Ejercicio_1.

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

nlogn, n^2logn, n^8, n^{1+a}, $(1+a)^n$, $(n^2+8n+\log^3 n)^4$, n^2/ logn, 2^n, siendo a una constante real, 0<a<1.

$$O(1) \subset O(\log n) \subset O(N) \subset (N \log N) \subset O(N^2)$$
$$\subset O(N^3) \subset O(2^n) \subset O(N!).$$

$$\lim_{n\to\infty} \frac{(n^2+8n+\log^3 n)^4}{n^8} = \frac{n^8+8n^4+\log^{12} n}{n^8} = \boxed{\frac{\infty}{\infty}}$$

$$\boxed{\Theta(n^2+8n+\log^3 n)^4 = \Theta(n^8)}$$

$$\lim_{n\to\infty} \frac{n \log n}{n^{1+a}} = \frac{n \log n}{n^1 \cdot n^{0'5}} \overset{L'hopital}{=} \frac{1/n}{0'5 \cdot n^{-0'5}} = \frac{0'5 \cdot n^{1/2}}{n} = \boxed{0}$$

$$\boxed{O(n^{1+a}) \supset O(n \log n)}$$

$$\frac{n^2}{n^5} = 0 \to O(n^5) \supset O(n^2)$$

$$\frac{n^5}{n^2} = \infty \to \Omega(n^5) \supset O(n^2)$$

$$\frac{n^5}{n^5} = 1 \to \Theta(n^5) = \Theta(n^5)$$

$$\boxed{\begin{array}{l} O(n \log n) \subset O(n^2/\log n) \subset O(n^2 \log n) \subset \Theta(n^8) \subset \\ \Theta(n^2+8n+\log^3 n)^4 \subset O(1+a)^n \subset O(2^n) \end{array}}$$

Solución.

Ejercicio_2.

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

$f(n) \leq CN^2 ; \forall N \geq N_0$ Para cota superior

$f(n) \geq C_2 N^2 ; \forall N \geq N_0$ Para cota inferior

$f(n) = 512n^2 + 5n$

$512n^2 + 5n \leq CN^2 \longrightarrow 512 + \frac{5}{n} \leq C \longrightarrow \begin{cases} N_0 = 5 \\ C = 513 \end{cases}$

Para valores $N_0 \geq 5$ siempre se cumple la desigualdad.

$512n^2 + 5n \in O(n^2)$

$512n^2 + 5n \geq C_2 n^2 \longrightarrow C_2 = 512 \longrightarrow 512n^2 + 5n \geq 512n^2 \longrightarrow \begin{cases} N_0 = 0 \\ C_2 = 512 \end{cases}$

Para valores $N_0 \geq 0$ siempre se cumple la desigualdad

$512n^2 + 5n \in \Omega(n^2)$

Como está acota superior e inferiormente $\longrightarrow 512n^2 + 5n \in \Theta(n^2)$

Ejercicio_3.

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

(a) $(n+1)! \in \mathbf{O}(3(n!))$

(b) $n^2 \in \mathbf{\Omega}((n+1)^2)$

A) $(n+1)! \in O(3(n!)) \longrightarrow (n+1)! \leq c(3n!)$

$(n+1) \cdot n! \leq c \cdot 3n! \longrightarrow n+1 \leq 3 \cdot c \longrightarrow c \geq \frac{n+1}{3}$

No existe ninguna C que cumple la desigualdad ya que n cambia constantemente.

La afirmación es falsa.

B) $n^2 \in \Omega((n+1)^2) \longrightarrow n^2 \geq c \cdot ((n+1)^2)$

$n^2 \geq c(n^2 + 2n + 1) \longrightarrow 1 \geq c \cdot \frac{n^2 + 2n + 1}{n^2} \longrightarrow$

$\longrightarrow 1 \geq c \left(1 + \frac{2}{n} + \frac{1}{n^2}\right) \longrightarrow n_0 = 1 \longrightarrow 1 \geq c \cdot 4 \longrightarrow \begin{cases} n_0 = 1 \\ C = 1/4 \end{cases}$

Para $n_0 \geq 1$ siempre se cumple la desigualdad por lo que la afirmación es verdadera.

