EJERCICIOS TEMA 1

1.1: Escribir un algoritmo que calcule el cociente y el resto de una división entera mediante restas. Analizar el algoritmo.

1.2: Escribir un algoritmo que reproduzca el método tradicional de multiplicar 2 números enteros. Suponer que las cifras de los enteros están almacenadas en vectores de tamaño n.

1.3: Escribir el algoritmo correspondiente a la multiplicación rusa: Se parte de dos enteros x e y. Sucesivamente uno se va multiplicando por 2 y el otro se va dividiendo por 2. Cuando la división (entera) de como resto 1, el entero que va a multiplicarse se añade a un acumulador. El proceso se repite hasta que el entero que se divide vale cero. (El caso de multiplicar 26 por 14 se muestra a continuación):

x	y	acumulador
26	14	0
13	28	28
6	56	28
3	112	140
1	224	364

1.4: Hacer un análisis comparativo de los algoritmos para multiplicar dos enteros (método tradicional y la multiplicación rusa).

1.5: Analizar el algoritmo de búsqueda lineal en el caso que: a) La probabilidad de encontrar el elemento en una posición i, sea proporcional a i, b) Lo mismo pero inversamente proporcional a i, y c) Que el vector sea de enteros de 16 bits y cada número tenga la misma probabilidad de aparecer en una posición determinada del vector.

1.6: Analizar los siguientes algoritmos.

```
ALGORITMO B1
DATOS:      n: Nat
            s: Nat
METODO:
      s ← 0
      Para i ← 1 hasta n hacer
            Para j ← 1 hasta i hacer
                  s ← s + 1
            fPara
      fPara
FB1
```

```
ALGORITMO B2
DATOS:      n: Nat
            s: Nat
METODO:
      s ← 0
      Para i ← 1 hasta n hacer
            Para j ← 1 hasta i^2 hacer
                  Para k ← 1 hasta j hacer
                        s ← s + 1
                  fPara
            fPara
      fPara
FB2
```

1.7: Analizar el siguiente algoritmo:.

```
ALGORITMO Que-no-hace-nada
DATOS:      n: N
METODO:
      i ← n
      mientras i > 0 hacer
            j ← i
            repetir
                  j ← j*2
            hasta j > n
            i ← ⌊i/2⌋
      fmientras
fQue-no-hace-nada
```

1.8 Demostrar si es cierto o no que: $n - \lg n \in \Theta(n)$

1.9 ¿Hay alguna diferencia entre $\Theta(n^2 + \lg n)$ y $n^2 + \Theta(\lg n)$?. ¿Cual?

1.10 Los costes de dos algoritmos que resuelven el mismo problema son $O(n^2)$ y $\Theta(n^2)$ respectivamente. ¿Se puede decir cual de los dos es más eficiente? ¿Por qué?. ¿Y si los costes fuesen $O(n) + \lg n$ y $O(\lg n) + n$?. Estudia en detalle este segundo caso.

1.11 Contestar V o F;

a) $\frac{\lg n}{n} sen(n!) + 10 \in \Theta(1)$ [ ] b) $(n^2 + n + 1)\sqrt{n} \in O(n\sqrt{n})$ [ ] c) $\lg\lg n \in O(\sqrt{\sqrt{n}})$ [ ]

d) $O(n^2) = o(n^2) \cup \Theta(n^2)$ [ ] e) $\Theta(n) \subset O(n^2) \cap \Omega(\sqrt{n})$ [ ]

1.12 Contestar V o F;

a) $n \lg n \in O(n\sqrt{n})$ [ ] b) $\sqrt{n} \lg n \in \Omega(n)$ [ ] c) $\lg(n^2) \in \Theta(\lg n)$ [ ]

d) $O(n) \subset \Omega(n) \cup \Theta(n)$ [ ] e) $\Theta(5n^2 + 2n + 3) = \Theta(n^2)$ [ ]

1.13 Contestar V o F;

a) $\lg(n^n) \in O(n^2)$ [ ] b) $n^3 \in \Omega(n!)$ [ ] c) $n^2 \in \Theta(n^3)$ [ ]

d) $O(n) \bigcap \Omega(2n) = \varnothing$ [ ] e) $n \cdot \sin(n) \in \Theta(n)$ [ ]

1.14 Analizar el siguiente algoritmo. Decir cuál es su talla, cuántas instancias tiene y los costes exactos en los casos mejor, peor y medio, para este último teniendo en cuenta que todas las instancias son equiprobables. Dar los mismos costes utilizando la notación asintótica Θ.

a)

```
Algoritmo num_ceros(a: vector[1..n][1..n] de nat) dev ceros: nat
ceros ← 0
para i ← 1 hasta n
       para j ← 1 hasta n
              si a[i][j] = 0 entonces
                     ceros ← ceros + 1
              fsi
       fpara
fpara
```

b)

```
Algoritmo f(a: vector[1..n][1..m] de nat) dev b: vector[1..n][1..m] de nat
para i ← 1 hasta n
       si a[i][1] ≠ 0 entonces
              aux ← a[i][1]
              para j ← 1 hasta m
                     b[i][j] ← a[i][j] / aux
              fpara
       fsi
fpara
```

1.15 Analizar el siguiente algoritmo. Calcular el coste medio suponiendo que la probabilidad de que a[1, n] sea igual a a[n, 1] es del 25 %.

```
Algoritmo simetrica(a: vector[1..n][1..n] de nat) dev s: nat
sim ← cierto
s ← 0
si a[1, n] ≠ a[n, 1] entonces
       para i ← 1 hasta n
              para j ← 1 hasta i
                     s ← s + a[i, j]
              fpara
       fpara
fsi
```