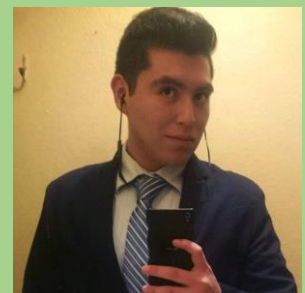


Análisis de algoritmos



```
0 1 0 1 0
1 0 1 0 1
0 1 0 1 0
1 0 1 0 1
0 1 0 1 0
1 0 1 0 1
0 1 0 1 0
1 0 1 0 1
0 1 0 1 0
1 0 1 0 1
0 1 0 1 0
1 0 1 0 1
0 1 0 1 0
```



Vargas Romero Erick Efraín

Prof. Franco Martínez Edgardo Adrián

Grupo 3CM2

Tarea 02: Complejidad temporal y análisis
de casos

Fecha de entrega 06/09/2018

Ejercicios: Complejidad de los algoritmos

- A. Para los siguientes 5 algoritmos determine la función de complejidad temporal y espacial en términos de n . *Considere las operaciones de: asignación, aritméticas, condicionales y saltos implícitos.

Ejercicios:

1) $\text{for } (i=1; i \leq n; i++)$
 $\text{for } (j=0; j < n-1; j++) \{$
 $\text{temp} = A[j];$ ①
 $A[j] = A[j+1];$ ②
 $A[j+1] = \text{temp};$ ②
 $\}$

$f_e(n) = 3 + n$
 $f_c(n) = 1 + n + n-1 + n + (n-1)(1 + n + n-1 + n + (n-1)(5))$
 $= 3n + (n-1)(3n + 5n - 5)$
 $= 3n + (n-1)(8n - 5)$
 $= 3n + 8n^2 - 5n - 8n + 5$
 $= 8n^2 - 10n + 5$

2) $\text{polinomio} = 0;$ ①
 $\text{for } (i=0; i \leq n; i++)$
 $\text{polinomio} = \text{polinomio} * Z + A[n-i];$

$f_e(n) = 3 + n$
 $f_c(n) = 1 + 1 + n + 2 + n + 2 + n + 1 + (n+1)(4)$
 $= 3n + 7 + 4n + 4$
 $= 7n + 11$

3) $\text{for } i=1 \text{ to } n \text{ do}$
 $\text{for } j=1 \text{ to } n \text{ do}$
 $C[i, j] = 0;$ ①
 $\text{for } k=1 \text{ to } n \text{ do}$
 $C[i, j] = C[i, j] + A[i, k] * B[k, j]$

$f_e(n) = 3 + 3n^2$

$f_c(n) = 1 + n + n + n-1 + (n-1)(1 + n + n + n-1 + (n-1)(1 + 1 + n + n + n-1 + (n-1)(3)))$
 $= 3n + (n-1)(3n + (n-1)(3n + 1 + 3n - 3))$
 $= 3n + (n-1)(3n + (n-1)(6n - 2))$
 $= 3n + (n-1)(3n + 6n^2 - 2n - 6n + 2)$
 $= 3n + (n-1)(6n^2 - 5n + 2)$
 $= 3n + 6n^3 - 5n^2 + 2n - 6n^2 + 5n - 2 = 6n^3 - 11n^2 + 10n - 2$

4) anterior = 1; ①

actual = 1; ①

while ($n > 2$) {

aux = anterior + actual; ②
 anterior = actual; ① ①
 actual = aux; ① ①
 n = n - 1; ②
 }

$$f_e(n) = 3$$

$$\begin{aligned}
 f_c(n) &= 1 + 1 + \overbrace{n-1}^{\text{operaciones}} + \overbrace{n-1}^{\text{saltos}} + (n-2)(6) \\
 &= 2n + 6n - 12 - 10 \\
 &= 8n - 12
 \end{aligned}$$

