*, *peor* y *promedio*)
- Deberás hacer la medida de tiempo varias veces y calcular la media de los tiempos para obtener una medida más fiable, independiente de la carga del sistema en un instante de tiempo concreto
- Estos tiempos deberán ser calculados para las diferentes tallas

Antes de empezar a evaluar el tiempo del algoritmo, define las siguientes variables:

```
final int INIT_SIZE = 100000,   // Talla inicial
        MAX_SIZE = 1000000,     // Talla final
        STEP = 100000,   // A cada medida, incrementamos la talla en STEP
        REPS = 1000;            // Repeticiones para una media más fiable
```

El resultado del ejercicio debe mostrar, por consola, un resultado como el siguiente:

```
                  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
```

Deberás establecer tallas grandes [`INIT_SIZE`, `MAX_SIZE`], dar pasos grandes (`STEP`) y realizar diferentes pruebas (`REPS`) para calcular el tiempo promedio. A continuación, se muestra el caso mejor, implementa el caso peor y el promedio.

```
System.out.printf("%10s;%10s;%10s;%10s\n",  "Talla", "Mejor", "Peor",
"Promedio");
for (int size=INIT_SIZE;size<=MAX_SIZE;size+=STEP) {
	array=generateOrderedArray(size);
	arraySize=array.length;

	vBest=array[0];
	timeStart=System.nanoTime();
	for (int rep=0; rep<REPS; rep++) {
		result=lineal SearchIterative(array, vBest);
	}
	timeEnd=System.nanoTime();
	tBest=(timeEnd-timeStart)/REPS;
	// Implementar el cálculo para el peor valor
	// Implementar el cálculo para el valor promedio
	System.out.printf("%10d;%10d;%10d;%10d\n", size, tBest, tWorst, tAvr);
}
```

Ejercicio 2.2.2

El siguiente ejercicio reside en tratar la visualización de los datos obtenidos de manera gráfica a partir de los datos tabulados.

Para ello el primer paso es exportar el resultado del programa (i.e., lo que se imprime por consola) a un fichero de texto de tipo CSV. Se puede realizar de varias formas: copiando directamente los datos a un fichero, redireccionando desde la consola o imprimiendo desde el código a un fichero, que será lo más adecuado especialmente cuando hay una gran cantidad de datos a exportar.

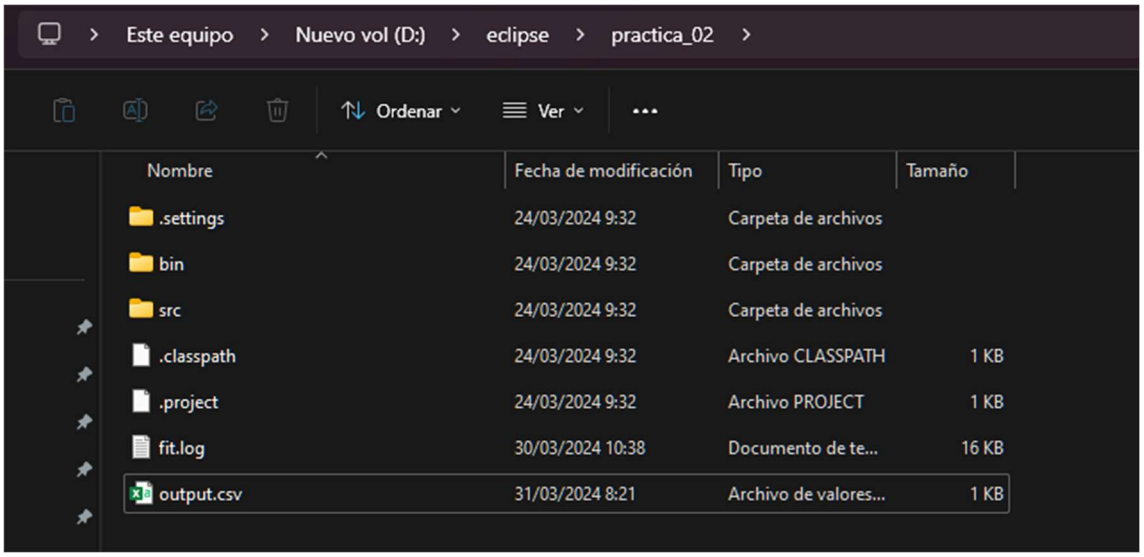
Para ello bastará con crear una salida de bytes hacia un fichero con el siguiente código:

```
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);
}
```

