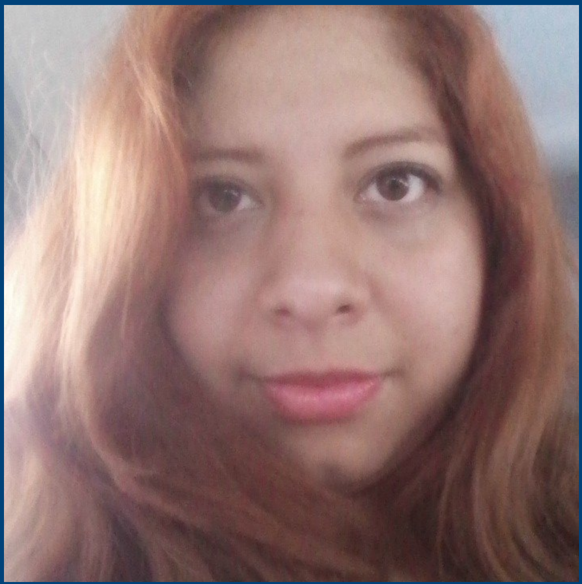
ESCOM-IPN

Ejercicio 2
Análisis de Complejidad

Análisis de Algoritmos

Laura Andrea Morales López

Marzo 2018



Índice

1. Complejidad temporal y espacial

1.1. Algoritmo 1:

```
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;
    }
}
```

Tenemos 2 ciclos que analizar.

El for que se encuentra mas dentro tiene 5 instrucciones, 2 asignaciones y 2 operaciones.

Dentro del for tenemos una resta y una suma.

Y el n sería $n - 2$ pues hay una restriccion de que debe ser menor que $n - 1$.

Entonces, tenemos un bloque de 5 instrucciones que se repiten $n - 2$ veces y luego una comparacion una resta y una suma y asignación dentro de la definicion que se repiten $n - 2$ veces, un salto que se repite $n - 2$, una asignacion al inicio una ultima comparación, una ultima resta y un salto cuando en for no entra en el ciclo.

Tenemos lo siguiente:

$5(n - 2) + 4(n - 2) + (n - 2) + 4$

$10(n - 2) + 4$

$10n - 6$

Después tenemos el siguiente for que es una asignacion al inicio una comparación $n - 1$ veces, una asignación, suma y salto implicito $n - 1$veces y una comparación final con salto final. Para este for tenemos la complejidad.

$4(n - 1) + 3$

Y colocamos la complejidad del anterior, entonces:

$(10n - 6 + 4)(n - 1) + 3$

La complejidad de este algoritmo es $10n^2 - 12n + 5$

La complejidad espacial es el tamaño del arreglo 2 iteradores y la variable temp.

La complejidad espacial es: $n + 3$

1.2. Algoritmo 2:

```
polinomio = 0;
```

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

Tenemos una asignación al principio de este algoritmo y tenemos 3 operaciones detro del for con una asignación.

En el for tenemos una asignacion inicial, una comparación final y un salto.

Tenemos una comparación, una suma, una asignación y un salto $n + 1$ veces

$8(n + 1) + 4$

La complejidad de este algoritmo es $8n + 12$

La complejidad espacial sería el tamaño del Arreglo, el iterador y la variable.

La complejidad espacial es:$n + 3$

1.3. Algoritmo 3:

```
for i = 1 to n do
    for j = 1 to n do
        C[i,j] = 0;
        for k = 1 to n do
C[i,j] = C[i,j] + A[i,k]*B[k,j];
```

El for más adentro:

Tiene 1 asignación y 2 operaciones. Tiene en su definicion tiene una comparación, una asignacion y suma y un salto. Todo esto n veces.

Tiene una asignación inicial, una compaacion final y un salto final.

Su complejidad es $7n + 3$

Para el siguiente for tenemos:

Una asignación,una comparación, una suma, una asignación y un salto n veces.

Tiene una asignación inicial, una compaacion final y un salto final.

Entonces colocandolo con el anterior tenemos:

$(7n + 3 + 5)(n) + 3$

$7n^2 + 8n + 3$

Finalmente para el ultimo ciclo tenemos:

Una comparación, una suma, una asignación y un salto n veces.

Tiene una asignación inicial, una compaacion final y un salto final.

