EJERCICIOS DE COMPLEJIDAD ALGORÍTMICA

1. Calcula la complejidad del siguiente procedimiento:

```
PROCEDIMIENTO intercambia(REF x : ENTERO, REF y : ENTERO) ES
   aux : ENTERO;
   aux := x;
   x := y;
   y := aux;
FINPROCEDIMIENTO
```

2. Calcular la complejidad de la siguiente función:

```
FUNCION max(x : ENTERO, y : ENTERO) : ENTERO ES

   result : ENTERO;
   SI (x >= y) ENTONCES
       result := x;
   SINO
       result := y;
   FINSI
   DEVOLVER result;
FINFUNCION
```

3. Calcular la complejidad de la siguiente función:

```
FUNCION suma (v : VECTOR(ENTERO), tamaño : ENTERO)  : ENTERO ES
   i, result : ENTERO; result := 0;
   PARA i DESDE 0 HASTA tamaño-1 HACER
       result := result + v[i];
   FINPARA
   DEVOLVER result;
FINFUNCION
```

4. Calcular la complejidad de la siguiente función

```
FUNCION aSaber(v : VECTOR(ENTERO),  n: ENTERO)  : ENTERO ES
   i, result : ENTERO;
   result := 0;     i := 0;
   MIENTRAS (i < tamaño) HACER
      result := result + v[i];
      i := i + 1;
   FINMIENTRAS
   DEVOLVER result;
FINFUNCION
```

5. Calcular la complejidad de la siguiente función

```
FUNCION aSaber(v : VECTOR(ENTERO),  n: ENTERO)  : ENTERO ES
   i, result : ENTERO;
   result := 0;     i := tamaño - 1;
   MIENTRAS (i >= 0) HACER
      result := result + v[i];
      i := i - 1;
   FINMIENTRAS
   DEVOLVER result;
FINFUNCION
```

6. Calcular la complejidad de la siguiente función

```
FUNCION aSaber(v : VECTOR(ENTERO), n: ENTERO)  : ENTERO ES
   i, result : ENTERO;
   result := 0; i := tamaño - 1;
   MIENTRAS (i >= 0) HACER
      result := result + v[i];
      i := i - 2;
   FINMIENTRAS
   DEVOLVER result;
FINFUNCION
```

7. Calcular la complejidad de la siguiente función

```
FUNCION aSaber(v : VECTOR(ENTERO), n: ENTERO)  : ENTERO ES
   i, result : ENTERO;
   result := 0; i :=0;
   MIENTRAS ((i < tamaño) && ((v[i] DIV 2) != 0) HACER
      result := result + v[i];
      i := i + 1;
   FINMIENTRAS
   DEVOLVER result;
FINFUNCION
```

8. Calcular la complejidad de la siguiente función

```
FUNCION aSaber(v : VECTOR(ENTERO), n: ENTERO)  : ENTERO ES
   i, result : ENTERO;
   result := 0; i :=0;
   MIENTRAS (i < tamaño-1) HACER
      SI (v[i] <= v[i+1]) ENTONCES
         intercambia(v[i],v[i+1]);
      FINSI
   FINMIENTRAS
   DEVOLVER result;
FINFUNCION
```

NOTA: intercambia es la función del ejercicio 1

9. Calcular la complejidad del siguiente procedimiento

```
PROCEDIMIENTO burbuja(v : VECTOR(ENTERO), tamaño : NATURAL) ES
   i, j : ENTERO;
   PARA i DESDE 2 HASTA tamaño - 1 HACER
      PARA j DESDE 0 HASTA tamaño – i HACER
         SI (v[j] > v[j+1]) ENTONCES
            intercambia(v[j],v[j+1]);
         FINSI
      FINPARA
   FINPARA
FINPROCEDIMIENTO
```

10. Calcular la complejidad del siguiente procedimiento.

```
PROCEDIMIENTO inserción(v : VECTOR(ENTERO), tamaño : NATURAL) ES
   i, j : ENTERO;
   PARA i DESDE 0 HASTA tamaño - 2 HACER
      min : ENTERO; min := i;
      PARA j DESDE i+1 HASTA tamaño – 1 HACER
         SI (v[j] < v[min]) ENTONCES
            min := j;
         FINSI
      FINPARA
      intercambia(v[i],v[min])
   FINPARA
FINPROCEDIMIENTO
```