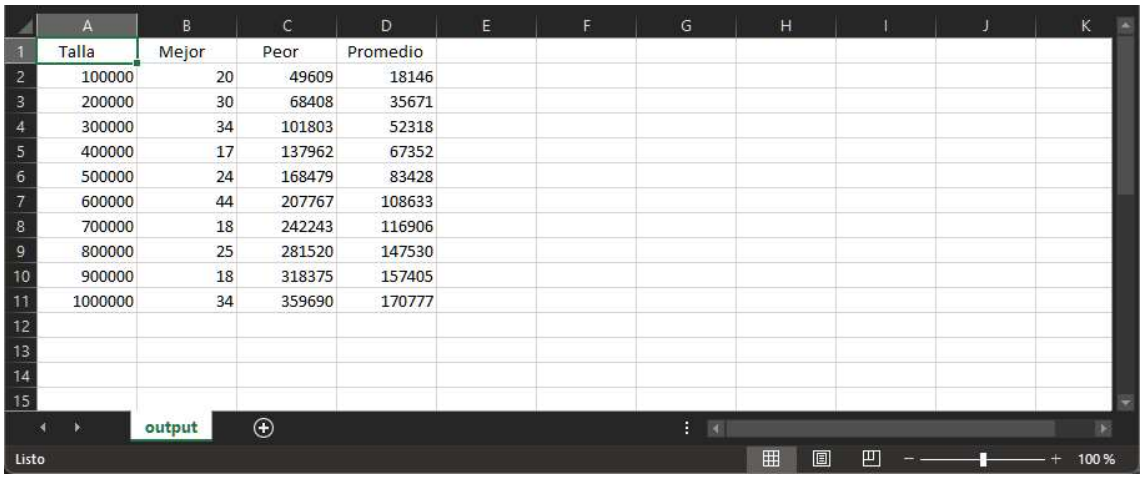
Y posteriormente indicar en los printf relacionados con los resultados, que vayan hacia dicha salida del siguiente modo.

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

De este modo tras la ejecución se creará un fichero “output.csv” en el directorio del proyecto.



Este fichero se puede abrir desde cualquier programa de tipo hoja de cálculo o importar a una base de datos.



	A	B	C	D
1	Talla	Mejor	Peor	Promedio
2	100000	20	49609	18146
3	200000	30	68408	35671
4	300000	34	101803	52318
5	400000	17	137962	67352
6	500000	24	168479	83428
7	600000	44	207767	108633
8	700000	18	242243	116906
9	800000	25	281520	147530
10	900000	18	318375	157405
11	1000000	34	359690	170777

Para ver los resultados de manera gráfica emplearemos la herramienta Gnuplot.

Windows: Instalación en Windows mediante fichero .exe

Linux: Instalación en Linux mediante apt-get

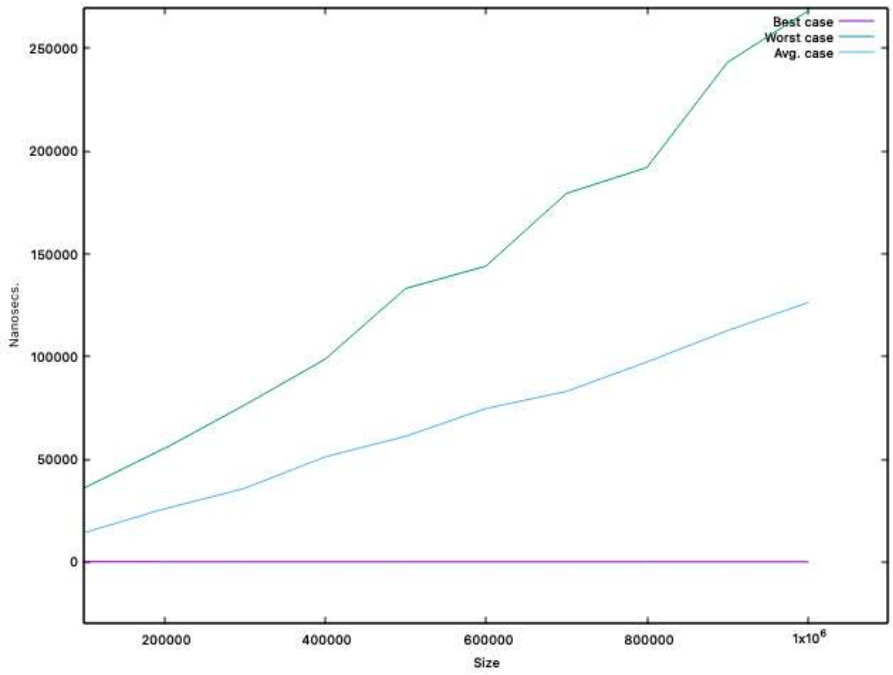
macOS: Instalación en macOS mediante Homebrew

