



Instituto Politécnico Nacional  
Escuela Superior de Cómputo



❖ UNIDAD DE APRENDIZAJE: ANÁLISIS DE ALGORITMOS

❖ PROFESOR: EDGARDO ADRIÁN FRANCO MARTÍNEZ

❖ ALUMNO:

BARRERA PÉREZ CARLOS TONATIHU



❖ Ejercicio 4: Análisis de algoritmos no recursivos

❖ GRUPO: 3CM3

## Determine la cota $O()$

### Problema 1

Ejercicio 1: Realizando el análisis por bloques

```
for(i=1; i<n; i++) {  
    for(j=0; j<n-1; j++) {  
        temp = A[j];  
        A[j] = A[j+1];  
        A[j+1] = temp;  
    }  
}
```

Se repite  $n-1$  veces,  $i$  veces =  $n$

Se repite  $n-1$  veces

$O(1)$

$O(n)$

$O(n^2)$

La función es de orden  $O(n^2)$

- Para el primer bloque todas las instrucciones son de orden  $O(1)$  por lo que el  $O(\max(g_1(n), g_2(n), g_3(n))) = O(1)$
- Para el for interno que se repite  $n-1$  veces el orden será  $O((n-1)1) = O(n-1) = O(n)$
- En el for exterior ocurre lo mismo  $O(n-1(n)) = O(n^2-n) = O(n^2)$

La cota es  $O(n^2)$

### Problema 2

Ejercicio 2: Realizando el análisis por bloques

```
polinomio = 0;  
for(i=0; i<=n; i++)  
    polinomio = polinomio * z + A[n-i];
```

$O(1)$

$O(1)$

$O(n)$

$O(\max(n, 1)) = O(n)$

La función es de orden  $O(n)$

- La instrucción interna es de orden  $O(1)$
- El for se ejecuta  $n+1$  veces por lo que su complejidad es  $O((n+1)(1)) = O(n+1) = O(n)$
- La asignación es de orden  $O(1)$
- Finalmente,  $O(\max(n, 1)) = O(n)$  debido a que el for se repite más de una vez

La cota es  $O(n)$

### Problema 3

Ejercicio 3: Realizando el análisis por bloques

```
for i=1 to n do
  for j=1 to n do
    C[i,j]=0; } O(1)
    for k=1 to n do
      C[i,j]=C[i,j]+A[i,k]*B[k,j]; } O(1) } O(n) } O(n^2) } O(n^3)
```

- La instrucción más interna es  $O(1)$
  - El siguiente for se ejecuta  $n$  veces, de esta forma tenemos  $O(n(1)) = O(n)$
  - La siguiente asignación es  $O(1)$
  - Después, tomamos el máximo de estas dos últimas operaciones  $O(\max(n, 1)) = O(n)$
  - El siguiente for se ejecuta  $n$  veces, por lo que considerando esto y lo anterior  $O(n(n)) = O(n^2)$
  - Lo mismo ocurre en el siguiente for que se ejecuta  $n$  veces  $O(n(n^2)) = O(n^3)$
- ∴ La función es de orden  $O(n^3)$

La cota es  $O(n^3)$

### Problema 4

Ejercicio 4: Realizando el análisis por bloques

```
anterior=1; } O(1)
actual=1; } O(1)
while(n>2) {
  aux=anterior+actual; } O(1)
  anterior=actual; } O(1)
  actual=aux; } O(1)
  n=n-1; } O(1)
}
```

La función es de orden  $O(n)$

- Las instrucciones internas son  $O(1)$
- El for se ejecuta  $n-2$  veces, de esta forma  $O(1(n)) = O(n)$
- El resto de asignaciones son  $O(1)$
- Finalmente, el orden de la función es  $O(\max(n, 1)) = O(n)$

La cota es  $O(n)$



## Problema 5

Ejercicio 5: Realizando el análisis por bloques

```
for (i = n-1, j = 0; i >= 0; i--, j++) { O(n)
    s2[j] = s[i]; } O(1)
```