Con el demás algoritmo tenemos

$(7n^2 + 8n + 3 + 4)(n) + 3$

La complejidad de este algoritmo es:$7n^3 + 8n^2 + 7n + 3$

La complejidad espacial esta dada por las 3 matrices n^2 y los 3 iteradores= $3n^2 + 3$

1.4. Algoritmo 4:

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

Para este algoritmo tenemos un while:

Dentro del while tenemos 6 instrucciones que se repetiran $n - 2$ veces.

Ademas tenemos el salto y la comparacion que igual serán realizadas $n - 2$ veces.

Dos asignaciones iniciales.

Tenemos $6(n - 2) + 2(n - 2) + 2$

La complejidad de este algoritmo es $8n - 14$. La complejidad espacial s solo las tres variables=3

1.5. Algoritmo 5:

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

En este algoritmo tenemos dentro del for 2 asignaciones y una operación inicial ademas del salto final y la comparacion final.

Una comparacion, tres asignaciones,y 2 operaciones, además del salto esto n veces.

La complejidadsería de $7n + 5$.

La del segundo for seria una asignacion inicial, un salto final y una comparacion final.

Una asignación, un salto,una comparacion, una suma y una asignacion n veces.Entonces es $= 5n + 3$

Entonces nuestra complejidad total es $= 12n + 8$

La complejidad espacial es el tamaño de los 2 arreglos mas los 2 iteradores. $= 2n + 2$

2. Impresiones

2.1. Algoritmo 6:

```
for(int i = 10; i <n*5; i *= 2){
    printf("Algoritmos\n");
}
```

Notemos que i no puede ser 0, entonces sabemos que cambia respecto al parameto inicial y la iteración es una multiplicación. Se comporta como una progresión geometrica donde $i = 10(2)^{k-1}$

Entonces debe mantenerse en medio de los limites $10 <= 10(2)^{k-1} <= 5n$ Tenemos que $1 <= k <= log_2\frac{5n}{10} + 1$

Asi tenemos que $= log_2\frac{5n}{10} + 1$ será el numero de prints que tendremos.

Para n=10 tendremos un número de impresiones =3.

Para n=100 tendremos un número de impresiones =6.

Para n=1000 tendremos un número de impresiones =9.

Para n=5000 tendremos un número de impresiones =12.

Para n=100000 tendremos un número de impresiones =16.

2.1.1. Pruebas

```
#include <stdio.h>

int main()
{
    int n, counter=0;
    scanf("%d",&n);

        for(int i = 10; i <n*5; i *= 2){
        printf("Algoritmos\n");
        counter++;
    }
    printf("Counter= %d\n",counter );
    return 0;
}
```

```
lalaandrea10@lalaandrea10:~/Escritorio$ ./a.out
10
Algoritmos
Algoritmos
Algoritmos
Counter= 3
lalaandrea10@lalaandrea10:~/Escritorio$ ./a.out
100
Algoritmos
Algoritmos
Algoritmos
Algoritmos
Algoritmos
Algoritmos
Counter= 6
lalaandrea10@lalaandrea10:~/Escritorio$ ./a.out
1000
Algoritmos
Algoritmos
Algoritmos
Algoritmos
Algoritmos
Algoritmos
Algoritmos
Algoritmos
Algoritmos
Counter= 9
```

```
lalaandrea10@lalaandrea10:~/Escritorio$ ./a.out
5000
Algoritmos
Algoritmos
Algoritmos
Algoritmos
Algoritmos
Algoritmos
Algoritmos
Algoritmos
Algoritmos
Algoritmos
Algoritmos
Algoritmos
Counter= 12
lalaandrea10@lalaandrea10:~/Escritorio$ ./a.out
100000
Algoritmos
Algoritmos
Algoritmos
Algoritmos
Algoritmos
Algoritmos
Algoritmos
Algoritmos
Algoritmos
Algoritmos
Algoritmos
Algoritmos
Algoritmos
Algoritmos
Algoritmos
Algoritmos
Counter= 16
lalaandrea10@lalaandrea10:~/Escritorio$
```

2.2. Algoritmo 7:

```
for(int j = n; j > 1; j /= 2){
        if(j < n / 2){
            for(int i = 0; i < n; i += 2){
                printf("\"Algoritmos\"\n");

            }
        }
}
```

El for interno se se ejecuta $\frac{n}{2}$ veces.

