Prácticas de Algorítmica.
3º de Grado en Ingeniería Informática.
Curso 2025-2026.

Práctica 1

Objetivos.

Con esta práctica se pretende que el alumno se familiarice con el cálculo de tiempos de ejecución de un determinado algoritmo en función del tamaño del ejemplar y hacer una estimación empírica de esos tiempos en función de dicho tamaño. Para ello, el alumno deberá implementar un programa en C++ donde se calculen los tiempos de ejecución de varios algoritmos para distintos tamaños del ejemplar y posteriormente se estime, utilizando un enfoque híbrido, la complejidad temporal de esos algoritmos, obteniendo la función de tiempo en función del tamaño del ejemplar. Esta función nos permitiría hacer estimaciones de tiempos de ejecución del algoritmo para cualquier tamaño del ejemplar.

El programa resultante tendrá un menú con tres apartados obligatorios y uno opcional. Cada apartado se invocará en la correspondiente opción del menú mediante una función de medio nivel que no tendrá ningún parámetro. De esta forma, cada opción se podría tratar como un módulo independiente que podría ser exportado a cualquier otro programa. Los prototipos de dichas funciones serán:

void ordenacionSeleccion();
void ordenacionQuicksort();
void determinanteIterativo();
void ordenacionSeleccionAmpliado();

El programa principal tendrá ***un menú que contendrá solo las llamadas a estas funciones.***

Apartado 1 (2.5 puntos)

1. *Implementación el método de ordenación por selección. Para ello se tendrán en cuenta las siguientes consideraciones:*
    1. *El vector será de elementos de tipo entero y se rellenará aleatoriamente con valores entre 0 y 9999999. Para ello implementad la función de prototipo* ***void rellenarVector(vector<int> &v);*** *Esta función es auxiliar y será invocada cuando haya que rellenar un vector.*
    2. *La función de ordenación tendrá como prototipo* ***void ordenacionSeleccion(vector<int> &v);*** *Esta función es auxiliar y será invocada cuando haya que rellenar un vector.*
    3. *En la función de medio nivel ordenaciónSeleccion() se declararán todas las variables que se usarán como parámetros en las funciones que se invocan desde ella, y que se describen a continuación. Algunas de estas variables se introducirán en esta función y otras se rellenarán en las funciones que se invocan.*
    4. ***void tiemposOrdenacionSeleccion(int nMin, int nMax, int incremento, int repeticiones, vector <double> &tiemposReales, vector <double> &numeroElementos);***.
        1. *Ésta será la primera función a invocar desde ordenacionSeleccion(), en ella se obtendrán los tiempos reales de ordenación para distintos valores del número de elementos. Ambas se almacenarán en vectores de doubles, tal y como se indica en el prototipo. Los cuatro primeros parámetros se habrán introducido en la función ordenacionSeleccion().*
        2. *Este ejemplo indica como proceder: si el mínimo es 1000, el máximo es 5000, el incremento es 100 y el número de repeticiones es 5, se probará primero*

generando 5 vectores de 1000 elementos (un vector nuevo cada vez), se ordenará cada uno de ellos y se calculará la media de tiempos de esas 5 pruebas y ese será el valor correspondiente de tiempo empleado para ese n=1000. Después con 1100 y se repite lo mismo y así hasta llegar a 5000. Esto se hará para todos los posibles valores de n. De esta forma rellenaremos el vector de tiemporReales y el de número de elementos.

3. *En la documentación se adjunta un ejemplo para ver como se calculan los tiempos de ejecución. (Nota: Aunque los valores del numero de elementos son enteros, como se van a usar para calcular los sumatorios y otra serie de operaciones mixtas con números doubles, se almacenarán como doubles para no tener problemas).*