La función es de orden  $O(n)$

```
for (i = 0; i < n; i++) { O(n)
    s[i] = s2[i]; } O(1)
```

El primer for se ejecuta n veces por lo que es  $O(n)$

El segundo for también es  $O(n)$

Después,  $O(\max(n, n)) = O(n)$

La cota es  $O(n)$

## Problema 9

Ejercicio 9: Realizando el análisis por bloques

```
func Producto2Mayores(A, n)
    if (A[1] > A[2])
        mayor1 = A[1];
        mayor2 = A[2]; } O(1)
    else
        mayor1 = A[2];
        mayor2 = A[1]; } O(1)
    i = 3; } O(1)
    while (i <= n)
        if (A[i] > mayor1)
            mayor2 = mayor1;
            mayor1 = A[i]; } O(1)
        else if (A[i] > mayor2)
            mayor2 = A[i]; } O(1)
        i = i + 1; } O(1)
    return mayor1 * mayor2; } O(1)
fin
```

Todas las instrucciones antes y después del while son de orden constante  $O(1)$

Dentro del while tenemos un orden constante  $O(1)$

El ciclo itera  $n-2$  veces por lo que es  $O(n(1)) = O(n)$

Finalmente,  $O(\max(1, n, 1)) = O(n)$

La cota es  $O(n)$

### Problema 10

Ejercicio 10: Realizando el análisis por bloques

```
func OrdenamientoIntercambio(a,n)
  for(i=0; i<n-1; i++)
    for(j=i+1; j<n; j++)
      if(a[j] < a[i]) {
        temp = a[i];
        a[i] = a[j];
        a[j] = temp;
      }
  fin
```

La función es  $O(n^2)$

$O(1)$   $O(n)$   $O(n^2)$

- El bloque interno se es  $O(1)$
- El ciclo interno se ejecuta un máximo de  $n-1$  veces por lo que es  $O(n)$
- El for externo también se ejecuta  $n-1$  veces por lo que  $O(n(n)) = O(n^2)$

La cota es  $O(n^2)$

### Problema 11

Ejercicio 11: Realizando el análisis por bloques

```
func MaximoComunDivisor(m,n) {
  a = max(n,m);
  b = min(n,m);
  residuo = 1;
  mientras (residuo > 0) {
    residuo = a mod b;
    a = b;
    b = residuo;
  }
  MaximoComunDivisor = a;
  return MaximoComunDivisor;
}
```

$O(1)$   $O(1)$   $O(n)$   $O(n)$

- Consideremos a las funciones  $\max$  &  $\min$  como  $O(1)$
- Si  $m$  &  $n$  son coprimos sólo tendrían al uno como divisor y requerirá el número más pequeño entre  $n$  &  $m$ , suponiendo que  $n < m$  la complejidad es  $O(n)$

La cota es  $O(n)$



## Problema 12

### Ejercicio 12: Realizando el análisis por bloques

Procedimiento Burbuja Optimizada ( $A, n$ )

cambios = "si" }  $O(1)$

$i = 0$

Mientras  $i < n-1$  && cambios != "No" hacer

cambios = "No" }  $O(1)$

para  $j = 0$  hasta  $(n-2) - i$  hacer

si  $(A[i] < A[j])$  hacer

aux =  $A[j]$ ;

$A[j] = A[i]$ ;

$A[i] = aux$ ;

cambios = "si"

Fin si

Fin para

$i = i + 1$ ; }  $O(1)$

Fin mientras

Fin Procedimiento

- Las instrucciones antes del mientras son de orden  $O(1)$
- El conjunto de instrucciones dentro de los dos ciclos es  $O(1)$
- El ciclo interno se ejecuta a lo más  $n-1$  veces por lo que es  $O(n)$
- Evaluando las instrucciones dentro del primer ciclo  $O(\max(1, n, 1)) = O(n)$
- Ya que el ciclo externo se ejecuta  $n-1$  veces en el peor caso tenemos  $O(n(n)) = O(n^2)$
- Finalmente,  $O(\max(1, n^2)) = O(n^2)$

