Práctica 0c: Análisis de rendimiento y complejidad
Estructuras de Datos – 1º GII/GIS

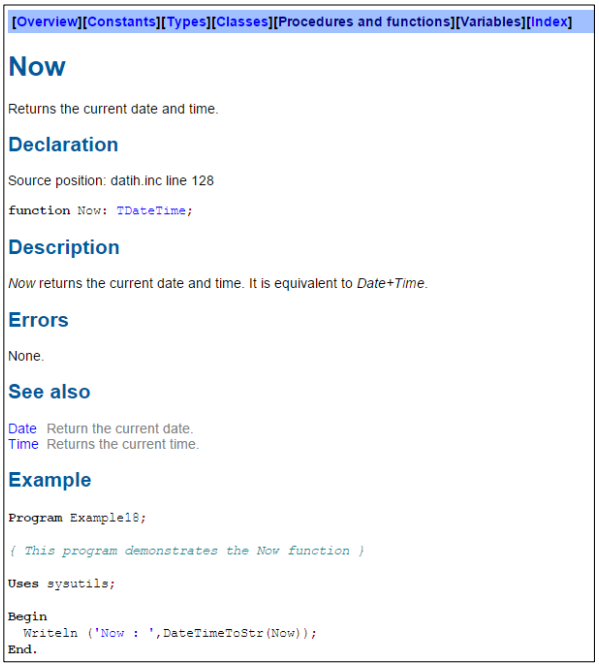
En esta práctica se desea evaluar el coste computacional de diversos algoritmos midiendo tiempos de ejecución para diferentes tamaños de problema.

Para ello será necesario hacer uso de la unidad de Pascal `Sysutils`. Las unidades en Pascal se pueden utilizar para importar a un proyecto tipos y subprogramas previamente creados y compilados[1]. Para utilizar una unidad en un programa principal (u otra unidad) basta invocar "`USES NombreUnidad;`" en la cabecera del módulo, de manera que para utilizar las definiciones de `Sysutils` sería necesario escribir algo como:

```
PROGRAM Complejidad;
USES Sysutils;
```

Una vez importada la unidad tendremos disponibles una variedad muy interesante de subprogramas que tienen que ver con funcionalidad del sistema (ficheros, cadenas de caracteres, funcionalidad horaria, etc.[2]). Para el propósito de esta práctica se utilizará la parte específica de la unidad dedicada a funcionalidad de manejo de variables temporales y fechas: http://www.freepascal.org/docs-html/rtl/sysutils/datetimeroutines.html

En particular, se puede observar el prototipo de la función `Now` y la función `DateTimeToTimeStamp`:



[Overview][Constants][Types][Classes][Procedures and functions][Variables][Index]

Now

Returns the current date and time.

Declaration

Source position: datih.inc line 128

```
function Now: TDateTime;
```

Description

Now returns the current date and time. It is equivalent to *Date+Time*.

Errors

None.

See also

Date Return the current date.
Time Returns the current time.

Example

```
Program Example18;

{ This program demonstrates the Now function }

Uses sysutils;

Begin
  Writeln ('Now : ',DateTimeToStr(Now));
End.
```

[Overview][Constants][Types][Classes][Procedures and functions][Variables][Index]

DateTimeToTimeStamp

Converts a *TDateTime* value to a TimeStamp structure.

Declaration

Source position: datih.inc line 111

```
function DateTimeToTimeStamp(
  DateTime: TDateTime
):TTimeStamp;
```

Description

DateTimeToSystemTime converts a date/time pair in *DateTime*, with *TDateTime* format to a *TTimeStamp* format.

Errors

None.

See also

DateTimeToFileDate Convert a *TDateTime* value to a file age (integer)
SystemTimeToDateTime Convert a system time to a *TDateTime* value.
DateTimeToSystemTime Converts a *TDateTime* value to a systemtime structure.

Example

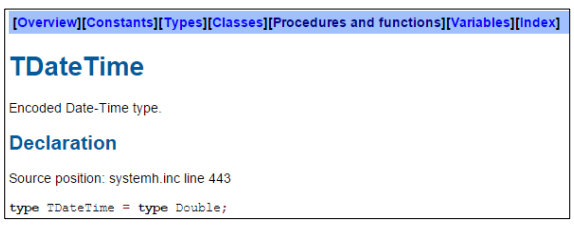
```
Program Example6;

{ This program demonstrates the DateTimeToTimeStamp function }

Uses sysutils;

Var TS : TTimeStamp;

Begin
  TS:=DateTimeToTimeStamp (Now);
  With TS do
    begin
    Writeln ('Now is ',time,' millisecond past midnight');
    Writeln ('Today is ' ,Date,' days past 1/1/0001');
    end;
End.
```



[Overview][Constants][Types][Classes][Procedures and functions][Variables][Index]

TDateTime

Encoded Date-Time type.

Declaration

Source position: systemh.inc line 443

```
type TDateTime = type Double;
```

[Overview][Constants][Types][Classes][Procedures and functions][Variables][Index]

TTimeStamp

TimeStamp structure

Declaration

Source position: datih.inc line 106

```
type TTimeStamp = record
  Time: LongInt;        Time part
  Date: LongInt;        Date part
end;
```

Description

TTimeStamp contains a timestamp, with the date and time parts specified as separate *TDateTime* values.

[1] La creación de unidades será motivo de las próximas prácticas.
[2] Más información en: http://www.freepascal.org/docs-html/rtl/sysutils/index-5.html

Como se puede comprobar, la llamada a la función `Now` no recibe argumentos y devuelve un valor de tipo `TDateTime` que representa la fecha y hora en un valor numérico de doble precisión. Este tipo puede ser descompuesto en `TTimeStamp`, más manejable, a través de la función `DateTimeToTimeStamp`. El tipo `TTimeStamp` es un registro con campos `Date` y `Time` como fecha y hora del sistema, y representados por sendos `LongInt`.