5. ***void ajustePolinomico(const vector <double> &numeroElementos, const vector <double> &tiemposReales, vector <double> &a).***
   1. *Ésta será la segunda función a invocar desde ordenacionSeleccion(), en ella se probará a ajustar un polinomio de segundo grado del tipo.* ***t(n) =a0 + a1*n + a2*n²,*** *ya que la complejidad es cuadrática.*
   2. ***a*** *será un vector de coeficientes de la curva de ajuste**.*** *Esta curva se obtiene usando el ajuste polinómico de grado 2 (Ver ejemplo en el anexo).*
   3. *Para obtener estos coeficientes hay que resolver un sistema de ecuaciones.*
   4. *Las matrices del sistema de ecuaciones se declararán dentro de la función ajustePolinomico() y se rellenarán invocando a la función de prototipo:* ***void calcularMatrices(const std::vector<double> &numeroElementos, const std::vector<double> &tiemposReales, int ordenMatrizSistema, std::vector< std::vector<double> > &matrizCoeficientes, std::vector< std::vector<double> > &matrizTerminosIndependientes)*** *donde ordenMatrizSistema es el orden de la matriz de coeficientes del sistema, matrizCoeficientes es la matriz de coeficientes, y matrizTerminosIndependientes es el vector de términos independientes del sistema almacenado como una matriz de ordenMatrizSistema filas y una columna.*
      1. *Para calcular los sumatorios necesarios para obtener las matrices del sistema, usad una función general con este prototipo* ***double sumatorio(vector <double> &n, vector <double> &t, int expN, int expT);*** *donde n y t son las variables y expN y expT son los exponentes respectivos de las variables en el sumatorio. Cuando solo intervenga una variable en el sumatorio, el exponente de la otra será 0.*
   5. *Una vez rellenadas las matrices del sistema, para resolverlo se invocará a la función* ***void resolverSistemaEcuaciones(vector < vector < double > > matrizCoeficientes, vector < vector < double > > matrizTerminosIndependientes, int ordenMatrizSistema, vector < vector < double > > &X);*** *que devolverá en la matriz X la solución del sistema con ordenMatrizSistema filas y una columna. Esta matriz de una columna se copiará n el vector a correspondiente a los coeficientes del ajuste polinómico.*
   6. *El código para resolver el sistema de ecuaciones que proporciona los coeficientes del ajuste polinómico se suministra con el material de la práctica, en los archivos* ***sistemaEcuaciones.cpp*** *y* ***sistemaEcuaciones.hpp****.*

6. ***void calcularTiemposEstimadosPolinomico(const vector <double> &numeroElementos, const vector <double> &a, vector <double> &tiemposEstimados);*** *Ésta será la tercera función a invocar desde ordenacionSeleccion(). En este caso, la función polinómica ajustada en el paso anterior, cuyos coeficientes se almacenan en el vector a, se usará para obtener los tiempos estimados mediante dicha función. De esta forma, al final de este*

paso, tendremos los tiempos reales de los algoritmos, que son los tiempos obtenidos en el apartado 4, y los tiempos estimados mediante el ajuste por mínimos cuadrados. Para calcular cada uno de los tiempos, usad la función indicada en el apartado 9: ***double calcularTiempoEstimadoPolinomico(const double &n, vector <double> &a)***

7. *Obtener el coeficiente de determinación del ajuste, sabiendo que es la varianza de los tiempos estimados dividida por la varianza de los tiempos reales. Para ello usar la función de prototipo* ***double calcularCoeficienteDeterminacion(const vector <double> &tiemposReales, const vector <double> &tiemposEstimados).*** *Esto dará una idea de la bondad del ajuste. Mientras más se acerque a 1, mejor será el ajuste.*
8. *Para representar la curva del ajuste (tiempos estimados) frente a los tiempos reales, se guardarán en un fichero de texto "****datosFinales.txt"****, tres columnas de datos: vector del numero de elementos (tamaño del ejemplar), vector de tiempos reales y vector de tiempos estimados. Se hará usando el programa* ***gnuplot*** *(se suministra un ejemplo de uso)****.***
9. *Para finalizar, el programa ha de mostrar la ecuación de la curva ajustada por mínimos cuadrados, su coeficiente de determinación y dar la posibilidad al usuario si quiere hacer una estimación de tiempos para un determinado valor del tamaño del ejemplar, en cuyo caso mostrará el tiempo de esa estimación en años, días, minutos y segundos. Esta opción ha de poder repetirse hasta que el usuario introduzca un tamaño de ejemplar igual a 0. Esto es útil cuando el tiempo es muy elevado para un tamaño de ejemplar relativamente grande. Por ejemplo, la ordenación de un vector de 100000 millones de elementos tardaría en calcularse varios días, y mediante la curva ajustada podemos obtener de una forma muy fiable el tiempo que tardaría en calcularse. Para ello usar la función* ***double calcularTiempoEstimadoPolinomico(const double &n, vector <double> &a).*** *Donde a será un vector de coeficientes del polinomio de ajuste. Esta función no es la misma que la del apartado 6, ya que calcula el tiempo para un determinado valor de n, y se debe usar para calcular todos los tiempos del apartado estimados del apartado 6.*

Apartado 2 (3 puntos)

2. *Implementación el método de ordenación quicksort. Para ello se tendrán en cuenta* ***l****as siguientes consideraciones****:***