La herramienta Gnuplot se encuentra instalada también en Polilabs VPN

Estudia la sección **Comandos Gnuplot** para más información sobre los comandos disponibles en la herramienta Gnuplot, se encuentra al final de la práctica.

Es interesante que te vayas apuntando los comandos que vas ejecutando en Gnuplot, porque seguramente tengas que repetirlos varias veces.

En base a los comandos vistos de Gnuplot, **genera una gráfica como la siguiente**, representado los casos mejor, peor y promedio, teniendo en cuenta las etiquetas de los ejes:



Análisis gráfico del comportamiento de la búsqueda lineal para el caso mejor (violeta), peor (verde), y promedio (azul)

2.3 Obteniendo una función de aproximación para predicciones

Una vez tenemos la gráfica, a simple vista podría incluso llegar a ser confuso identificar si la tasa de crecimiento es lineal o cuadrática. En este caso, como sabemos que, para el caso peor, la **tasa de crecimiento es lineal**, podemos representarlo mediante la **ecuación de la recta**:

$$f(x)=ax+b$$

Visualización de la recta

Para solucionar este problema, podemos usar el comando `fit` para obtener una función/modelo que muestre un comportamiento aproximado del algoritmo.

En este último apartado vamos a utilizar una regresión lineal para calcular una recta que pueda predecir su comportamiento en los casos PEOR y PROMEDIO (el caso MEJOR no nos interesa, el coste temporal es constante)

Brevemente, tratamos de buscar una función que se aproxime a los datos que tenemos, minimizando el error/distancia de los datos observados a su respectivo valor sobre la recta estimada, y así poder predecir como se comportará el algoritmo frente a tallas mucho más grandes de las evaluadas.

Tras ejecutar el último comando con el cual has obtenido la gráfica sobre los datos obtenidos, ejecuta los siguientes comandos:

```
# Definimos dos funciones (recta, tipo ax+b)
# a y c son las pendientes
# b y d los valores de y cuando x vale 0 (términos independientes)

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
```

```
gnuplot> fit f(x) "data_lin.out" using 1:3 via a,b
iter      chisq       delta/lim  lambda   a             b
   0 2.1161116252e+12   0.00e+00  4.39e+05    1.000000e+00   1.000000e+00
   1 5.4916572837e+09  -3.84e+07  4.39e+04    2.940429e-01   9.999990e-01
   2 6.9479491826e+08  -6.90e+05  4.39e+03    2.587627e-01   1.000000e+00
   3 6.9479371954e+08  -1.73e-01  4.39e+02    2.587450e-01   1.000135e+00
iter      chisq       delta/lim  lambda   a             b

After 3 iterations the fit converged.
final sum of squares of residuals : 6.94794e+08
rel. change during last iteration : -1.72529e-06

degrees of freedom    (FIT_NDF)                        : 8
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 9319.29
variance of residuals (reduced chisquare) = WSSR/ndf   : 8.68492e+07

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a               = 0.258745         +/- 0.01026      (3.965%)
b               = 1.00014          +/- 6366         (6.365e+05%)

correlation matrix of the fit parameters:
                a      b
a               1.000
b              -0.886  1.000
```