El externo podemos escribirlo como $for(intj = 1; j < n/2; j* = 2$ Asi tenemos que:

Se comporta como una progresión geometrica donde $i = 2(2)^{k-1}$

Entonces debe mantenerse en medio de los limites $2 <= 2(2)^{k-1} <= n/2$ Tenemos que $1 <= k <= log_2 \frac{n/2}{2} + 1$

Asi tenemos que $= log_2 \frac{n/2}{2} + 1$ será el numero de prints que tendremos.

Todo esto por n/2=

$= \frac{n}{2} log_2 \frac{n/2}{2} + 1.$

Para n=10 tendremos un número de impresiones =5.

Para n=100 tendremos un número de impresiones =200.

Para n=1000 tendremos un número de impresiones =3500.

Para n=5000 tendremos un número de impresiones =25000.

Para n=100000 tendremos un número de impresiones =700000.

2.2.1. Pruebas

```
#include <stdio.h>

int main()
{
    int n, counter=0;
    scanf("%d",&n);

        for(int j = n; j > 1; j /= 2){
        if(j < n / 2){
            for(int i = 0; i < n; i += 2){
```

```
                    //printf("\"Algoritmos\"\n");
                    counter++;
                }
            }
}
    printf("Counter= %d\n",counter );
    return 0;
}
```

2.3. Algoritmo 8:

```
while(i >= 0){
        for(int j = n; i < j; i -= 2, j /= 2){
            printf("\"Algoritmos\"\n");

        }
}
```

La respuesta en ninguna pues primero declaramos i=n y despues en el inicio del for j=n pero la condicino es que i<j pero esto nunca es cierto pues son iguales, entonces el while se seguira cumpliendo pero nunca entra.

3. Caso mejor, peor y medio.

3.1. Algoritmo 9:

```
func Producto2Mayores(A,n)
    if(A[1] > A[2])
        mayor1 = A[1];
```

```
        mayor2 = A[2];
    else
        mayor1 = A[2];
        mayor2 = A[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
```

Analicemos por bloques:

El primer if consta de una comparacio, y 2 asignaciones.

Si no entra al if entonces tenemos que hacer la comparacion del if y 2 instrucciones.

Tenemos la asignacion que siempre se realiza.

Curiosamente nos cuesta lo mismo elejir uno u otro.

el while hará $n-2$ repeticiones de una compraracion.

Además tenemos dentro el while tenemos un if que consta de 3 instrucciones.

Ademas si entra tambien al segundo la primera comparación, mas la compraracion y la asignacion.

Tenemos 2 instrucciones dentro del while.

Regresa una operación.

3.1.1. Mejor caso

Es cuando los 2 primeros son el mayor y el segundo mayor de esta manera tenemos 3 instrucciones antes del while, y $2(n-2)$ instrucciones dentro del while.

Por tnto la complejidad es $2n-1$

3.1.2. Peor Caso

Cuando el arreglo esta ordenado de menor a mayor.

Tendremos 3 instrucciones iniciales y tendremos $3(n-2)$ dentro del while.

Por lo tanto la coplejidad será de $3n-3$

3.1.3. Caso medio

Tenemos al inicio 3 instrucciones, de ahi se deriva el caso de que entre al primer if dentro del while, sería igual a $3n - 3$ con el segundo es igual y si no entramos sería $2n - 1$

Entonces el caso medio será igual a :

$\frac{1}{3}(3n-3) + \frac{1}{3}(3n-3) + \frac{1}{3}(2n-1) + 3 = \frac{8n-7}{3} + 3$

3.2. Algoritmo 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
```

El if ejecuta 4 instrucciones. Si no se ejecuta es 1 instruccion.

El for interno se ejecuta n-i-1 veces.

El otro for se ejecuta n-1 veces.

3.2.1. Mejor caso

Cuando el arreglo esta ordenado ascendentemente.

El if no estra entonces tenemos una instrucción. La complejidad sería$n^2 + 2n + 1$

3.2.2. Peor caso

Cuando el arreglo esta descendentemente. El peor caso es 4 veces la del mejor caso.

$4n^2 + 8n + 4$

3.2.3. Caso medio

En este caso es el promedio del peor y el mejor caso entonces sería: $\frac{5n^2 + 10n + 5}{2}$

3.3. Algoritmo 11:

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

3.3.1. Mejor caso

Cuando n=0 nunca entramos al while entonces la complejidad es 0.

3.3.2. Peor caso

Entrara al while a/b. veces. Entonces asumiendo que n es el mayor y $m << n$ entonces será n.

3.3.3. Caso medio

Sera el peor caso. n.

3.4. Algoritmo 12:

```
func BurbujaOptimizada(A, n)
    cambios = "Si"
    i = 0
    Mientras i < n - 1 && cambios != "No" hacer
        cambios = "No"
        Para j = 0 hasta (n - 2) - i hacer
            Si A[j + 1] < A[j] hacer
                aux = A[j]
                A[j] = A[j + 1]
                A[j + 1] = aux
                cambios = "Si"
```

```
                Fin Si
            Fin Para
            i = i + 1
        Fin Mientras
Fin func
```

3.4.1. Mejor caso

Cuando A esta ordenado ascendentemente. Entrara al while solo una vez y la varicable quea en No. Entonces será $n - 1$

3.4.2. Peor caso

Cuando A esta ordenado descendentemente.

En el while habra cambio en la variable entonces la complejidad será $n^2 + 2n + 2$

3.4.3. Caso medio

La complejidad sera dependiendo del if. $\dfrac{2n^2 - n - 1}{2}$

3.5. Algoritmo 13:

```
func BurbujaSimple(A, n)
    Para i = 0 hasta n - 2 hacer
        Para j = 0 hasta (n - 2) - i hacer
            Si A[j + 1] < A[j] hacer
                aux = A[j]
                A[j] = A[j + 1]
                A[j + 1] = aux
            Fin Si
        Fin Para
    Fin Para
Fin func
```

El if ejecuta 4 instrucciones. Si no entra al if se ejecuta solo una.

El for de más dentro se ejcuta n-i-1 veces y el mas fuera se ejecuta n-1 veces.

3.5.1. Mejor caso

Cuando el arreglo está ascendente. $\sum_{i=0}^{n-2} 1 = \dfrac{n^2 - n}{2}$

3.5.2. Peor caso

Cuando el arreglo esta ordenado descentemente. Como el if tiene 4 instrucciones entonces la complejidad es: $2n^2 - 2$

3.5.3. Caso medio

Será: $= \frac{n^2 - n}{4} + \frac{2n^2 - 2}{2}$

3.6. Algoritmo 14

```
func Ordena(a, b, c)
    if(a > b)
        if(a > c)
            if(b > c)
                salida(a, b, c);
            else
                salida(a, c, b);
        else
            salida(c, a, b);
    else
        if(b > c)
            if(a > c)
                salida(b, a, c);
            else
                salida(b, c, a);
        else
            salida(c, b, a);
Fin func
```

Tenemos una serie de comparaciones.

3.6.1. Mejor caso

Cuando entra a la primera compraración pero no a la segunda, y cuando se entra al primer else y al segundo else. La complejidad en este caso es $= 2$.

3.6.2. Peor caso

Cuando estra a algun otra de las 4 combinaciones es de $= 3$

3.6.3. Caso medio

Es la probabilidad de cada caso que es $= \dfrac{8}{3}$

3.7. Algoritmo 15:

```
func Seleccion(A, n)
    Para k = 0 hasta n - 2 hacer
        p = k
        Para i = k + 1 hasta n - 1 hacer
            Si A[i] < A[p] entonces
                p = i
            Fin Si
        Fin Para
        temp = A[p]
        A[p] = A[k]
        A[k] = temp
    Fin Para
Fin func
```

El for mas dentro se realiza $n - k - 1$ veces, el if que esta dentro cuesta 1 instruccion.

Tenemos otro for que se realiza n-1 veces, despues del for hace 3 opeaciones en caso de que entre al if o no.

3.7.1. Mejor caso

Cuando el arreglo esta ordenado ascendentemente.

Entonces: $\sum k = 0^{n-2}(3 + \sum i = k + 1^{n-1} 1) = (n+2)(n-1) - \dfrac{(n-2)(n-1)}{2}$

3.7.2. Peor caso

Cuando el arrego esta ordenado descendemente. Curiosamente es la misma complejidad aunque la condicion del if siempre se cumpla. Entonces es la misma complejidad:

$= (n+2)(n-1) - \dfrac{(n-2)(n-1)}{2}$

3.7.3. Caso medio

Como el peor y el mejor caso es el mismo, el caso medio es $= (n+2)(n-1) - \dfrac{(n-2)(n-1)}{2}$