1. *Los pasos 1, 2, 3 y 4 son similares al apartado anterior.*
2. *La funcion ajustePolinomico() será cambiada por* ***void ajusteNlogN(const vector <double> &numeroElementos, const vector <double> &tiemposReales, vector <double> &a); y la curva de ajuste será t(n) =a0 + a1*n*log(n),*** *ya que la complejidad es nlog(n).*
    1. *El proceso para obtener los coeficientes de la curva de ajuste es similar al del apartado 1, excepto que habrá que hacer un cambio de variable para el vector del número de términos. El cambio será z = n*log(n).*
3. *Se obtendrán los tiempos estimados mediante la función* ***void calcularTiemposEstimadosNlogN(const vector <double> &numeroElementos, const vector <double> &a, vector <double> &tiemposEstimados);*** *que se hará de forma similar al apartado anterior con la salvedad del cambio de variable realizado.*
4. *El resto de pasos son similares a los del apartado 1.*

Apartado 3 (2.5 puntos)

3. *Implementar el cálculo del determinante de una matriz de tipo double, de orden n, por el método de triangularización, teniendo en cuenta las siguientes consideraciones:*
    1. *El código del cálculo del determinante lo podéis obtener del código suministrado para la resolución del sistema de ecuaciones.*
    2. *Las matrices serán de elementos de tipo double y se rellenará aleatoriamente con valores entre 0.95 y 1.05.*
    3. *Realizar las mismas pruebas que en el caso del apartado 1, teniendo en cuenta las siguientes diferencias:*
        1. *Para un valor de n solo se realiza una prueba.*
        2. *Se ajustará un polinomio de grado 3.*
        3. *La función para calcular los tiempos estimados será similar a la del apartado 1*

Apartado 4. Opcional (2 puntos)

Repetir el apartado 1 hasta que obtengáis los tiempos reales. Una vez obtenidos los tiempos reales, seleccionar de forma aleatoria el 80% de los puntos que conforman el número de elementos (tamaño del ejemplar) y sus tiempos reales, y calcular la curva de ajuste para esos puntos.
Usando esa curva de ajuste, obtener los tiempos estimados para el 20% de los puntos restantes. Finalmente, obtener el coeficiente de determinación para ese 20% de puntos.

Anexo

Para estimar los parámetros de un ajuste polinómico de orden m se puede usar el siguiente sistema de acuaciones (http://es.slideshare.net/diegoegas/regresion-polinomial-2512264):

$$
\begin{aligned}
a_0 n &+ a_1 \sum x_i &+ a_2 \sum x_i^2 &+ \dots &+ a_m \sum x_i^m &= \sum y_i \\
a_0 \sum x_i &+ a_1 \sum x_i^2 &+ a_2 \sum x_i^3 &+ \dots &+ a_m \sum x_i^{m+1} &= \sum x_i y_i \\
a_0 \sum x_i^2 &+ a_1 \sum x_i^3 &+ a_2 \sum x_i^4 &+ \dots &+ a_m \sum x_i^{m+2} &= \sum x_i^2 y_i \\
\vdots & & \vdots & & \vdots & \vdots \\
a_0 \sum x_i^m &+ a_1 \sum x_i^{m+1} &+ a_2 \sum x_i^{m+2} &+ \dots &+ a_m \sum x_i^{2m} &= \sum x_i^m y_i
\end{aligned}
$$

En este sistema de ecuaciones las incógnitas son los valores de los a_i. Los valores de la variable independiente x_i se corresponden con el tamaño del ejemplar(valores de numero de elementos en el vector u orden en el caso de la matriz) y los valores de la variable dependiente y_i se corresponden con los tiempos reales (valores de t). La n que aparece en el primer término de la primera ecuación del sistema es el tamaño de la muestra de tiempos con los que estamos trabajando (el número de tiempos medidos).

Cada sumatorio tendrá tantos sumandos como valores de n se hayan usado para el tamaño del ejemplar. Por ejemplo, si al ordenar se usan valores de n desde 1000 hasta 10000, de 1000 en 1000 (11 valores de n), cada sumatorio tendrá 11 elementos.

- *Se suministra el código fuente para resolver un sistema lineal de ecuaciones, cuyo prototipo es:*

void resolverSistemaEcuaciones(vector < vector < double > > A, vector < vector < double > > B, int n, vector < vector < double > > &X);
donde:
A es la matriz de coeficientes de nxn, siendo n el número de incógnitas del sistema.
B es la matriz de terminos independientes de nx1
n es el orden de las matrices, que se corresponde con el número de incógnitas del sistema.
X es el valor de las variables que se obtienen resolviendo el sistema de orden nx1

Nota: De este código se puede extraer el código para resolver el determinante iterativo, si quisiérais usarlo para cualquier otra aplicación.

