Teoría de la computación
Octubre , 2025

Laboratorio 8

Para este laboratorio cree un repositorio de github

- Incluya un `README.md` con instrucciones de ejecución.
- Para ejercicios sin código, cree una carpeta separada con respuestas en PDF.
- Grabe un video ($\leq$ 10 min) mostrando la ejecución de sus programas, súbalo a YouTube como no listado y enlácelo en el README.

Problema 1: 25%

Analice el siguiente programa

```
1  void function(int n) {
2      int i, j, k, counter = 0;
3      for (i = n/2; i <= n; i++) {
4          for (j = 1; j+n/2 <= n; j++) {
5              for (k = 1; k <= n; k = k*2) {
6                  counter++;
7              }
8          }
9      }
10 }
```

Parte a

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

Parte b

Implemente el programa en el lenguaje de su elección y utilice profiling para medir el tiempo de ejecución con los siguientes valores de n: 1, 10, 100, 1000, 10000, 100000, 1000000.
Presente los resultados en:

- Una tabla con tamaño de input vs. tiempo
- Una gráfica de tamaño de input vs. tiempo

Teoría de la computación
Octubre , 2025

Problema 2: 25%

Analice el siguiente programa:

```
void function(int n) {
    if (n <= 1) return;
    int i, j;
    for (i = 1; i <= n; i++) {
        for (j = 1; j <= n; j++) {
            printf("Sequence\n");
            break;
        }
    }
}
```

Parte a

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

Parte b

Implemente y realice profiling con $n \in \{1, 10, 100, 1000, 10000, 100000, 1000000\}$. Presente resultados en tabla y gráfica.

Problema 3: 50%

Analice el siguiente programa:

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

Parte a

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

Parte b

Implemente y realice profiling con $n \in \{1, 10, 100, 1000, 10000, 100000, 1000000\}$. Presente resultados en tabla y gráfica.

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.

Problema 5

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

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:

```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])
```

Problema 4

- Complejidad peor caso
  una última posición, es N-1

peor caso $= O(N)$

Mejor caso: se encuentra en primera posición

$O(1), \Omega(1)$

Caso promedio: hay misma probabilidad

$$\text{Suma} = \frac{N(n+1)}{2} \qquad \text{promedio } \frac{\frac{N(N+1)}{2}}{N} = \frac{N+1}{2}$$

R// Por lo que $O(N)$

Probalidad 5

A. Si $f(n) = \theta(g(n))$, $g(n) = \theta(h(n))$, $h(n) = \theta(f(n))$

Verdadero, Porque θ define una relación de igualdad de crecimiento histórico entre función. si $f(n)$ tiene el mismo ritmo de crecimiento que $g(n)$ y $g(n)$ lo tiene con $h(n)$.

B. Si $f(n) = \theta(g(n))$, $g(n) = O(h(n))$, $h(n) = \Omega(f(n))$

Verdadero, Porque O y Ω son transistores, la cadena $f(n) \leq g(n)$ y $g(n) \leq h(n)$ implica necesariamente que $f(n) \leq h(n)$, lo que importe a $h(n) = \Omega(f(n))$

C. falso, la complejidad es $\theta(n^3)$, no $\theta(n^2)$ que i y j, vayan n^2 para la operación. Un acto propiedad a la longitud del triple. Este resulta en $O(n^3)$