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.

```
funcion triangular (n : entero): booleano
  inicio
    numtri ← 1  // 1
    ultimoSumando ← 1  // 1
    mientras numtri < n hacer  // k veces
      ultimoSumando ← ultimoSumando + 1  // 2
      numtri ← numtri + ultimoSumando  // 2
    fmientras
  devuelve (numtri = n)  // 1
ffuncion
```

$$n = \sum_{i=1}^{k} (1) = \frac{(1+k)\cdot k}{2} = \frac{k^2+k}{2} = n \rightarrow \boxed{k^2+k-2n=0}$$

$$\boxed{k = \frac{-1 \pm \sqrt{1+8n}}{2}} \rightarrow O(\sqrt{n})$$

nº veces que se ejecuta el bucle

por lo tanto triangular $(n) \in \Theta(\sqrt{n})$

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.

• Caso peor:

$$t(n) = \begin{cases} 0 & \text{Si } n \leq 1 \\ 1 + T(n-2) & \text{Si } n > 1 \end{cases}$$

$t(n) = 1 + T(n-2) \rightarrow t(n) - t(n-2) = 1^n \cdot n^0$ No homogenea

$(x^2 - 1)(x - 1) = 0 \rightarrow r_1 = 1$ doble, $r_2 = -1$

$t(n) = c_1 \cdot 1^n \cdot n^0 + c_2 \cdot 1^n \cdot n^1 + c_3 (-1)^n \cdot n^0$

$t(n) = c_1 + c_2 \cdot n + c_3 (-1)^n$

$$\boxed{c_1 = -\tfrac{1}{2} \quad c_2 = 2 \quad c_3 = \tfrac{1}{2}}$$

$$\begin{cases} t(0) = c_1 + c_3 \rightarrow \boxed{c_1 = -c_3} \\ t(1) = c_1 + c_2 - c_3 \rightarrow 1 = 2c_1 + c_2 \rightarrow \boxed{c_2 = 1 - 2c_1} \\ t(2) = c_1 + 2c_2 + c_3 \rightarrow c_1 + 1 - 2c_1 - c_1 = 2 \rightarrow -2c_1 = 1 \quad \boxed{c_1 = -\tfrac{1}{2}} \end{cases}$$

$t(n) = -\tfrac{1}{2} + 2n + \tfrac{1}{2}(-1)^n$

$t(n) \approx 2n \rightarrow \boxed{t(n) \in O(n)}$

• Caso medio

No homog.

$$T(n) \begin{cases} 0 & si\ n \leq 1 \\ \frac{1}{2} + \frac{1}{2}(1 + T(n-2)) & si\ n > 1 \end{cases}$$

$$T(n) = \frac{1}{2} + \frac{1}{2}(1 + T(n-2)) \rightarrow \boxed{T(n) - \frac{1}{2} \cdot T(n-2) = 1^n \cdot n^0}$$

$$(x^2 - \tfrac{1}{2})(x-1) = 0 \rightarrow (x - \sqrt{1/2})(x + \sqrt{1/2})(x-1) = 0$$

$$\begin{cases} r_1 = \sqrt{1/2} \\ r_2 = -\sqrt{1/2} \\ r_3 = 1 \end{cases}$$

$$T(n) = C_1 (\sqrt{1/2})^n n^0 + C_2 (-\sqrt{1/2})^n \cdot n^0 + C_3 (1)^n \cdot n^0$$

$$T(n) = C_1 \left(\frac{\sqrt{2}}{2}\right)^n + \left(\frac{-\sqrt{2}}{2}\right)^n C_2 + C_3$$

$$\boxed{T(n) \in O\left(\left(\frac{\sqrt{2}}{2}\right)^n\right) \in O(1)}$$

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.

$$T_{A_1}(n) = \begin{cases} n! & \text{si } n \leq 3 \\ 3T_1(n/2) + 4T_1(n/4) + 3n_2 & \text{si } n > 4 \end{cases}$$

$$T_{A_2}(n) = \begin{cases} c & n \leq 3 \\ 2T_2(n-2) + T_2(n-3) & \text{si } n > 4 \end{cases}$$

a) no homogenea

$$T(n) - 3T(n/2) - 4T(n/4) = 3n^2$$

$$\boxed{\begin{matrix} n = 2^k \\ k = \log_2 n \end{matrix}} \longrightarrow T(2^k) - 3T(2^{k-1}) - 4T(2^{k-2}) = 3 \cdot 2^{2k}$$