5) for($i = n-1$; $j = 0$; $i > 0$; $i--$, $j++$) $f_e(n) = 2 + 2n$

n { s2[j] = s[i]; } ①

for($i = 0$; $i < n$; $i++$)

{ s[i] = s2[i]; } ①

$$\begin{aligned}
 f_c(n) &= 2 + 1 + \overbrace{n+1} + \overbrace{n+1} + \overbrace{2n} + \overbrace{n(1)} \\
 &\quad + \overbrace{1} + \overbrace{n+1} + \overbrace{n+1} + \overbrace{n} + \overbrace{n(1)} \\
 &= 9n + 8
 \end{aligned}$$

- B. Para los siguientes 3 algoritmos determine el número de veces que se imprime la palabra "Algoritmos". Determine una función lo más cercana a su comportamiento para cualquier n y compruébela codificando los tres algoritmos (Adjuntar códigos). De una tabla de comparación de sus pruebas para n 's igual a 10, 100, 1000, 5000 y 100000 y demostrar lo que la función encontrada determina será el número de impresiones.

6) for($i=10$; $i < 5*n$; $i*=2$)

printf("Algoritmos\n");

• la condición $i < 5*n$ será estática

• $i*=2$ tiene un crecimiento bastante rápido $(10)2$, $(10)2 \cdot 2$, $(10)2 \cdot 2 \cdot 2$
de forma general $i*=2 = x^2 = n$ despejando

$$x^2 = n;$$

$$\log_2(x^2) = \log_2(n)$$

$$x = \log_2(n)$$

∴ El comportamiento es $\lfloor \log_2(n) \rfloor$

```
#include <stdio.h>
#include <stdlib.h>
int main (int argc, char *argv[]){
    int n, i;
    printf ("Introduce una N\n");
    scanf("%d", &n);
    for (i=10; i<n*5; i*=2){
        printf("\nAlgoritmos\n");
    }
}
```

N	Impresiones reales	Según la función
10	3	3
100	6	6
1,000	9	9
5,000	12	12
10,000	13	13

```

7) for(j=n; j>1; j/=2) { ... (A)
    if(j < (n/2)) { ... (B)
        for(i=0; i<n; i+=2) { ... (C)
            printf("Algoritmos\n");
        }
    }
}

```

- En (A) el comportamiento es decreciente siendo aproximadamente $X^2 = n$ es decir $\log_2(n)$
- En (B) tenemos "dos caminos" si la condición se cumple entonces se realizan más operaciones, la que ocurren en (C)
- En (C) tenemos un crecimiento lineal con saltos de 2 en 2 en este caso se ejecutará el código interno $\frac{n}{2}$
- La función es $\log_2(n) \cdot \frac{n}{2} = \left\lfloor \frac{n(\log_2(n) - 2)}{2} \right\rfloor$

```

#include <stdio.h>
#include <stdlib.h>
int main (int argc, char *argv[]){
    int n, i, j;
    printf ("Inroduce una N\n");
    scanf("%d", &n);
    for (j=n; j>1; j/=2){
        if (j<(n/2)){
            for(i=0; i<n; i+=2){
                printf("\nAlgoritmos\n");
            }
        }
    }
}

```

N	Impresiones reales	Según la función
10	5	5
100	10	100
1,000	3,500	3,500
5,000	25,000	25,000
10,000	55,000	55,000

```
8) i = n
while (i >= 0) { ... (A)
    for (j = n; i < j; i -= 2, j /= 2) { ... (B)
        printf("Algoritmos\n");
    }
}
```

Por la condición $i < j$ tendremos siempre la condición en falsa, desde la primera vez que sea evaluada, nunca imprimiremos la palabra algoritmos

```
#include <stdio.h>
#include <stdlib.h>
int main (int argc, char *argv[]){
    int n, i, j;
    printf ("Introduce una N\n");
    scanf("%d", &n);
    i=n;
    while(i>=0){
        for (j=n; i<j; i-=2, j/=2){
            printf("\nAlgoritmos\n");
            k++;
        }
    }
    printf("%d\n", k);
    return 0;
}
```

N	Impresiones reales	Según la función
10	0	0
100	0	0
1,000	0	0
5,000	0	0
10,000	0	0

- C. Para los siguientes algoritmos determine las funciones de complejidad temporal, para el mejor caso, peor caso y caso medio. Indique cual(es) son las condiciones (instancia de entrada) del peor caso y cual(es) la del mejor caso.

