Laboratorio 8 — Teoría de la Computación

Nombre: aletzbarr11

Octubre 2025

Problema 1 — Análisis de Complejidad

El programa es:

```
void function(int n){
    int i, j, k, counter = 0;
    for(i = n/2; i <= n; i++){
        for(j = 1; j + n/2 <= n; j++){
            for(k = 1; k <= n; k = k*2){
                counter++;
            }
        }
    }
}
```

Análisis:

Bucle exterior: i va de $\lfloor n/2 \rfloor$ a n, es decir, aproximadamente $n/2 = \Theta(n)$ iteraciones.

Bucle medio: la condición $j+n/2 \leq n$ implica $j \leq n-n/2 = \lceil n/2 \rceil$, también $\Theta(n)$ iteraciones.

Bucle interior: k se duplica en cada paso ($k = 1, 2, 4, \dots$) hasta superar n. El número de iteraciones es $\lfloor \log_2 n \rfloor + 1 = \Theta(\log n)$.

Multiplicando las complejidades:

$$T(n) = \Theta(n) \cdot \Theta(n) \cdot \Theta(\log n) = \Theta(n^2 \log n) = O(n^2 \log n).$$

Problema 2 — Análisis de Complejidad

El programa es:

```
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;
        }
    }
}
```

Análisis:

El bucle exterior se ejecuta n veces.

El bucle interior contiene un `break` en la primera iteración, por lo que solo se ejecuta **una vez** por cada valor de i.

Por lo tanto, el cuerpo del bucle interno se ejecuta exactamente n veces.

Así, la complejidad es:

$$T(n) = \Theta(n) = O(n).$$

Problema 3 — Análisis de Complejidad

El programa es:

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

Análisis:

Bucle exterior: i va de 1 a $\lfloor n/3 \rfloor$, es decir, $\Theta(n)$ iteraciones.

Bucle interior: j comienza en 1 y aumenta en 4 hasta n, lo que da aproximadamente $n/4 = \Theta(n)$ iteraciones.

Por lo tanto:

$$T(n) = \Theta(n) \cdot \Theta(n) = \Theta(n^2) = O(n^2).$$

Problema 4 — Casos de Algoritmos

Búsqueda Lineal (Linear Search):

- Mejor caso: $O(1)$ — el elemento está en la primera posición.
- Caso promedio: $O(n)$ — el elemento está en una posición aleatoria.
- Peor caso: $O(n)$ — el elemento no está o está en la última posición.

- **Búsqueda Binaria (Binary Search)** (requiere arreglo ordenado):
- Mejor caso: $O(1)$ — el elemento está en la mitad del arreglo.
- Caso promedio: $O(\log n)$.
- Peor caso: $O(\log n)$ — se reduce el espacio a la mitad en cada paso.
- **Quicksort**:
- Mejor caso: $O(n \log n)$ — particiones balanceadas (pivote cerca de la mediana).
- Caso promedio: $O(n \log n)$ — asumiendo datos aleatorios o pivote aleatorio.
- Peor caso: $O(n^2)$ — particiones desbalanceadas (e.g., arreglo ordenado y pivote extremo).

Problema 5 — Verdadero o Falso

a) **Verdadero.**
La notación Θ es una relación de equivalencia (reflexiva, simétrica y transitiva).
Si $f(n) = \Theta(g(n))$ y $g(n) = \Theta(h(n))$, entonces por transitividad $f(n) = \Theta(h(n))$,
y por simetría, $h(n) = \Theta(f(n))$.

b) **Verdadero.**
Si $f(n) = O(g(n))$, entonces $\exists c_1 > 0, n_1$ tal que $f(n) \leq c_1 g(n)$ para $n \geq n_1$.
Si $g(n) = O(h(n))$, entonces $\exists c_2 > 0, n_2$ tal que $g(n) \leq c_2 h(n)$ para $n \geq n_2$.
Luego, $f(n) \leq c_1 c_2 h(n)$ para $n \geq \max(n_1, n_2)$,
lo cual implica que $h(n) = \Omega(f(n))$ por definición.

c) **Falso.**
El programa genera todas las subsecuencias contiguas $a[i:j]$ con $0 \leq i < j \leq n$.
El número de pares (i, j) es $\sum_{i=0}^{n-1}(n - i - 1) = \frac{n(n-1)}{2} = \Theta(n^2)$.
Sin embargo, cada operación `S.add(atupla[i:j])` copia una subsecuencia de longitud $j - i$,
y la longitud promedio de estas subsecuencias es $\Theta(n)$.
Por lo tanto, el tiempo total es $\Theta(n^2) \cdot \Theta(n) = \Theta(n^3)$, no $\Theta(n^2)$.