Fit caso peor (1:3),

```
gnuplot> fit g(x) "data_lin.out" using 1:4 via c,d
iter      chisq       delta/lim  lambda   c             d
   0 2.9556383693e+12   0.00e+00  4.39e+05    1.000000e+00   1.000000e+00
   1 6.7345207831e+09  -4.38e+07  4.39e+04    1.655437e-01   9.999988e-01
   2 3.2468255838e+07  -2.06e+07  4.39e+03    1.238417e-01   9.999992e-01
   3 3.2466581933e+07  -5.16e+00  4.39e+02    1.238209e-01   1.000040e+00
   4 3.2466575381e+07  -2.02e-02  4.39e+01    1.238209e-01   1.004166e+00
iter      chisq       delta/lim  lambda   c             d

After 4 iterations the fit converged.
final sum of squares of residuals : 3.24666e+07
rel. change during last iteration : -2.01815e-07

degrees of freedom    (FIT_NDF)                       : 8
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 2014.53
variance of residuals (reduced chisquare) = WSSR/ndf   : 4.05832e+06

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
c               = 0.123821         +/- 0.002218     (1.791%)
d               = 1.00417          +/- 1376         (1.37e+05%)

correlation matrix of the fit parameters:
                c      d
c               1.000
d              -0.886  1.000
```

Fit caso promedio (1:4)

El comando `fit` trata de encontrar un valor de *a* y *b* (aplicando el método de mínimos cuadrados) que minimice el error entre la función y los datos, permitiendo obtener una función aproximada/modelo y que pueda ser utilizada para realizar predicciones.

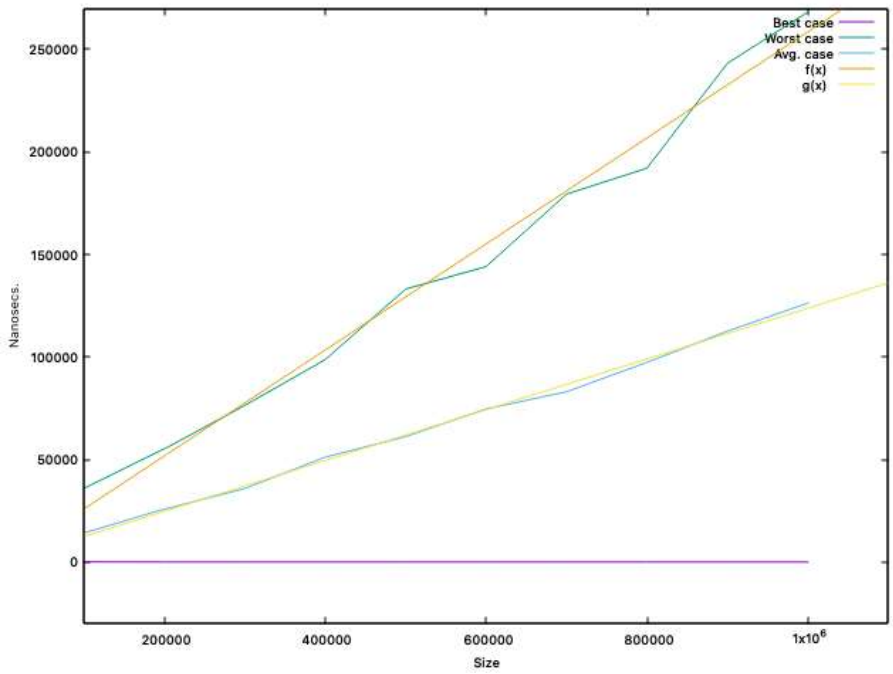
- El comando **fit** realiza varias iteraciones para ajustar esos valores de *a* y *b* (+ info, Regresión lineal y Machine Learning)

Ejercicio 2.3.1

Sustituye los coeficientes *a*, *b*, *c* y *d* en *f(x)* y *g(x)* con los valores estimados y haz un **replot** de *f(x)* y *g(x)* para ver cómo se ajusta la función a los datos.

```
replot f(x)
replot g(x)
```

Deberías obtener una gráfica similar a la siguiente:



Observación empírica y modelo de predicción

Interpretación de los resultados

Si observamos la relación entre las pendientes *a* de *f(x)* y *c* de *g(x)*:

0.258745/0.123821 ≈ 2

Esto significa que la pendiente *a* tiene una tasa de crecimiento el doble de rápida que *c*, y que, por tanto, es el doble de lenta si lo analizamos en relación al tiempo de ejecución (siempre hablando de una tasa de crecimiento lineal en ambos casos).

