

Laboratorio 8

Ejercicio 1.

```
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++;  
            }  
        }  
    }  
}
```

1.
① $n/2$
② $n/2$
③ ~~$n/2$~~

$\log_2 n$

① La variable i tiene un valor inicial de $n/2$. Ya que su incremento es de uno en uno, este ciclo correrá $n/2$ veces hasta llegar a n .

② La variable j tiene un valor inicial de uno. Su incremento va de uno en uno. La condición indica que el valor de j más $n/2$ debe ser menor a n . Esto ocurrirá hasta que $j = n/2$, es decir, $j + n/2 = n$.

③ El incremento sigue la función 2^k . Ya que mi condición es $k \leq n$, yo quiero saber cuando $2^k \leq n$. Si consideramos $2^k = n$, esto se rescribe cómo $\log_2 n = k$. Es decir, la instrucción interna se ejecutará $\log_2 n$.

$$f(n) = \frac{n}{2} * \frac{0}{2} + \log_2 n + 1 = \frac{n^2}{4} \log_2 n + 1$$

$$O\left(\frac{n^2 \log_2 n}{4} + 1\right) = n^2 \log_2 n \quad c = 1$$

$$f(n) = O(n^2 \log_2 n)$$

$$\frac{n^2}{4} \log_2 n + 1 \leq n^2 \log_2 n$$

$$\frac{n^2 \log_2 n}{4} - n^2 \log_2 n \leq -1$$

$$n^2 \log_2 n \left(\frac{1}{4} - 1\right) \leq -1$$

$$n^2 \log_2 n \left(-\frac{3}{4}\right) \leq -1$$

$$n^2 \log_2 n \left(\frac{3}{4}\right) \geq 1$$

$$n^2 \log_2 n \geq \frac{4}{3}$$

Ejercicio 2

```
Void function (int n) {  
    if (n <= 1) return;  
    int i, j;  
    for (i = 1; i <= n; i++) {  
        for (j = 1; j <= n; j++) {  
            Print ("Scavence\n");  
            break;  
        }  
    }  
}
```

1
1
1
① n
② 1

① Un ciclo for básico. Valor inicial de uno e incremento de uno en uno.

② El break causa que el for cicle una sola vez. Esto únicamente ejecuta una instrucción.

$$f(n) = 1 + 1 + n * 1 = n + 2$$

$$O(n+2) = n + 2$$

$$f(n) = O(n)$$

$$c=2$$

$$n+2 \leq 2n$$

$$-n \leq -2$$

$$n \geq 2$$

Ejercicio 3

```
void function(int n) {
```

```
    int i, j;
```

```
    for (i = 1; i <= n / 3; i++) {
```

```
        for (j = 1; j <= n; j += 4) {
```

```
            printf("%s avance\n");
```

3

3

3

1

n / 3

① n / 4

1

① El incremento es de cuatro en cuatro.

Comparado con el incremento de uno

en uno, un ciclo exitoso avanza cuatro

veces este incremento básico. Por tanto, la

instrucción interna se ejecutara n / 4 veces.

$$f(n) = 1 + \frac{n}{3} * \frac{1}{4} * 4 = \frac{n^2}{12} + 1$$

$$O\left(\frac{n^2}{12} + 1\right) = n^2 \quad c = 1$$

$$\frac{n^2}{12} + 1 \leq n^2 \quad \frac{n^2}{12} - n^2 \leq -1 \quad n^2 \left(\frac{1}{12} - 1\right) \leq -1$$

$$n^2 \left(-\frac{11}{12}\right) \leq -1 \quad n^2 \left(\frac{11}{12}\right) \geq 1 \quad n^2 \geq \frac{12}{11}$$

$$n^2 - \sqrt{\frac{12}{11}} = 2 - \sqrt{\frac{3}{11}}$$

Ejercicio 4.

Ejemplo de Linear Search

```
int search(int arr[], int size, int x) {  
    int i = 0;  
    for (i = 0; i < size; i++) {  
        if (arr[i] == x)  
            return i;  
    }  
    return -1;
```

Existen dos casos para este algoritmo de búsqueda.

Caso 1: el elemento x puede estar en alguno de N índices desde 0 hasta $N-1$.

Caso 2: El elemento x no está en la lista.

Además, sabemos que si el elemento x está en el índice k , en total se realizarán $k+1$ comparaciones.

El total de comparaciones de todos los subcasos de caso 1 se puede definir como las comparaciones para cada índice.

Es decir, comparaciones para el índice 0 más comparaciones para el índice 1 hasta comparaciones en el índice $N-1$.

$$1+2+3\dots+N = \frac{N*(N+1)}{2}$$

Por otro lado, si el elemento no se encuentra en la lista, el algoritmo realizará N comparaciones. Este es el único subcaso del caso 2.

El número promedio de comparaciones es la suma de todos los subcasos del caso 1 y 2, dividido el número total de casos.

$$\left(\frac{N*(N+1)}{2} + 1 \right) \div (N+1) = \frac{N}{2} + \frac{N}{N+1}$$

$$O\left(\frac{N}{2} + \frac{N}{N+1}\right) = N \quad O(N)$$

El mejor caso ocurre si el elemento buscado está en el índice 0

Complejidad del mejor caso: $O(1)$

Complejidad del peor caso: $O(n)$

Complejidad del caso promedio: $O(n)$

Ejercicio 5.

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

Falso

$f(n) = \Theta(g(n))$ indica que existen M_1 , c_1, c_2 tal que $c_1|g(x)| \leq |f(x)| \leq c_2|g(x)|$
para toda $x > M_1$.

Respectivamente, $g(n) = \Theta(h(n))$; M_2, c_3, c_4
tal que $c_3|h(x)| \leq |g(x)| \leq c_4|h(x)|$ para
toda $x > M_2$.

Por tanto es posible que exista una
función $b(n)$ que no este contenida
dentro de múltiplos de $f(n)$.

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

Por definición de notación O . si
 $f(n) = O(g(n))$, existe un M tal que
 $|f(x)| \leq c_1|g(x)|$ para toda $x > M$.

Si $g(n) = O(h(n))$, existe un M tal que
 $|g(x)| \leq c_2|h(x)|$ para toda $x > M$.

Por tanto, es posible que $h(n) \leq f(n)$ y
la afirmación no se cumple siempre.

$$c) f(n) = \Theta(n^2)$$

def A(n):

atuple = tuple(range(0, n))

S = set()

for i in range (0, n):

 for j in range (i+1, n):

 S.add(atuple[i:j])

n

n - i - 1

n

$$f(n) = n(n-i-1)*2 = 2n^2 - 2ni - 2n$$

$$\mathcal{O}(2n^2 - 2ni - 2n) = n^2$$

$$\Omega(2n^2 - 2ni - 2n) = n^2$$

Verdaderd