9) func Producto2Mayores(A, n)

```

    if (A[1] > A[2])
        mayor1 = A[1]; ①
        mayor2 = A[2]; ①
    else
        mayor1 = A[2]; ①
        mayor2 = A[1]; ①
    i = 3;
    while (i <= n)
        if (A[i] > mayor1)
            mayor2 = mayor1;
            mayor1 = A[i];
        else if (A[i] > mayor2)
            mayor2 = A[i];
        i = i + 1;
    return mayor1 * mayor2;
fin

```

2 comparaciones
3 op

Análisis: Primeramente debemos percatarnos que este código inicialmente busca los dos números más grandes del arreglo, por tanto.

- Mejor caso: Si en las primeras posiciones están los números más grandes no se realizan las operaciones de asignación dentro del ciclo while

$$\therefore 3 + (n-2)(2) = 2n - 1$$

- Peor caso: Si los mayores son los últimos del arreglo

$$\therefore 3 + 3(n-2) = 3n - 3$$

- Caso medio: Consideramos las probabilidades de que ocurran los eventos al comparar (suponemos que tienen la misma probabilidad)

$$\therefore \frac{1}{2}(2n-1) + \frac{1}{2}(3n-3) = \frac{1}{2}(5n-4)$$

10) func OrdenamientoIntercambio(a, n)

```

    for (i = 0; i < n-1; i++)
        for (int j = i+1; j < n; j++)
            if (a[j] < a[i])
                temp = a[i];
                a[i] = a[j];
                a[j] = temp;
    }
fin

```

Peor caso: Que el arreglo se encuentre ordenado

$$f_{\text{wc}}(n) = \sum_{i=0}^{n-1} i$$

Peor caso: Se encuentran en orden descendente

$$f_{\text{wc}}(n) = \sum_{i=0}^{n-1} i$$

- Caso medio:

$$f_{\text{cm}}(n) = \frac{1}{2} \sum_{i=0}^{n-1} i$$

11) Func MaximoComunDivisor(m, n)

$a = \max(n, m);$

$b = \min(n, m);$

residuo = 1;

mientras (residuo > 0) {

elementos de fibo hasta a
 $\left\{ \begin{array}{l} \text{residuo} = a \bmod b; \textcircled{1} \\ a = b; \\ b = \text{residuo} \end{array} \right.$

MaximoComunDivisor = a;

return MaximoComunDivisor;

• Mejor caso: Cuando a sea factor de b es decir $a \bmod b = 0$

$$\therefore P_{tmc}(n) = 1$$

• Peor caso: Que a y b sean elementos de la sucesión de los Fibonacci, si esto ocurre

$$\therefore f_{tpe}(n) = \# \text{elementos de fibonacci hasta que sea igual con } a$$

• Caso medio:

$$\therefore f_{tem}(n) = \# \text{elementos de fibbo hasta que sea igual con } a + 1$$

12) Procedimiento BurbujaOptimizado(A, n)

cambios = "No";

i = 0

Mientras (i < n-1) de cambios != "No" hacer

cambios = "No";

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;

Fin mientras

Fin procedimiento

• Operaciones básicas: Comparación de los elementos de A[k] y asignación hacia A[k]