Declaracion y reserva de matrices usando el tipo vector de la STL
vector < vector < double > > matrizDatos;
matrizDatos = vector< vector< double > >(filas, vector< double >(columnas)); //Matriz de filas x columnas.

Ejemplo práctico 1.
En este ejemplo se va a ajustar un polinomio de grado 2 a una curva que representa tiempos frente a valores de n. Este ejemplo podría ser el de la ordenación de un vector por inserción. En este caso, la curva sería del tipo

$$t(n) = a0 + a1 * n + a2 * n^2.$$

En la imagen se muestran los distintos valores de n y t para 11 observaciones, así como los elementos que componen el sistema de ecuaciones para obtener la curva de ajuste.

En

	n	t(tiempo real)	n^2	n^3	n^4	n*t	$t*n^2$	t(estimado)
	100	900	10000	1000000	100000000	90000	9000000	892,1688
	110	1080	12100	1331000	146410000	118800	13068000	1086,1228
	120	1310	14400	1728000	207360000	157200	18864000	1299,1368
	130	1525	16900	2197000	285610000	198250	25772500	1531,2108
	140	1785	19600	2744000	384160000	249900	34986000	1782,3448
	150	2030	22500	3375000	506250000	304500	45675000	2052,5388
	160	2360	25600	4096000	655360000	377600	60416000	2341,7928
	170	2635	28900	4913000	835210000	447950	76151500	2650,1068
	180	2995	32400	5832000	1049760000	539100	97038000	2977,4808
	190	3320	36100	6859000	1303210000	630800	119852000	3323,9148
	200	3710	40000	8000000	1600000000	742000	148400000	3689,4088
Sumatorios	1450	19940	218500	34075000	5473330000	3114100	500823000	19936,818
Varianza		874710						868493,111

este caso, el sistema resultante sería:

*11*a0 + 1450*a1 + 218500*a2 = 19940*
*1450*a0 + 218500*a1 + 34075000*a2 = 3114100*
*218500*a0 + 34075000*a21+ 5473330000*a2 = 500823000*

***(Nota:** El coeficiente de a0 en la primera ecuación es 11 porque la muestra tiene 11 elementos.)*

De donde a0 = 0.9288, a1 = -0.6176, a2 = 0.0953.
De esta forma, la ecuación de la curva para obtener los tiempos estimados es:

$$t(n) = 0.9288 - 0.6176 * n + 0.0953 * n^2.$$

Esta función permitiría estimar el tiempo para cualquier valor de n.

Los tiempos estimados mediante la curva son los que están en la última columna y se han obtenido usando la ecuación de la curva.
*Por último, el **coeficiente de determinación** sería la varianza de los tiempos estimados dividida entre la varianza de los tiempos reales = 868493.111/874710 = 0.992, el cual es un valor bastante bueno (99.2%)*

Ejemplo práctico 2:
En este ejemplo se va a ajustar una curva del tipo nlog(n) a una curva que representa tiempos frente a valores de n. Este ejemplo podría ser el de la ordenación de un vector por el quicksort.
En este caso, la curva sería

$$t(n) = a0 + a1*n*log(n).$$

Para obtener el ajuste, se puede realizar el cambio de variable z = nlog(n) y tendríamos una situación similar a cuando se ajusta un polinomio de grado 1(ecuación de una recta).
Por tanto, la curva de ajuste quedaría como

$$t(n) = a0 + a1 * z.$$

Veamos como quedaría la tabla de las observaciones.

	n	t(tiempo real)	z=nlog(n)	z^2	z*t	t(estimado)
	100,000	50,500	200,000	40000,000	10100,000	49,813
	110,000	58,411	224,553	50424,138	13116,296	55,288
	120,000	58,400	249,502	62251,123	14570,989	60,852
	130,000	66,463	274,813	75521,985	18264,742	66,496
	140,000	67,592	300,458	90274,965	20308,427	72,215
	150,000	80,783	326,414	106545,896	26368,591	78,003
	160,000	83,032	352,659	124368,509	29281,942	83,856
	170,000	89,335	379,176	143774,679	33873,816	89,769
	180,000	93,690	405,949	164794,632	38033,290	95,740
	190,000	102,093	432,963	187457,119	44202,353	101,764
	200,000	109,541	460,206	211789,562	50411,517	107,839
Sumatorios	1450,000	859,839	3606,693	1257202,607	298531,964	861,636
Varianza		374,830				371,157