$$\boxed{T(2^k) = T_k} \longrightarrow T_k - 3T_{k-1} - 4T_{k-2} = \underbrace{3 \cdot 2^{2k}}_{b^k \cdot p(k) = 0} \begin{cases} b = 2^2 \\ d = 0 \end{cases}$$

$$(x^2 - 3x - 4)(x - 4) = 0 \begin{cases} r_1 = -1 \\ r_2 = 4 \text{ doble} \end{cases}$$

$$T_k = c_1 \cdot (-1)^k \cdot \cancel{k^0} + c_2 \cdot 4^k \cdot \cancel{k^0} + c_3 \cdot 4^k \cdot k^1$$

$$\underline{T_k = c_1 \cdot (-1)^k + c_2 \cdot 4^k + c_3 \cdot 4^k \cdot k}$$

$$\boxed{\begin{matrix} t_k = t(2^k) \\ k = \log n \\ n = 2^n \end{matrix}} \longrightarrow \begin{matrix} t(n) = c_1 (-1)^{\log_2 n} + c_2 4^{\log_2 n} + c_3 \cdot 4^{\log_2 n} \cdot \log_2 n \\ t(n) = c_1 (-n^{0'5}) + c_2 \cdot n^2 + c_3 \cdot n^2 \log_2 n \\ \boxed{T(n) = -c_1 \sqrt{n} + c_2 n^2 + c_3 n^2 \log_2 n} \end{matrix}$$

$$T(n) \in O(n^2 \log_2 n)$$

b) Homogénea

$T(n) - 2T(n-2) - T(n-3) = 0 \longrightarrow \underset{x^3}{T_n} - 2\underset{x}{T_{n-2}} - \underset{1}{T_{n-3}} = 0$

$x^3 - 2x - 1 = 0$

$$\begin{array}{r|rrrr} & 1 & 0 & -2 & -1 \\ -1 & & -1 & 1 & 1 \\ \hline & 1 & -1 & -1 & \boxed{0} \end{array} \longrightarrow (x+1)(x^2-x-1)=0$$

$$x = \frac{1 \pm \sqrt{1-4\cdot(-1)}}{2} = \frac{1 \pm \sqrt{5}}{2} \begin{cases} \frac{1+\sqrt{5}}{2} \\ \frac{1-\sqrt{5}}{2} \end{cases}$$

$$(x+1)\left(x - \frac{1+\sqrt{5}}{2}\right)\left(x - \frac{1-\sqrt{5}}{2}\right) = 0$$

$$\boxed{T(n) = c_1(-1)^n + c_2\cdot\left(\frac{1+\sqrt{5}}{2}\right)^n + c_3\cdot\left(\frac{1-\sqrt{5}}{2}\right)^n}$$

$$T(n) \in O\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right)$$

7.

```
funcion Busquedaternaria(V[1...n]; primero, ultimo, clave: int)
    si primero ≥ ultimo entonces
        devolver V[ultimo] = clave
    fsi
    tercio ← ((ultimo - primero + 1) / 3)
    si clave = V[primero + tercio] entonces
        devolver cierto
    sino
        si clave < V[primero + tercio] entonces
            devolver Busquedaternaria(V, primero, primero+tercio-1, clave)
        sino
            si clave = V[ultimo - tercio] entonces
                devolver cierto
            sino
                si clave < V[ultimo - tercio] entonces
                    devolver Busquedaternaria(V, primero+tercio+1, ultimo, clave)
                sino
                    devolver Busquedaternaria(V, ultimo-tercio+1, ultimo, clave)
                fsi
            fsi
        fsi
    fsi
ffuncion
```

$$t(n)\begin{cases} 4 & si\ n=1 \\ T(n/2)+13 & si\ n>1 \end{cases}$$

$$\boxed{t(n) - t(n/2) = 13^n \cdot n^0}$$ No homogenea

$$\boxed{\begin{array}{l} n = 2^m \\ m = \log_2 n \end{array}} \longrightarrow T_m - T_{m-1} = 13 \cdot 1^m (2^m)^0$$

$$(x-1)(x-1)=0 \quad r_1 = 1 \text{ (doble)}$$

$$t(2^m) = c_1 (1)^m \cdot m^0 + c_2 (1)^m \cdot m^1$$

$$\boxed{t(n) = c_1 (1)^{\log_2 n} + c_2 (1)^{\log_2 n} \cdot \log_2 n}$$

$$t(2) = c_1 + c_2 \rightarrow c_1 = 2 - c_2 \rightarrow \boxed{c_1 = -2}$$

$$t(13) = c_1 + c_2 \cdot 2 \rightarrow \rightarrow 2 - c_2 + 15 c_2 \rightarrow \boxed{c_2 = 13}$$

$$\boxed{t(n) = 13 \log_2 n + 4 \in O(\log n)}$$

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.