Además, sabiendo los coeficientes, por ejemplo, del **caso promedio** (1:4), podríamos llegar predecir el tiempo de ejecución para una talla determinada:

e.g., a = 0.123821, b = 1.00417, talla 10^9, a=> pendiente, b => termino independiente u ordenada en el origen

10^9 * 0.123821 + 1.00417 ≈ 123821001 ns ≈ **0.1238 s**

Comandos Gnuplot

Comando `plot`

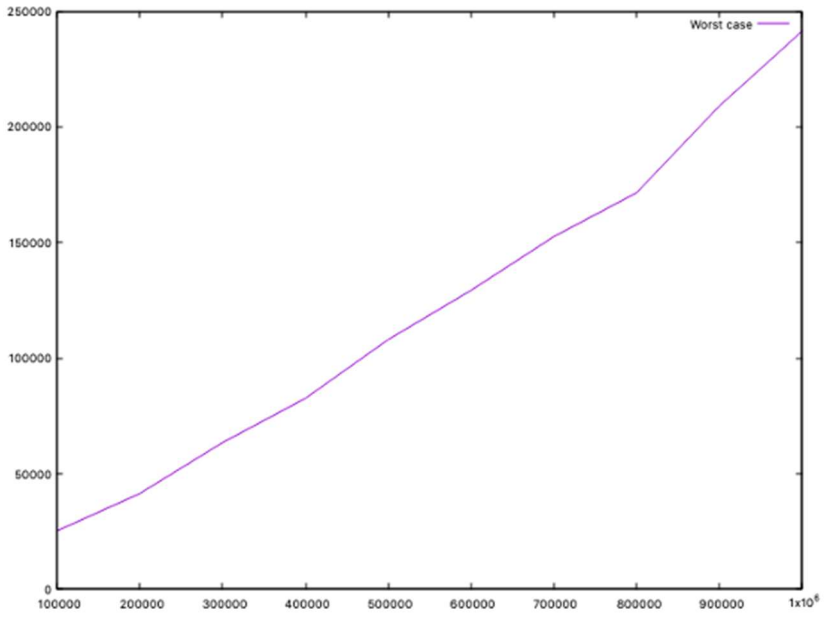
Dibuja datos a partir de un fichero o funciones predefinidas.

Sus modificadores más importantes son:

- "*file*": se especifica entre comillas, e indica dónde se encuentran los datos. Las líneas que empiezan con el símbolo # se ignoran
- **using** *i:j* : Indica las columnas desde las cuales sacaremos los datos (i => eje x, j => eje y), empezando a contar desde 1 (i.e., la columna de la talla es la columna 1, y la de los tiempos serán las columnas 2, 3 y 4)
- **title** *text*: Nombre que aparecerá en la leyenda para los datos representados
- **with** *format*: Forma de representación, a elegir entre `lines`, `points` o `linespoints`