11. Calcular la complejidad del siguiente procedimiento.

```
PROCEDIMIENTO inserción(v : VECTOR(ENTERO), tamaño : NATURAL) ES
   i, j : ENTERO;
   PARA i DESDE 0 HASTA tamaño - 2 HACER
      intercambia(v[i],v[posMinimo(v,i,tamaño-1)])
   FINPARA
FINPROCEDIMIENTO

FUNCION posMinimo(v : VECTOR(ENTERO), inicio, fin : NATURAL) ES
   i, result: ENTERO;
   result := inicio;
   PARA i DESDE inicio+1 HASTA fin HACER
      SI (v[i] < v[min]) ENTONCES
         result := i;
      FINSI
    FINPARA
    DEVOLVER result;
FINFUNCION
```

12. Plantear la ecuación de recurrencia para el siguiente algoritmo recursivo

```
FUNCION factorial (n : NATURAL) : NATURAL  ES
   result : NATURAL;
   SI ((n == 0) OR (n == 1)) ENTONCES
      result := 1;
   SINO
      result := n*factorial(n-1);
   FINSI
   DEVOLVER result;
FINFUNCION
```

13. Plantear la ecuación de recurrencia para el siguiente algoritmo recursivo

```
FUNCION fibonacci (n : NATURAL) : NATURAL  ES
   result : NATURAL;
   SI ((n == 0) ENTONCES
      result := 0;
   SINOSI (n==1) ENTONCES
      result := 1;
   SINO
      result := fibonacci(n-1)+fibonacci(n-2);
   FINSI
   DEVOLVER result;
FINFUNCION
```

14. Calcular la complejidad del siguiente algoritmo recursivo tipo divide y vencerás

```
PROCEDIMIENTO mezcla (REF v : VECTOR(ENTERO), tamaño : NATURAL) ES
    mezclaRecursivo(v : VECTOR(ENTERO), 0,tamaño-1);
FINPROCEDIMIENTO

PROCEDIMIENTO mezclaRecursivo (REF v:VECTOR(ENTERO), inicio, fin : NATURAL) ES
    SI (inicio < fin) ENTONCES
       mezclaRecursivo(v,inicio, (inicio + fin) div 2);
       mezclaRecursivo(v,((inicio + fin) div 2)+1,fin);
       fusiona(v,inicio,(inicio + fin) div 2),fin)
    FINSI
FINFUNCION

PROCEDIMIENTO fusiona(REF v : VECTOR(ENTERO), inicio, medio, fin : NATURAL) ES
   aux : VECTOR DIM (fin – inicio + 1) DE ENTERO;
   i1,i2, pos : ENTERO; i1 := inicio; i2 := medio+1; pos := 0;
   MIENTRAS ((i1 <= medio) AND (i2 <= fin)) HACER
     SI (v[i1] <= v[i2]) ENTONCES
         aux[pos] := v[i1];
         i1 := i1 + 1;
     SINO
         aux[pos] := v[i2];
         i2 := i2 + 1;
     FINSI
     pos := pos + 1;
   FINMIENTRAS
```

```
   inicioResto, finResto, i : NATURAL;
   SI (i1 > medio) ENTONCES
       inicioResto := i2; finResto := fin;
   SINO
        inicioResto := i1; finResto := medio;
   FINSI
   PARA i DESDE inicioResto HASTA finResto DESDE
       aux[pos] := v[i];
        pos := pos + 1;
   FINPARA
   PARA i DESDE inicio HASTA fin HACER
      v[i] := aux[pos];
   FINPARA
FINPROCEDIMIENTO
```

15. Calcular la complejidad del siguiente algoritmo recursivo tipo divide y vencerás

```
FUNCION esta(v : VECTOR(NATURAL), e : NATURAL,
             inicio, fin : NATURAL) : BOOLEANO ES
   result : BOOLEAN; result := FALSO;
   SI  (inicio == fin) ENTONCES
     result := (v[inicio] == e);
   SINOSI (v[(inicio + fin) div 2] == e) ENTONCES
     result := VERDADERO;
   SINO
     result := esta(v,e,inicio,((inicio + fin) div 2)- 1) OR
               esta(v,e,((inicio + fin) div 2)+ 1,fin);
   FINSI
   DEVOLVER result;
FINFUNCION
```

