1. Para cada uno de los tiempos que toma un algoritmo en terminar, hallar el orden de complejidad correspondiente:

a) $T_A(n) = 2n^3 - 3n + 1$

→ $O(n^3)$

Jesús D. García cuello

b) $T_A(n) = n^5 + 4^2 - \sqrt{n} + 1$

→ $O(n^5)$

c) $T_A(n) = n^2 \log n - 2n^3 \log n + 2$

→ $O(n^2 \log n)$

2. Calcule la complejidad del siguiente algoritmo

```
fun XXXXXX(n: Int): Unit
{
   1. var x = 0;                          O(1)
   2. for (i in 1 .. n step 5)            O(n)
      {
        2.1 var j = 1                     O(1)
        2.2 while (j <= n)                O(log n)   (O(log_4 n - 1))
            {
            2.2.1  x = x + j;             O(1)
            2.2.2 j *= 4                  O(1)
            }
        2.3 for (k in n downto 1 step 2)  O(n)
            {
            2.3.1  x = x + 1;             O(1)
            }
      }
}
```

La complejidad es de $O(n^2)$

3. (5%) Responda las siguientes preguntas a partir del código de la siguiente función:

```
fun buscar(a: IList<Int>, elem: Int): Int {
    val n: Int = a.size
    for (i in 0 until n) {
        if (a[i] == elem) {
            return i
        }
        else {
            return -1
        }
    }
}
```

Para todo caso, es decir, siempre, este ciclo realiza 1ra sola iteración

a. ¿Cuántas veces se repite el **for** que está dentro de la función?

Se ejecuta 1ra vez.

b. A partir de la respuesta anterior, ¿cuál es la complejidad de la función **buscar** presentada previamente?

La complejidad se puede representar con la notación Big O de la siguiente forma: O(1)

4. (5%) Ordene las siguientes complejidades de los siguiente tiempos, de la mejor a la peor

5. $4n \log n + 2n \lesssim 4n^2+2n$	1. 2^{10}	8. $2^{\log n}$
3. $3n + 100 \log n \lesssim 3n + 100n \cong 103n$	2. $4n$	9. 2^n
6. $n^2 + 10n$	7. n^3	4. $n \log n \lesssim n^2$

① 2^{10}	④ $n \log n$	⑦ n^3
② $4n$	⑤ $4n \log n + 2n$	⑧ $2^{\log n}$
③ $3n + 100 \log n$	⑥ $n^2 + 10n$	⑨ 2^n

5. (10%) Suponga que se tienen los siguientes dos algoritmos para resolver el mismo problema, y suponga que la función **buscar** tiene complejidad $O(\text{nlog}\,n)$.

```
fun Ex1(a: IList<Int>, elem: Int): Boolean {
    val pos: Int = buscar(a, elem)→O(nlogn)
    val n: Int = a.size → O(1)
    var x: Int = pos → O(1)

    for (i in 0 until n) { → O(n)
        x += 2 → O(1)
        for (j in 0 until n) { O(n)
            if (a[j] > a[pos]) { O(1)
                x++ O(1)
            }
        }
    }
    return x > elem → O(1)
}
```

La complejidad es de: $O(n^2)$

```
fun Ex2(a: IList<Int>, elem: Int): Boolean {
    val n: Int = a.size → O(1)
    var x: Int = 0 → O(1)

    for (i in 0 until n) { → O(n)
        val pos: Int = buscar(a, elem)→O(nlogn)
        x += pos + 2 → O(1)
        for (j in 0 until n) {→O(n)
            if (a[j] > a[pos]) {→O(1)
                x++ → O(1)
            }
        }
    } → O(nlogn)
    return x > elem → O(1)
}
```

La complejidad es de $O(n\log n)$

Indique con cuál de los dos algoritmos se queda para resolver el problema. Justifique muy bien su respuesta.

De ambas soluciones, escogería la primera (Ex1, subrayada en amarillo) por el siguiente motivo:

- La función Ex1 tiene un orden de complejidad menor, es decir, le costará al ordenador menos tiempo de ejecución a medida que n se hace más grande.