7.

```
función Busquedaternaria (V [1 ... n]; primero, ultimo, clave: int)
    si primero ≥ ultimo entonces
        devolver V [ultimo] = clave
    fsi
    tercio ← ((ultimo - primero + 1) /3)
    si clave = V [primero + tercio] entonces
        devolver cierto
    sino
        si clave < V [primero + tercio] entonces
            devolver Busquedaternaria (V, primero, primero + tercio - 1, clave)
        sino
            si clave = V [ultimo - tercio] entonces
                devolver cierto
            sino
                si clave < V [ultimo - tercio] entonces
                    devolver Busquedaternaria (V, primero + tercio + 1, ultimo, clave)
                sino
                    devolver Busquedaternaria (V, ultimo - tercio + 1, ultimo, clave
                fsi
            fsi
        fsi
    fsi
ffuncion
```

$$T(n) \begin{cases} 4 & \text{si } n=1 \\ T(n/2)+13 & \text{si } n>1 \end{cases}$$

$$\boxed{T(n) - T(n/2) = 13^n \cdot n^0}$$ No homogenea

$$\boxed{\begin{array}{l} n = 2^m \\ m = \log_2 n \end{array}} \rightarrow T_m - T_{m-1} = 13 \cdot 1^m (2^m)^0$$

$(x-1)(x-1)=0 \quad r_1 = 1$ (doble)

$$T(2^m) = C_1 (1)^m \cdot m^0 + C_2 (1)^m \cdot m_1$$

$$\boxed{T(n) = C_1 (1)^{\log_2 n} + C_2 (1)^{\log_2 n} \cdot \log_2 n}$$

$$T(2) = C_1 + C_2 \rightarrow C_1 = 2 - C_2 \rightarrow \boxed{C_1 = -2}$$

$$T(13) = C_1 + C_2 \cdot 2 \rightarrow \rightarrow 2 - C_2 + 15 C_2 \rightarrow \boxed{C_2 = 13}$$

$$\boxed{T(n) = 13 \log_2 n + 4 \in O(\log n)}$$

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**log* n). Analiza el algoritmo y demuestra que es así.

```
funcion busca (v[1..n], x : int)
    N1, N2 : int
    vord : [1..n]   //Almacena el vector v ordenado
empezar
    Mergesort (v, vord);
    para ind = 1 hasta n-1 hacer
        N1 ← vord(ind)
        N2 ← x - N1
        BusquedaBinaria (vord, ind+1, n, N2, pos)
        si pos ≠ 0 entonces
            devolver (vord(ind), vord(pos))
        fsi
    fpara
    devolver (0,0) // Suponemos 0 los valores devuelto cuando no se
                   // encuentra el numero y x ≠ 0
ffuncion
```