16. Calcular la complejidad del siguiente algoritmo recursivo tipo divide y vencerás

```
FUNCION esta(v : VECTOR(NATURAL), e : NATURAL,
             inicio, fin : NATURAL) : BOOLEANO ES
   result : BOOLEAN; result := FALSO;
   SI  (inicio == fin) ENTONCES
     result := (v[inicio] == e);
   SINOSI (v[(inicio + fin) div 2] == e) ENTONCES
     result := VERDADERO;
   SINOSI (v[(inicio + fin) div 2] < e) ENTONCES
     result := esta(v,e,inicio,((inicio + fin) div 2)- 1)
   SINO
     Result := esta(v,e,((inicio + fin) div 2)+ 1,fin);
   FINSI
   DEVOLVER result;
FINFUNCION
```

17. Calcula la complejidad del siguiente algoritmo recursivo tipo divide y vencerás, asumiendo que las siguientes funciones poseen las complejidades indicadas:

Función	Complejidad
tamaño(x : ENTERO_LARGO) : ENTERO	O(1)
aEnteroLargo(x : ENTERO) : ENTERO_LARGO	O(x)
aEntero(x : ENTERO_LARGO) : ENTERO	O(tamaño(x))
mayor(x, y : ENTERO) : ENTERO	O(1)
extraeSuperior(x:ENTERO_LARGO, e:ENTERO) : ENTERO_LARGO	O(n)
extraeInferior(x:ENTERO_LARGO, e:ENTERO) : ENTERO_LARGO	O(n)
suma(x, y : ENTERO_LARGO) : ENTERO_LARGO	O(max(tamaño(x), tamaño(y))
desplaza(x : ENTERO_LARGO, e : ENTERO) : ENTERO_LARGO	O(tamaño(x))

```
PROCEDIMIENTO multiplicarNumerosLargos(x, y : ENTERO_LARGO) : ENTERO_LARGO ES
       result : ENTERO;
       SI (tamaño(x) <= UMBRAL_RAZONABLE)) AND
          (tamaño(y) <= UMBRAL_RAZONABLE)) ES
          result := aEnteroLargo(aEntero(x)*aEntero(y));
       SINO
          exponente : ENTERO;
          exponente := mayor(tamaño(x),tamaño(y));
          x1, x2, y1, y2 : ENTERO_LARGO;
          x1 := extraeSuperior(x,exponente div 2);
          x2 := extraeInferior(x,exponente div 2);
          y1 := extraeSuperior(x,exponente div 2);
          y2 := extraeInferior(x,exponente div 2);
          result := suma(suma(multiplica(x2,y2),
                         desplaza(suma(multiplica(x1,y2),
                                       multiplica(x2,y1)),
                                  exponente div 2)),
                         desplaza(multiplica(x1,y1),exponente));
       FINSI
       DEVOLVER result;
FINPROCEDIMIENTO
```

El algoritmo de multiplicación de números largos se basa en la siguiente propiedad. Dados dos números largos a y b, podemos expresarlos de la siguiente forma:

$$a = a'*10^k + a''$$
$$b = b'*10^k + b''$$

Por tanto, la multiplicación la podemos reescribir de la siguiente forma:

$$a*b = (a'*10^k + a'')*(b'*10^k + b'') = a'*b'*10^{2k}+(a'*b''+a''*b')*10^k+a''*b''$$

De esta forma conseguimos reducir el problema de multiplicar dos números grandes al problema de multiplicar números más pequeños para a continuación sumarlos y desplazarlos de acuerdo a un exponente.

18. Paco y Lola elaboran sendos algoritmos para calcular el número de Champions League que ganará el Racing de Santander en las 4 próximas décadas en función de la edad a la cual se jubile D. Pedro Munitis. Tanto Paco como Lola ejecutan su algoritmo asumiendo que D. Pedro se retirará de forma gloriosa tras ganar el campeonato de liga con el Racing, a la edad de 47 años. El algoritmo de Paco tarda 2 minutos 30 segundos en ejecutarse, mientras que el de Lola tarda apenas 40 segundos. ¿Significa esto que el algoritmo de Lola es más eficiente?

19. Paco y Lola elaboran dos algoritmos para calcular el número de Champions League que ganará la Gimnástica de Torrelavega en las próximas 4 décadas en función de la edad a la cual se jubile D. Pedro Munitis. Paco y Lola calculan las complejidades de sus algoritmos. La complejidad de ambos algoritmos es O(n). ¿Significa esto que los dos algoritmos tardan el mismo tiempo en ejecutarse?

20. ¿Cómo se calcularía la complejidad para una sentencia tipo *switch*?

Pablo Sánchez Barreiro.