La cota es  $O(n^2)$

### Problema 13

#### • Ejercicio 13: Realizando el análisis por bloques

Procedimiento Burbuja Simple ( $A, n$ )

para  $i=0$  hasta  $n-2$  hacer

para  $j=0$  hasta  $(n-2)-i$  hacer

si  $(A[j] > A[j+1])$  entonces

aux =  $A[j]$ ;

$A[j] = A[j+1]$ ;

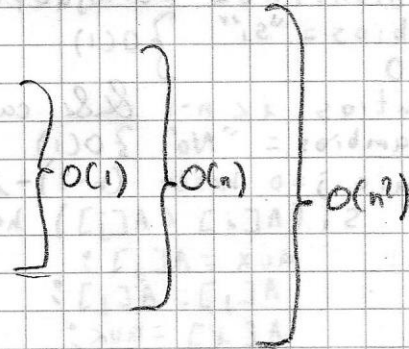
$A[j+1] = aux$ ;

fin si

fin para

fin para

fin procedimiento



— Las instrucciones dentro de los dos ciclos son de orden  $O(1)$

— El ciclo interno se ejecuta un máximo de  $n-1$  veces, así,  $O(n(1)) = O(n)$

— El siguiente ciclo también se ejecuta  $n-1$  veces por lo que al final tenemos  $O(n(n)) = O(n^2)$

∴ La función es de orden  $O(n^2)$

La cota es  $O(n^2)$

Problema 14

- Ejercicio 14.

```
Procedimiento Ordena(a,b,c)
  if(a > b)
    if(a > c)
      if(b > c)
        salida(a,b,c); {O(1)}
      else
        salida(a,c,b); {O(1)}
    else
      salida(c,a,b); {O(1)}
  else
    if(b > c)
      if(a > c)
        salida(b,a,c); {O(1)}
      else
        salida(b,c,a); {O(1)}
    else
      salida(c,b,a); {O(1)}
}
```

La función es constante  $O(1)$  //

- Debido a que no existen ciclos y sólo tenemos comparaciones la función es de orden constante  $O(1)$  o en su defecto su complejidad es la complejidad de la función salida().

La cota es  $O(1)$



## Problema 15

• Procedimiento Selección( $A, n$ )  
  para  $k=0$  hasta  $n-2$  hacer  
     $p=k$   $\{O(1)$   
    para  $i=k+1$  hasta  $n-1$  hacer  
      si  $A[i] < A[p]$  entonces  $\{O(1)$   $\left. \begin{array}{l} \} O(n) \end{array} \right\} O(n^2)$   
       $p=i$   
    fin para  
     $\text{temp} = A[p]$   $\{O(1)$   
     $A[p] = A[k]$   
     $A[k] = \text{temp}$   
  fin para  
fin procedimiento

- La instrucción dentro de los dos ciclos en el peor de los casos es  $O(1)$
- El ciclo anidado se ejecuta a lo más  $n$  veces por lo que  $O(n(1)) = O(n)$
- El resto de instrucciones dentro del ciclo externo es  $O(1)$
- El ciclo externo se ejecuta  $n-1$  veces por lo que la complejidad final es  $O(n(\max(1, n, 1))) = O(n(n)) = O(n^2)$

La cota es  $O(n^2)$

## Conclusiones

Los resultados obtenidos en el ejercicio 2, a pesar de que el análisis que se realizó fue más extenso que el realizado aquí, son muy similares a los que se obtuvieron en esta tarea, es evidente que los que tienen mayor similitud son los del caso medio y peor caso. Por lo que optar por un análisis como el realizado en este ejercicio es una buena idea si se quiere obtener un resultado rápido y preciso.