1. Analizar la complejidad (peor caso) de los siguientes algoritmos

   a) INPUT = V un vector con n números
   1. $s \leftarrow 0$
   2. Desde $i = 1$ hasta $i = n$
   3. s $\leftarrow V[i]$.
   4. Fin Desde
   5. Retornar s.

   b) INPUT = V un vector de n números y e un número.
   1. $i \leftarrow 1$
   2. Mientras $i < n$
   3. Si $V(i) = e$
   4. Imprimir i
   5. Finalizar
   6. Fin Si
   7. Fin Mientras
   8. Imprmir "no se encontró".

   c) Dado un número natural n:
   función $Factorial(n)$
   Si $n = 0$ devolver 1
   sino devolver $n \times Factorial(n-1)$
   fin función

   d) INPUT = un vector V con n números.

   1. Crear un vector W con n ceros.
   2. Desde $i = 1$ hasta n
   4. Desde $j = i$ hasta n
   5. $W[i] \leftarrow W[i] + V(j)$
   6. Fin Desde
   7. Fin Desde
   8. Retornar W.

2. ¿Cuál es la complejidad en el peor caso y en el caso promedio del algoritmo de búsqueda secuencial?

3. Escribir un algoritmo que permita calcular el máximo de un arrgelo de n elementos. Estudiar la complejidad.

4. Implementar un algoritmo de búsqueda binaria para poder encontrar un elemento dentro de un vector ya ordenado. Estudiar la complejidad.

5. a) Escribir algoritmos para realizar suma, resta y multiplicación de matrices. Determinar la complejidad y expresarla en función de la dimensión de las matrices.
   b) Escribir un algoritmo que dada una matriz A y un número n calcule A^n (o sea $A \times A \times \ldots \times A$ n veces). Determinar la complejidad del mismo. Fíjese si no se puede hacer esta cuenta de una manera "astuta".

c) Escribir y determinar la complejidad del algoritmo de Gauss (triangulación) para resolver un sistema de n ecuaciones lineales con n incógnitas.

d) Escribir una función que permita calcular el determinante de una matriz usando el método de los cofactores. Estudiar la complejidad.

6. Implementar los algoritmos de ordenamiento por selección y por inserción. Calcular su complejidad.

7. Considere el siguiente algoritmo de ordenamiento para un vector de n elementos:

Función $Quicksort(V : vector)$
Si longitud(V) ≤ 1
  retornar V
$S_1 \leftarrow []$
$S_2 \leftarrow []$
$x \leftarrow$ elegirPivote(V)
Sacar a x de V.
Desde $i = 1$ hasta longitud(V)
  Si $V[i] \leq x$ luego $S_1 \leftarrow S_1 \circ V[i]$
  Sino $S_2 \leftarrow S_2 \circ V[i]$
Fin Desde
retornar $Quicksort(S_1) \circ [x] \circ Quicksort(S_2)$
Fin Función

La función *elegirPivote* retorna un elemento del vector parámetro con algún criterio.

a) Si *elegirPivote* devuelve el primer elemento, analizar la complejidad en el peor caso.

b) Supongamos que el pivote elegido fuese tal que siempre se parta al vector en dos mitades iguales, ¿cuál sería la complejidad en el peor caso? Esto pasaría si por ejemplo el pivote fuese la mediana y todos los elementos del vector son distintos.

c) Analizar la complejidad en el peor caso suponiendo que el pivote elegido es siempre tal que deja 9/10 de los números en S_1 y el resto en S_2.

d*) Supngamos ahora que *elegirPivote* retorna un elemento al azar del arreglo, ¿cuál sería la complejidad en el caso promedio?

8. Otro algoritmo (incompleto) para ordenar n elementos:

Función $Mergesort(V : vector)$
Si longitud(V) ≤ 1
  retornar V
$i \leftarrow$ longitud(V) / 2
$S_1 \leftarrow V[1 \dots i]$
$S_2 \leftarrow V[i+1 \dots long(V)]$
$V_1 \leftarrow Mergesort(S_1)$
$V_2 \leftarrow Mergesort(S_2)$
Retornar Mezclar(V_1, V_2)
Fin Función

a) Implementar la función "Mezclar" para que esto quede efectivamente un algoritmo de ordenamiento.

b) Analizar la complejidad de este algoritmo (puede suponer que la longitud de V es potencia de 2, si eso le ayuda).

9. (*) Demuestre que cualquier algoritmo que ordene un vector de n elementos realizando comparaciones entre elementos requerirá por lo menos $nlog_2(n)$ en el peor caso.

10. a) Diseñar un algoritmo que dada una secuencia de números determine si existe un número repetido o no. Calcular la complejidad.

    b) Diseñar un algoritmo para calcular la moda de una secuencia de números. Nota: la moda es el número que más veces se repite. Como siempre, calcular la complejidad.

11. *a*) Escribir un procedimiento que dado un número natural determine si éste es primo o no.

    b) Escribir un procedimiento que dado un número natural devuelva su factorización en números primos.

    c) Analizar la complejidad.