• Mejor caso: Que todos los A[i] sean mayores que A[j] es decir que A esté ordenado

$$\therefore f_{tmc}(n) = \sum_{i=0}^{n-1} (3n - 6 - 3i)$$

• Peor caso: Cuando A[i] < A[j] se realizan todas las operaciones

$$\begin{aligned} \therefore f_{tpe}(n) &= (n-1)(n-2-i)(3) \\ &= (n-1)(3n-6-3i) \\ &= 3n^2-6n-3ni-3n+6-3i \\ &= 3n^2-9n-3(ni-i+2) \end{aligned}$$

pero por la "i" obtenemos

$$\sum_{i=0}^{n-1} (3n^2-9n-3(ni-i+2))$$

• Medio: Supongamos que todos los casos tienen la misma probabilidad

$$\frac{1}{2} \left(\sum_{i=0}^{n-1} (3n^2-9n-3ni-6i+12) \right)$$

13) Procedimiento BurbujaSimple(A, n)

```

para i = 0 hasta n-2 hacer
    para j = 0 hasta (n-2)-i hacer
        si (A[j] > A[j+1]) entonces ①
            aux = A[j]
            A[j] = A[j+1] ②
            A[j+1] = aux ③
        fin si
    fin para
fin para
fin procedimiento
    
```

* Operaciones básicas: comparación entre elementos de A[k], asignaciones a A[k]

Mejor caso: A[k] este ordenado

$$f_{bmc}(n) = \sum_{i=0}^{n-2} (n-2-i)(1) //$$

Peor caso: que A[j] > A[j+1]

$$\therefore f_{bpc}(n) = \sum_{i=0}^{n-2} (n-2-i)(3)$$

Caso medio: Consideremos probabilidades iguales

$$\frac{1}{2} \sum_{i=0}^{n-2} \{4(n-2-i)\}$$

14) Procedimiento Ordena(a, b, c) {

```

    If(a > b) { ①
        If(a > c) { ②
            If(b > c) { ③
                salida(a, b, c);
            } else {
                salida(a, c, b);
            }
        }
    } else {
        salida(c, a, b);
    }
}
    
```

• Mejor caso: Si los números cumplen

$$c > a > b \text{ ó } c > b > a \therefore f_{bmc}(n) = 2$$

• Peor caso: Si los números cumplen

$a > b > c$ ó $a > c > b$ ó $b > a > c$ ó $b > c > a$ es decir que se realicen todas las condiciones

$$\therefore f_{bpc}(n) = 3$$

• Caso medio:

$$f_{bcm}(n) = \frac{5}{2}$$


```

if (b > c) { ①
    if (a > c) { ①
        salida(b, a, c);
    } else {
        salida(b, c, a);
    }
} else {
    salida(c, b, a);
}
}

```

* Operaciones básicas, comparaciones entre a, b, c

15) Procedimiento Selección (A, n)

```

para k = 0 hasta n-2 hacer
    p = k
    para i = k+1 hasta n-1 hacer
        si n-k-1 A[i] < A[p] entonces ①
            p = i
        fin si
    fin para
    temp = A[p] ①
    A[p] = A[k] ①
    A[k] = temp ①
fin para
fin procedimiento

```

(Diagrama de flujo implícito: El bucle interno para i se repite n-k-1 veces para cada k, y el bucle externo para k se repite n-1 veces en total.)

• Mejor caso: Si $A[i] > A[p]$
es decir A está ordenado

$$\therefore f_{mc}(n) = 2n - 1 - k(n-2)$$

• Peor caso: Si $A[i] < A[p]$
es decir A está de forma descendente

$$\therefore f_{pc}(n) = 2n^2 - 4n - k(n-2)$$

• Caso medio: Considerando
múltiples posibilidades

$$\begin{aligned}
 f_{cm}(n) &= \frac{1}{2} (2n - 1 - k(n-2) + 2n^2 - 4n - k(n-2)) \\
 &= \frac{1}{2} (2n^2 - 2n - 4 - 2k(n-2)) \\
 &= \underline{\underline{n^2 - n - 2 - k(n-2)}}
 \end{aligned}$$