Laboratorio 8

martes, 7 de octubre de 2025 6:42 p. m.

Problema 3: 50%

Analice el siguiente programa:

```
1  void function(int n) {
2      int i, j;
3      for (i = 1; i <= n/3; i++) {
4          for (j = 1; j <= n; j += 4) {
5              printf("Sequence\n");
6          }
7      }
8  }
```

Parte a

Encuentre la complejidad de tiempo en notación Big-Oh. Muestre todo su procedimiento.

For $(i = 1 ; i < n/3; i++)$

$\frac{n}{3} - 1 + 1 = \frac{n}{3}$ $O_T(T_1) = O(n)$

For $(j = 1; j \leq n; j += 4)$

$\frac{n-1}{4} + 1 = \frac{n}{4} + \frac{3}{4} = O(n)$

Complejidad : $O(B_1) \cdot O(B_2) = O(n^2)$ //

Problema 4: 25%

Encuentre el **mejor caso**, **caso promedio** y **peor caso** del algoritmo de Búsqueda Lineal (Linear Search, Binary Search y Quick sort). Muestre todo su procedimiento.

Linear Search

Mejor caso

El elemento está solo en una posición.
Complejidad : $O(1)$

Caso promedio

El elemento está en cualquier posición intermedia.
Complejidad: O(n)

Peor caso

El elemento esta al final o no está en el arreglo.
Complejidad: O(n)

Binary Search

Mejor caso

El elemento se encuentra justo en el medio al primer intento.
Complejidad: O(1)

Caso promedio

El elemento se encuentra después de dividir varias veces, en un número de pasos proporcional a $\log_2(n)$.
Complejidad: O(log n)

Peor caso

El elemento no está presente, y el algoritmo sigue dividiendo hasta un único elemento.
Complejidad: O(log n)

Quick Sort

Mejor caso

El pivote divide el arreglo en dos mitades iguales en cada paso.
Complejidad: O(n log n)

Caso promedio

Las divisiones son más o menos equilibradas
Complejidad: O(n log n)

Las divisiones son más o menos equilibradas
Complejidad: $O(n \log n)$

Peor caso

El pivote es siempre el menor o mayor elemento
Complejidad: $O(n^2)$

Problema 5

Determine si los siguientes enunciados son verdaderos o falsos. **Justifique sus respuestas** para recibir créditos completos.

Teoría de la computación
Octubre , 2025

a) Si $f(n) = \Theta(g(n))$ y $g(n) = \Theta(h(n))$, entonces $h(n) = \Theta(f(n))$.

b) Si $f(n) = O(g(n))$ y $g(n) = O(h(n))$, entonces $h(n) = \Omega(f(n))$.

c) $f(n) = \Theta(n^2)$, donde $f(n)$ está definido como el tiempo de ejecución del siguiente programa Python:

```
def A(n):
    atupla = tuple(range(0, n))
    # Una tupla es una version inmutable de una lista
    # que puede ser hasheada
    S = set()
    for i in range(0, n):
        for j in range(i + 1, n):
            # Anade la tupla (i,...,j-1) al set S
            S.add(atupla[i:j])
```

a) Verdadero: $f(n) = \Theta(g(n))$ implica que existen constantes positivas c_1, c_2, n_1 tales que para $n \geq n_1$:
$c_1 g(n) \leq f(n) \leq c_2 g(n)$.

b) Verdadero: $f(n) = O(g(n))$ implica $\exists C_1, n_1$: $f(n) \leq C_1 g(n)$ para $n \geq n_1$.
$g(n) = O(h(n))$ implica $\exists C_2, n_2$: $g(n) \leq C_2 h(n)$ para $n \geq n_2$.
Combinando: $f(n) \leq C_1 C_2 h(n)$ para $n \geq \max(n_1, n_2)$. Esto equivale a $f(n) = O(h(n))$.

c) Falso:

- La creación de atupla = tuple(range(0,n)) toma $O(n)$.
- El doble bucle itera sobre todos los pares (i,j) con $0 \leq i < j \leq n-1$. Para un i fijo, j recorre

la creación de arupa = tuple(range(0,n)) toma O(n).

- El doble bucle itera sobre todos los pares (i,j) con $0 \leq i < j \leq n-1$. Para un i fijo, j recorre aproximadamente n-i-1 valores.
- La operación tiene un costo proporcional a la longitud de la tupla, por lo que este resulta en $O(n^3)$.