El sistema resultante sería

*11*a0 + 3606,693*a1 = 859.84*
3606,693 a0 + 1257202,607*a1 = 298531,964*

***(Nota:** El coeficiente de a0 en la primera ecuación es 11 porque la muestra tiene 11 elementos.)*

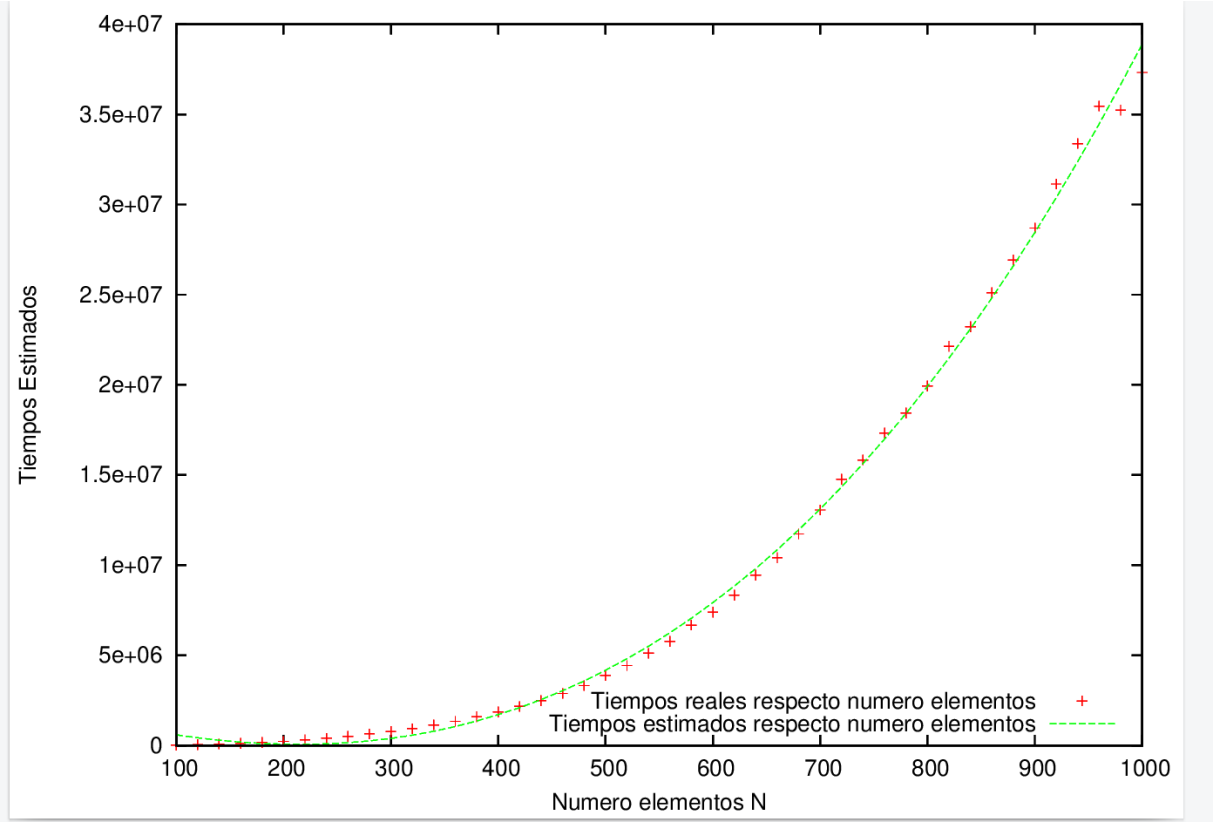
De donde a0 = 5.213 y a1 = 0.223. La ecuación de la curva para obtener los tiempos estimados sería:

$$t(n) = 5.213 + 0.223*n*log(n).$$

Esta función permitiría estimar el tiempo para cualquier valor de n.
Los tiempos estimados mediante la curva son los que están en la última columna y se han obtenido usando la ecuación de la curva.
*Por último, **el coeficiente de determinación** sería la varianza de los tiempos estimados dividida entre la varianza de los tiempos reales = 371,157/374,830= 0.99, el cual es un valor bastante bueno (99.0%).*

Finalmente se muestra una gráfica en la que se ha ajustado un polinomio de grado 3 para el algoritmo del producto de matrices. Las cruces en rojo son los puntos correspondientes a los

valores de n y de los tiempos reales, la curva en verde es la curva obtenida. Como se puede apreciar el ajuste es bastante bueno y con dicha curva se podría estimar por ejemplo que tiempo emplearía el algoritmo del producto de matrices para multiplicar matrices de orden 10000.



Fecha de comienzo: 15 de septiembre de 2025
Fecha de Entrega: 13 de Octubre de 2025