$$T(n) = T(merge) + T(para) = O(n \log n) + [O(n) \cdot O(\log n) \cdot k]$$

$$\boxed{T(n) = O(n \log n) + O(n \log n)} \rightarrow T(n) \in O(n \log n)$$

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;
```

$$T(n) = 1+2+\sum_{i=1}^{n-1}\left(1+T(\text{posMinimo})+1+T(\text{intercambia})\right)+1$$

$$T(\text{posminimo}) = 1+2+1+\left(\sum_{i=\text{primero}+1}^{\text{ultimo}}(3+1+1+2)+1\right)+1 = 5+\left(\sum_{i=\text{primero}+1}^{\text{ultimo}}(6)+1\right)$$

$$\boxed{T(\text{posminimo}) = 6+6(\text{ultimo}-\text{primero}+1+1)}$$

$$T(\text{intercambia}) = 2+3+2 = \boxed{7}$$

luego queda que:

$$T(n) = 1+2+\left(\sum_{i=1}^{n-1}\left(1+(6+6(n-i))+1+(7)+1+2\right)+1\right)$$

$$T(n) = 4+\sum_{i=1}^{n-1}(18+6n-6i) = 4+(18+6n)(n-1-1+1)+6\sum_{i=1}^{n-1} i$$

$$T(n) = 4+(18+6n)(n-1)+6\cdot\frac{n(n-1+1)}{2}$$

$$T(n) = 4+18n-18+6n^2+\frac{6n^2}{2}$$

$$T(n) = 6n^2+3n^2-18n-6n+4-18$$

$$\boxed{T(n) = 9n^2-12n-14} \in O(n^2)$$

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:

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

$t(n) = 2t(n/2) + n\log n + n \log n$

a) ecuación característica

$t(n) - 2t(n/2) = 2\, n \log n$

$\boxed{\begin{array}{l} n = 2^k \\ k = \log n \end{array}}$ $\quad t(2^k) \cdot 2t(2^{k-1}) = 2 \cdot 2^k \cdot \log(2^k)$

$t(2^k) = t_k \longrightarrow t_k - 2t_{k-1} = 2 \cdot \underbrace{2^k \cdot k^1}_{b^k \cdot p(k) = 0} \cdot \cancel{\log_2 2}^{1}$ $\quad \begin{cases} b = 2 \\ d = 1 \end{cases}$

$(x-2)(x-2)^{1+1} = 0$ $\quad \{ x_1 = 2$ triple

$t_k = c_1 \cdot 2^k \cdot k^0 + c_2 \cdot 2^k \cdot k^1 + c_3 \cdot 2^k\, k^2$

$T(n) = c_1 \cdot 2^{\log_2 n} + c_2 \cdot 2^{\log_2 n} \cdot \log_2 n + c_3 \cdot 2^{\log_2 n} (\log_2 n)^2$

$\boxed{t(n) = c_1\, n + c_2\, n\log_2 n + c_3\, n \log_2^2 n} \in \Theta(n \log^2 n)$

b) Teorema maestro

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

$a = 2 \quad a = b^k$

$b = 2 \quad 2 = 2^1$

$p = 1 \quad t(n) \in \Theta(n^k \cdot \log^{p+1} n)$

$k = 1 \quad t(n) \in \Theta(n^1 \log^{1+1} n)$

$\hookrightarrow \boxed{t(n) \in \Theta(n \log^2 n)}$

c)

$$\lim_{n\to\infty} \frac{n \log^2 n}{n^2} = \lim_{n\to\infty} \frac{\log^2 n}{n} = \lim_{n\to\infty} \frac{2 \log n}{n^2 \ln 2} = 0$$

$\boxed{\Theta(n \log^2 n) \subset \Theta(n^2)}$

la versión divide y vencerás es más eficiente.

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.

1. función NacciDV (n: entero): real
inicio
si n = 1 entonces
devuelve 1
Sino
si n = 2 entonces
devuelve 3
sino
devuelve 3/2 * NacciDV (n-1) + NacciDV (n-2)
fsi
fsi
ffuncion

$T(n) - 3/2 \cdot T(n-2) = 0$ Homogenea

$(x^2 - 3/2\, x - 1) = 0 \rightarrow x = \dfrac{3/2 \pm \sqrt{9/4 + 4}}{2} = \begin{cases} x_1 = 2 \\ x_2 = -1/2 \end{cases}$

$(x-2)(x+1/2) = 0 \rightarrow r_1 = 2$
$r_2 = -1/2$

$T(n) = c_1 (2^n) n^0 + c_2 (-1/2)^n \cdot n^0 \rightarrow T(n) \in O(2^n)$

2. función NacciIt (n: entero): real
var:
V [1 ... n]: real
V [1] ← 1
V [2] ← 3
inicio
para i ← 3 hasta n hacer
V[i] ← 3/2 · V [i-1] + V [i-2]
fpara
devuelve V [n]
ffunción

$T(n) = 1 + 1 + \left(\sum_{i=3}^{n} (9 + 1 + 1) + 1\right) \rightarrow T(n) = 3 + \sum_{i=3}^{n} (11)$

$T(n) = 3 + 11\,(n - 3 + 1) \rightarrow T(n) = 3 - 22 + 11n$

$\boxed{T(n) = 11n - 19} \in O(n)$

3.

funcion Naccibinet (n: entero): real
  inicio
    devuelve $7/10 * 2^n + 4/5 * (-1/2)^n$
funcion

$O(1) \in O(n) \in O(2^n) \rightarrow O(naccibinet) \in O(nacciIt)$
$\in O(nacciRec)$.

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 €.**