Ejemplo

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

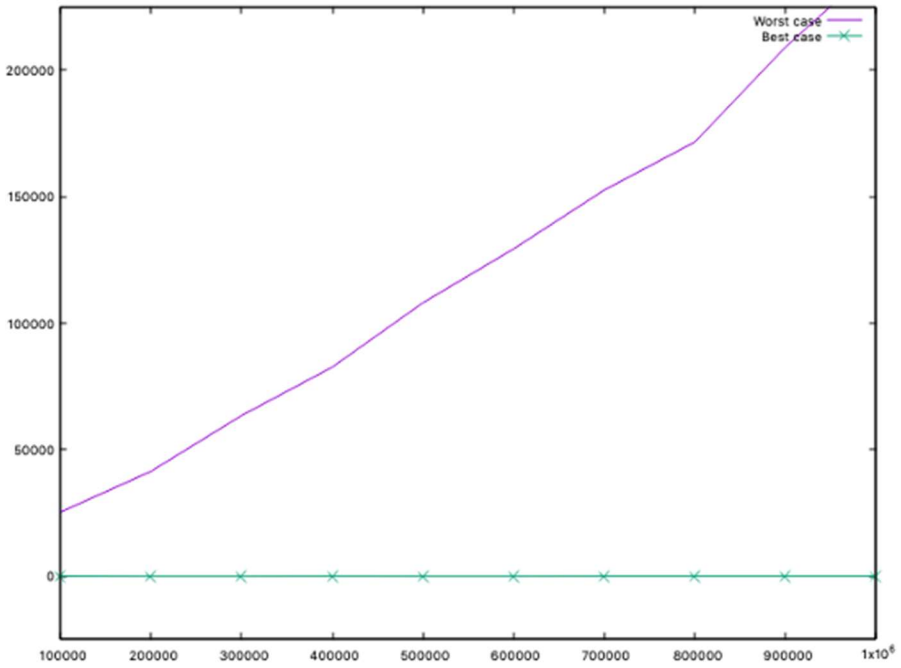


Cambiar al directorio donde tengamos el fichero a través de File>Change Directory

Comando `replot`

Similar a `plot`, pero dibujando sobre la gráfica anterior si la hemos creado.

```
# Misma línea que la del ejemplo anterior
plot " output.csv" using 1:3 title "Worst case" with lines
replot " output.csv" using 1:2 title "Best case" with linespoints
```



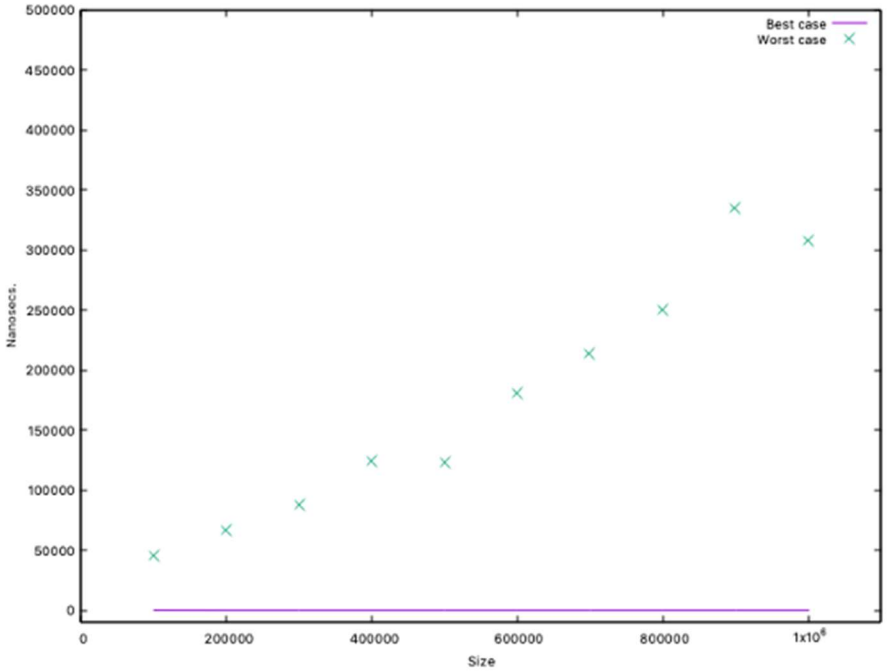
Comando `set`

Este comando permite establecer diferentes elementos de la gráfica, mediante:

- **set xrange** *[start:end]* y **set yrange** *[start, end]*
- Establece el rango para los ejes x e y
- **set xtics** *interval* y **set ytics** *interval*
- Establece el intervalor entre marcas en lo ejes
- **set xlabel** "*text*" y **set ylabel** "*text*"
- Establece etiquetas de texto para cada eje (e.g., la X es la talla y la Y el tiempo)

Ejemplo

```
set xrange [0:1100000]
set yrange [-10000:500000]
set xtics 200000
set ytics 50000
set xlabel "Size"
set ylabel "Nanosecs."
plot "data_lin.out" using 1:2 title "Best case" with lines
replot "data_lin.out" using 1:3 title "Worst case" with points
```



Lo habitual es ir probando diferentes valores para el comando set, y ver cómo afectan a la gráfica actual. Para ello, si ya dispones de una gráfica en pantalla, puedes utilizar cualquier comando `set` y luego el comando `replot` (sin argumentos) para que se repinte la gráfica actual.

Comando load

El comando **load** *file* permite ejecutar los comandos Gnuplot, de manera secuencial, que aparezcan escritos en un fichero de texto.