Ejercicio_1.

Usar las relaciones $\subset$ y **=** para ordenar los órdenes de complejidad, **O**, **Ω**, y **Θ**, de las siguientes funciones:

$n\log n$, $n^2\log n$, n^8, n^{1+a}, $(1+a)^n$, $(n^2+8n+\log^3 n)^4$, $n^2/\log n$, 2^n,

siendo **a** una constante real, $0 < \mathbf{a} < 1$.

Ejercicio_2.

Usando la definición de notación asintótica Θ demostrar que $512n^2 + 5n \in \Theta(n^2)$.

Ejercicio_3.

Usando las definiciones de notación asintótica, demostrar si son verdaderas o falsas las afirmaciones siguientes:

a. $(n+1)! \in O\,(3(n!))$
b. $n^2 \in \Omega\,((n+1)^2)$

Ejercicio_4.

Escribir un algoritmo que, dado un entero positivo $n \geq 1$, verifique si es un número triangular. Analizar el algoritmo implementado.

NOTAS:

- Un número natural $n \geq 1$ es *triangular* si es la suma de una sucesión ascendente no nula de naturales consecutivos que comienza en 1. Por tanto, los cinco primeros números triangulares son:

  1, 3 = 1+2, 6 = 1+2+3, 10 = 1+2+3+4 y 15 = 1 + 2 + 3 + 4 + 5.

- Un posible algoritmo para comprobar que un número natural positivo n *es* triangular consiste en calcular sucesivamente los números triangulares y compararlos con n. En cada iteración, se genera el siguiente número triangular, si es igual a n se termina con éxito; en caso contrario, el número generado será mayor que n y se termina con fracaso.

Ejercicio_5.

Dado el algoritmo siguiente, que determina si una cadena C es palíndroma:

```
función PAL (C, i, j) : booleano;
  if i ≥ j then
      return cierto
  else
      if C(i)≠C(j) then
             return falso
      else
             return PAL(C, i+1, j-1)
  ffunción
```

➢ **Calcular** el tiempo de ejecución para PAL(C, 1, n) en el caso peor y en el caso medio, suponiendo equiprobabilidad de todas las entradas y siendo {a, b} el alfabeto que forma las cadenas

NOTA:

- Para calcular la eficiencia temporal considerar como operación característica el número de comparaciones entre componentes de la cadena (C(i)≠C(j)), siendo n=j-i+1 el tamaño de la cadena.

Ejercicio_6.

Para resolver cierto problema se dispone de dos algoritmos, A_1 y A_2, de divide y vencerás:

- A_1 descompone el problema de tamaño **n** en tres subproblemas de tamaño **n/2** y cuatro subproblemas de tamaño **n/4**. La división y combinación requieren $3n^2$, y el caso base, con **n** menor que 5, es **n!**
- A_2 descompone el problema de tamaño **n** en un subproblema de tamaño **n-3** y dos de tamaño **n-2**. El tiempo de la división y combinación es despreciable, y el caso base, con **n** menor que 5, es de orden constante.
  1. Calcular el orden de complejidad de los dos algoritmos.
  2. Estudiar cuál de los dos algoritmos es más eficiente.

Ejercicio_7. Algoritmo Recursivo para la Búsqueda Ternaria.

El algoritmo de "búsqueda ternaria" realiza una búsqueda de un elemento en un vector ordenado.
La función compara el elemento a buscar "clave" con el que ocupa la posición n/3 y si este es menor que el elemento a buscar se vuelve a comparar con el que ocupa la posición 2n/3. En caso de no coincidir ninguno con el elemento buscado se busca recursivamente en el subvector correspondiente de tamaño 1/3 del original.

1. Escribir un algoritmo para la búsqueda ternaria.
2. Calcular la complejidad del algoritmo propuesto.
3. Comparar el algoritmo propuesto con el de búsqueda binaria.

Ejercicio_8.

Escribir un algoritmo que dados un vector de **n** enteros y un entero **X**, determine si existen en el vector dos números cuya suma sea X.
El tiempo del algoritmo debe ser de O(n*lgn). Analiza el algoritmo y demuestra que es así.

Ejercicio_9.

Estudiar la complejidad del algoritmo de ordenación por Selección por la llamada al procedimiento, especificado a continuación, `Selection (a,1,n)`.

- El procedimiento Selección puede ser implementado como sigue:

```
procedimiento  Selection (a:vector; primero,ultimo: int);
    para i=primero hasta ultimo-1 hacer
            posmin = PosMinimo(a,i,ultimo);
            Intercambia(a, i, posmin);
    fpara
fprocedimiento Selection
```

- En el algoritmo anterior se utiliza una función **`PosMinimo`** que calcula la posición del elemento mínimo de un subvector :

```
Int función PosMinimo (a:vector;primero,ultimo:int);
/* devuelve la posición del mínimo elemento de a[primero..ultimo] */
    pmin=primero;
    para i=primero+1 hasta ultimo hacer
            si a[i] < a[pmin] entonces
                    pmin = i
            fsi;
    fpara
    return pmin;
ffuncion PosMinimo;
```

- También se utiliza el procedimiento **`Intercambia`** para intercambiar dos elementos de un vector:

```
función Intercambia (a:vector ; i , j :int );
```

```
/* intercambia a[i]  con a[j]  */
   aux = a[i] ;
   a[i] = a[j] ;
   a[j] = aux;
ffuncion Intercambia;
```

Ejercicio_10.

