Asignatura: Fundamentos de algoritmia
Evaluación continua. Cuestionario.

Profesor: Isabel Pita.

Nombre del alumno: Victoria Martín Rojo.

1. Enumera las tres propiedades que se han estudiado en clase sobre el orden de complejidad $\mathcal{O}(f(n))$.

Const no afectan al orden ✓
Logs da igual a la base ✓
Suma de costes = mayor ✓

2. Compara los siguientes órdenes de complejidad indicando cuál está incluido en el otro.
   a) $\mathcal{O}(n^2)$, $\mathcal{O}(2^n)$ n^2 ~~contenido~~ incluido en 2^n, a partir de un bajo número para n 2^n crece más que n^2 ✓
   b) $\mathcal{O}(\log_2 n)$, $\mathcal{O}(\log_{10} n)$ log 2 ~~incluido~~ = ~~en~~ log 10 pues al ser igual a la base, el 2<10.
   c) $\mathcal{O}(n \log n)$, $\mathcal{O}(n^2)$ n log n incluido en n^2 ✓
   d) $\mathcal{O}(n^n)$, $\mathcal{O}(2^n)$ 2^n incluido en n^n, n^n muchísimo mayor, crece más rápido ✓
   e) $\mathcal{O}(n^3)$, $\mathcal{O}(n^2)$ n^2 incluido en n^3 ✓

3. Dado el problema de contar el número de veces que aparece el valor mínimo de un vector v, resuelto con el siguiente algoritmo:

```
int m = v[0]; int numVeces = 1;
for (int i = 1; i < v.size(); ++i)
    if (m < v[i]) { m = v[i]; numVeces = 1; } -1
    else if (m == v[i]) ++numVeces; -> 1
```

n · = n · 1 · 1 = n.

Hay que sumar el coste de todas las instrucciones

   a) Calcula la función de coste del algoritmo analizando cada una de las instrucciones.
   b) Indica el orden de complejidad al que pertenece la función de coste anterior.

a) En caso de una lista ordenada de mayor a menor, el mínimo aparece n veces ~~(el caso mayor)~~ (el mayor coste)
~~Por lo que la función sería O(n)~~

Esto se debe concluir una vez se de la función de coste

b) Orden por lo tanto es O(n) ~~ya que recorremos el for 1 sola vez y el~~ y el ~~if - else if no aumentan la complejidad pues solo~~

Indicar que representa n en los datos de entrada.

4. Dado el problema de contar el número de veces que aparece el valor mínimo de un vector v, resuelto con el siguiente algoritmo:

```
int m = v[0];
for (int i = 1; i < v.size(); ++i)
    if (m < v[i]) m = v[i];
int numVeces = 0;
for (int i = 0; i < v.size(); ++i)
    if (m == v[i]) ++numVeces;
```

max se recorre n veces } n veces · 2

max = n veces

sumar el coste de todas las instrucciones

a) Calcula la función de coste del algoritmo analizando cada una de las instrucciones.

b) Indica el orden de complejidad al que pertenece la función de coste anterior.

a)

b) Orden = $2n \to O(n)$ pues ~~repite 2 veces~~ se ~~recorre dos veces el vector~~. → se deduce de la función de coste

5. Dada una matriz m de dimensión $m_1 \times m_2$ indica el orden de complejidad en tiempo más ajustada del siguiente algoritmo que calcula la suma de todos los elementos de la matriz y justifícala indicando el número de vueltas que da cada bucle y el coste de cada vuelta.

```
int suma = 0;
for (int i = 0; i < m1; ++i)
  for (int j = 0; j < m2; ++j)
        suma += m[i][j];
```

$O(n^2)$ $m1 \cdot m2$

$n = n \cdot n = n^2$ $m2$

m1 y m2 son 2 cosas diferentes

~~m1 pongamos que es n~~

m1 recorre cada fila, desde la 0 hasta la m2 ~~n~~ m1 veces por lo que el orden es $m2 \cdot m1 = n^2$

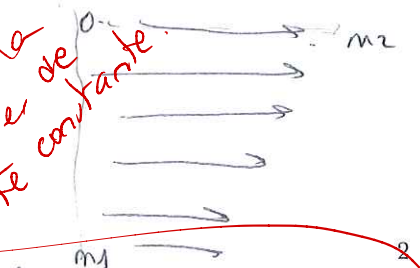
m2

Cada vuelta tiene coste 1

6. Dada una matriz m de dimensión $m_1 \times m_2$ indica el orden de complejidad en tiempo más ajustada del siguiente algoritmo que calcula la suma de todos los elementos de la matriz y justifícala indicando el número de vueltas que da el bucle y el coste de cada vuelta.

```
int suma = 0; int i = 0; int j = 0;
while (i < m1) {
    suma += m[i][j];
    if (j < m2-1) ++j;
    else {++i; j = 0;}
}
```

el bucle da $m1 \cdot m2$ vueltas y cada vuelta ~~es~~ tiene un coste de constante pues hasta que j no es ~~aaa~~ igual a m2 no ~~empieza~~ la ~~siguiente vuelta sube el~~ cambia el valor de i, por lo que el bucle se queda en bucle hasta que no sea $j = m2$.



NO. Da $n1 \cdot n2$ vuelta y cada vuelta es de coste constante.

Orden ~~coste~~ por vuelta = $O(n)$

Coste total = $O(n) \cdot O(n) = O(n^2)$ → bucle da n^2 vueltas

Orden

Cada vuelta tiene un coste 1

$m1 \cdot m2$ $(n1 \cdot n2)$