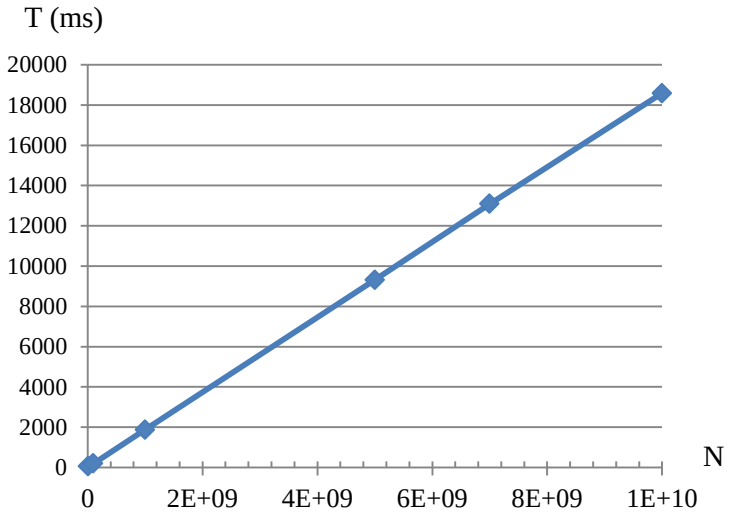
Si la fecha no nos interesa porque queremos cronometrar simplemente porciones de código en ejecución nos bastaría la parte de tipo `Time`, que como a cualquier registro se accede a través de la notación "." como `registro.time`.

Por ejemplo considérese el siguiente código:

```
PROGRAM Complejidad;
USES Sysutils;
CONST
      N = 1000000;
VAR
      ts, ts2: TTimeStamp;
      i, x: longint;
BEGIN
      ts := DateTimeToTimeStamp(Now);
      x := 0;
      FOR i := 1 TO N DO
            x := x + 1;
      ts2 := DateTimeToTimeStamp (Now);
      writeln('tiempo: ',ts2.time - ts.time, ' ms');
END.
```

A partir de aquí ya es posible evaluar el coste computacional de diferentes algoritmos para diferentes tamaños de problema. Los tiempos que devuelvan serán posteriormente representados en una gráfica N-T (tamaño del problema, N, en abscisas y tiempo, T, en ordenadas). Por ejemplo para unas ejecuciones con diversos valores de la constante podemos obtener los siguientes resultados[3]:

N	Tiempo (ms)
10.000.000	55
100.000.000	191
1000.000.000	1871
5.000.000.000	9312
7.000.000.000	13084
1E+10	18584



Se pide hacer en Pascal un cuerpo de programa principal que utilice un tipo Array definido por el usuario del siguiente modo:

[3] Para este ejemplo se ha hecho uso de un doble bucle porque la variable iteradora sólo admite valores del rango `longint` (32 bits, hasta 2^{31}-1 =~ 2147 millones). Existe el tipo `Cardinal` para aumentar a unos 4000 millones.

```
const
      N = 1048576; {2^20}
type
      TAlmacen = array [1..N] of integer;
var
      coleccion: TAlmacen;
```

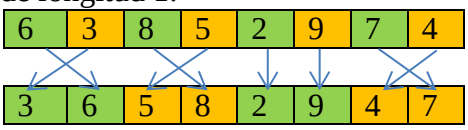
A continuación codifique 4 subprogramas:

1) Búsqueda secuencial (O(n)): itera a lo largo del array hasta encontrar el elemento que se busca. En el peor de los casos nunca lo encuentra:

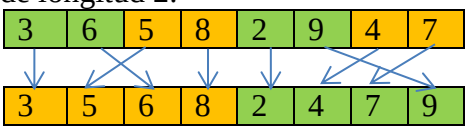
```
function Busqueda_secuencial(var c: TAlmacen): boolean;
```

2) Busqueda binaria (O(log n)): en un array previamente ordenado busca un elemento comenzando por la posición central del array y evaluando si se ha encontrado, si no, sigue buscando en la mitad restante que interese (mitad mayor o mitad menor). En el peor de los casos nunca lo encuentra.
3) Ordenación directa (O(n^2)): se realiza mediante 2 bucles (ordenación por intercambio directo, inserción directa, etc.), por ejemplo haciendo N pasadas de intercambios entre elementos vecinos (siendo N el tamaño del array).
4) Ordenación por mezcla (O(n log n)): algoritmo multipasada. En la primera pasada se ordena un elemento con su respectivo vecino (longitud 1) "recorriendo ambas posiciones por separado" (elementos en posiciones i e i+1, siendo i mod 2 = 0, ordenando 2 elementos entre sí), en la segunda pasada el array está ordenado por parejas y se ordenan dobles parejas mediante 2 recorridos de 2 elementos (longitud 2, ordena 4 elementos entre sí), en la siguiente pasada igual con 2 recorridos sobre 4 elementos (longitud 4, ordenando 8 elementos entre sí), y así sucesivamente hasta terminar con 2 recorridos de cada mitad de elementos del array (longitud N/2) para ordenar todos los valores. Ejemplo:

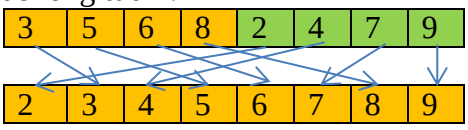
Recorridos de longitud 1:



Recorridos de longitud 2:



Recorridos de longitud 4:



A continuación mide el tiempo (T) invertido con cada subprograma probando con diferentes tamaños del array (N) definido anteriormente y haz una representación gráfica de cada comportamiento.