Para resolver cierto problema se dispone de un algoritmo trivial cuyo tiempo de ejecución t(n) (para problemas de tamaño n) es cuadrático ($t(n) \in \Theta(n^2)$). Se ha encontrado una estrategia Divide y Vencerás para resolver el mismo problema; dicha estrategia realiza ***D(n)= n log n*** operaciones para dividir el problema en dos subproblemas de tamaño mitad y ***C(n)= n log n*** operaciones para componer una solución del original con la solución de dichos subproblemas.

1. Calcular la eficiencia para el algoritmo Divide y Vencerás por el método de la **ecuación característica**
2. Corroborar el resultado anterior aplicando el teorema maestro.
3. Estudiar cuál de los dos algoritmos es más eficiente.

NOTA:

Teorema : La solución a la ecuación $T(n) = aT(n/b) + \Theta(n^k \log^p n)$, con $a \geq 1$, $b > 1$ y $p \geq 0$, es:

$$T(n) = \begin{cases} O(n^{\log_b a}) & \text{si } a > b^k \\ O(n^k \log^{p+1} n) & \text{si } a = b^k \\ O(n^k \log^p n) & \text{si } a < b^k \end{cases}$$

Ejercicio_11.

Se realiza una variante de los números de Fibonacci que denominaremos "**Nacci**" cuya ecuación recurrente es:

$$\textbf{Nacci}\,(n) = \begin{cases} 1 & \text{Si } n = 1 \\ 3 & \text{Si } n = 2 \\ 3/2\ \textbf{Nacci}\,(n-1) + \textbf{Nacci}\,(n-2) & \text{En otro caso} \end{cases}$$

1. Escribir tres posibles implementaciones, **simples y cortas**, para el cálculo del n-ésimo número de **f** con las siguientes estrategias:
   a. divide y vencerás recursivo.
   b. Procedimiento iterativo.
   c. procedimiento directo, mediante una simple operación aritmética.
2. Realizar una estimación del orden de complejidad de los tres algoritmos del apartado anterior. Comparar los órdenes de complejidad obtenidos, estableciendo una relación de orden entre los mismos.

Ejercicio_12.

Escribir un algoritmo voraz para entregar billetes en un cajero automático que suministra la cantidad de billetes solicitada de forma que el número total de billetes sea **mínimo**. Se supone que el cajero dispone de suficientes billetes de todas las cantidades consideradas. Explicar el funcionamiento del algoritmo: cuál es el conjunto de candidatos, la función de selección, la función para añadir un elemento a la solución, el criterio de finalización, el criterio de coste, etc. Suponer billetes de 10, 20 y 50 €.
Aplicar el algoritmo para el caso que se solicite la cantidad de **570 €.**

Ejercicio_13. (Ex_Sept 16)

Resolver las siguientes ecuaciones de recurrencia y calcular el orden temporal (1 punto cada apartado):

1. **(1 punto)**

```
function total(n:positivo)
        if n=1 then 1 else total(n-1) + 2 * parcial(n-1)
```

- siendo

```
function parcial (m:positivo)
        if m=1 then 1 else 2 * parcial(m-1)
```

2. **(1 punto)**

```
function total(n,m:positivo)
        if n=1 then m else    m + total (n-1, 2 * m)
```

3. **(1 punto)**

$$T(n) = \begin{cases} a & \text{si } n=1 \\ 2T\left(\frac{n}{4}\right) + \lg n & \text{si } n>1 \end{cases}$$

Ejercicio_14. (Ex_Sept 16)

Tenemos que ejecutar un conjunto de n tareas, cada una de las cuales requiere un tiempo unitario. En un instante T=1, 2, ... podemos ejecutar únicamente una tarea. La tarea i produce unos beneficios b_i ($b_i > 0$) sólo en el caso en el que sea ejecutada en un instante anterior o igual a d_i.

- Diseñar un algoritmo **voraz** para resolver el problema aunque no se garantice la solución óptima que nos permita seleccionar el conjunto de tareas a realizar de forma que nos aseguremos que tenemos el **mayor beneficio posible**.
- Detallar :

a. (1,5 puntos) Las estructuras y/o variables necesarias para representar la información del problema y el método voraz utilizado (El procedimiento o función que implemente el algoritmo). Es necesario marcar en el seudocódigo propuesto a que corresponde cada parte en el esquema general de un algoritmo voraz. Si hay más de un criterio posible elegir uno razonadamente y discutir los otros. Indicar razonadamente el orden de dicho algoritmo.

b. (0,5 puntos) Aplicar el algoritmo implementado en el apartado anterior a la siguiente instancia:

i	1	2	3	4
b_i	50	10	15	30
d_i	2	1	2	1