1. Complejidad de los Algoritmos

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

Para demostrar que (T_A(n) = 2n^3 - 3n^2 + 1) es (O(n^3)), debo encontrar una constante (C) y un valor (n_0) tales que (T_A(n) leq C cdot n^3) para todo (n geq n_0).

1. Observando (T_A(n)), el término dominante es (2n^3). Los términos (-3n^2) y (1) son menores comparados con (n^3).

2. Para (n geq 1), (2n^3 - 3n^2 + 1 leq 2n^3) siempre se cumple.

3. Tomando (C = 3) y (n_0 = 1), tenemos que (T_A(n) leq 3n^3) para todo (n geq 1). Así, (T_A(n)) es (O(n^3)).

b) (T_A(n) = n^5 + 42 - sqrt{n} + 1) es (O(n^5))

Para mostrar que (T_A(n) = n^5 + 42 - sqrt{n} + 1) es (O(n^5)), necesito encontrar una constante (C) y un valor (n_0) tales que (T_A(n) \leq C cdot n^5) para todo (n geq n_0).

1. Observando (T_A(n)), el término dominante es (n^5). Los términos (42), (-sqrt{n})y (1) son mucho menores en comparación con (n^5).

2. Para (n geq 1), (n^5 + 42 - sqrt{n} + 1 leq n^5 + (42 - sqrt{n} + 1)) se cumple porque (42 - sqrt{n} + 1) es insignificante comparado con (n^5).

3. Tomando (C = 2) y (n_0 = 1), (T_A(n) \leq 2n^5) para todo (n geq 1). Así, (T_A(n)) es (O(n^5)).

c) \(T_A(n) = n^2 log n + 2n^4 + 2sqrt{n}) es (O(n^4))

Para demostrar que (T_A(n) = n^2 log n + 2n^4 + 2sqrt{n}) es (O(n^4)), debo encontrar una constante (C) y un valor (n_0) tales que (T_A(n) leq C cdot n^4) para todo (n geq n_0).

1. Observando (T_A(n)), el término dominante es (2n^4). Los términos (n^2 log n) y (2sqrt{n}) son menores comparados con (n^4).

2. Para (n geq 1), (n^2 \log n + 2n^4 + 2sqrt{n} leq 2n^4 + (n^2 log n + 2sqrt{n})) porque (n^2 log n) y (2sqrt{n}) son insignificantes comparados con (n^4).

3. Tomando (C = 3) y (n_0 = 1), (T_A(n) leq 3n^4) para todo (n geq 1). Así, (T_A(n)) es (O(n^4)).

2. Análisis de la Complejidad del Algoritmo

Vamos a analizar la complejidad del siguiente algoritmo:

1. Primer bucle:

  - Itera con un incremento exponencial hasta que el valor supera (n).

  - El número de iteraciones es aproximadamente (log_5(n)), lo que corresponde a (O(log n)).

2. Dentro del primer bucle:

  - Segundo bucle:

    - Itera con un incremento lineal en pasos de 2 hasta (n).

    - El número de iteraciones es aproximadamente (frac{n}{2}), que se simplifica a {O(n)].

  - Tercer bucle:

    - Itera con una división exponencial hasta llegar a 1.

    - El número de iteraciones es aproximadamente (log_2(n)], lo que corresponde a [O(log n)].

3. Complejidad total:

  - El primer bucle tiene (O(log n)) iteraciones.

  - Dentro de este bucle, el segundo bucle tiene (O(n)) iteraciones.

  - Dentro del mismo primer bucle, el tercer bucle tiene (O(log n)) iteraciones.

La complejidad total del algoritmo es el producto de las complejidades de estos bucles:

[O(log n) times O(n) times O(log n) = O(n (log n)^2)]

Por lo tanto, la complejidad total del algoritmo es [O(n (log n)^2)]

3. Análisis Adicional

1. `buscar(a, elem)` tiene (O(n log n)).
2. El primer bucle tiene (O(n)).
3. El segundo bucle tiene (O(n^2)) (anidado dentro del primero).